Project Number:	720270	Project Title:	Human Brain Project

Document Title:	Collaboratory Architecture
Document Filename[(1)]:	SP11 D11.3.1 FINAL_Resubmission
Deliverable Number:	D11.3.1
Deliverable Type:	Platform Architecture Document
Work Package(s):	WP 5.5, WP7.5, WP11.2, WP11.3
Dissemination Level:	PU
Planned Delivery Date:	M 6 / 30 Sept 2016
Actual Delivery Date:	12 Oct 2016, Resubmission 31 May 2017

Authors:	Jeffrey MULLER (EPFL-P1), Chris EBELL (EPFL-P1), Martin TELEFONT (EPFL-P1)
Compiling Editor(s):	EPFL (P1): Jeffrey MULLER
Contributors:	EPFL (P1): Selim ARSEVER, Jean-Denis COURCOL, Mike GEVAERT, Daniel PEPPICELLI, Luis RIQUALME, Felix SCHÜRMANN, Werner VAN GEIT, Stefano ZANINETTA, Martin TELEFONT, Guy WILLIS, Marc-Oliver GEWALTIG JUELICH (P20): Thomas LIPPERT, Anna LÜHRS, Timo DICKSCHEID ETHZ (P18): Thomas SCHULTHESS, Colin MCMURTRIE UMAN (P63): David LESTER UIO (P81): Jan BJAALIE, Dmitri DARINE
Reviewers:	EPFL (P1): Colin MCKINNON

Abstract:	
Keywords:	

Table of Contents

List of Figures and Tables

Executive Summary

There are six Information and Communications Technology (ICT) Platforms that will be developed by the Human Brain Project (HBP). The HBP Platforms will be made accessible to scientific, medical and engineering researchers over the internet via an HBP Collaboratory (HBP-COLL or COLL). The Neuroinformatics Platform (HBP-NIP or NIP) plays a crucial role in guaranteeing traceability and discoverability in the COLL. This is essential in the upcoming phase of the Project to ensure a solid ecosystem for data sharing and software and to enable application of that data to scientific problems. The HPAC Platform provides the computational resources, storage and networking necessary for key data and modelling use cases and must also be well integrated with the NIP and the COLL.

The purpose of this document is to set out the architecture for the COLL, along with clear architecture for its integration with the NIP and HPAC Platform.

The NIP, on the one hand, will be deeply integrated with the COLL in SGA1 in order to ensure an effective ecosystem for data sharing and software, enabling application of that data to scientific problems. On the other hand, SP7's HPAC Platform provides the computational resources, storage and networking necessary for both the primary archive platform for the HBP as well as the management, analysis, transport and storage capabilities along, with the federation of very large data sets required for key data and modelling use cases. The HPAC Platform thus needs to be deeply integrated with the NIP and the COLL as well. The linkages between the COLL and ongoing HPAC initiatives highlights the importance of these planned integrations.

1. Introduction

The Human Brain Project (HBP) is a 10-year research project, funded by the European Commission (EC), to lay the foundations for a new approach to brain research. Neuroscience, medicine and information technology each have important roles to play in addressing this challenge, but their contributions are currently fragmented. The HBP will integrate these inputs and catalyse a community effort to achieve a new understanding of the brain, new treatments for brain disease, and new brain-like computing technologies.

It is a central tenet of the HBP strategy that a comprehensive understanding of the brain requires knowledge of structure and function across all levels of brain organisation; this understanding cannot be achieved at any one level alone. To achieve this understanding, interdisciplinary expertise joining neuroscience, computer science, physics and mathematics is key. A massive scientific collaboration is required to reconstruct such multi-level models. The social internet and open source software communities have shown that modern Information and Communications Technology (ICT) permits the massive collaborative efforts needed.

To facilitate the scientific community's access to the HBP's ICT Platforms and, in a broader sense, to make large-scale collaborations possible in neuroscience, the HBP is developing the HBP Collaboratory (HBP-COLL or simply COLL). This web-based collaborative scientific platform provides access to the HBP's research, community and administrative activities, as well as its six ICT Platforms. A tool within the COLL of particular importance is a deeply integrated search developed in the Neuroinformatics Platform (NIP).

The COLL is to be equipped with a layer of social networking functions to allow fluid sharing of data, theories, applications and models prior to publication, while still maintaining proper attribution. This social networking framework will be expanded throughout the operation of the HBP and has already enabled the inclusion of researchers outside the HBP in Collaborative HBP activities. This sharing of research, results and expertise should help to accelerate neuroscience and the achievement of the HBP's ambitious goals.

This document sets out the architecture for the COLL with specific focus on integration of the COLL with NIP and HPAC Platform's respective components. It is intended for a technical and scientific readership. The document focuses on the functionality that the COLL will provide and the use cases it will serve. It describes the ways in which web-based platform components will interact with other system. It also provides details of how the COLL and HPAC Platform are integrated, highlighting where this is different from the currently available COLL system.

2. The HBP Collaboratory (HBP-COLL)

This section presents the HBP Collaboratory. It starts with the requirements that informed the design process and the relationship to other Platforms, before dealing with the architecture of the COLL and its components. Where necessary, it describes in technical detail the architecture of planned integrations between the COLL and the NIP and HPAC Platforms.

2.1 HBP-COLL: Overall Goals

The COLL is a web-based portal intended to provide a single point of access to collaborators participating in all research activities in HBP. The COLL will allow scientists from around world to collaboratively:

- Gather and organise multi-level neuroscience data (via the NIP)
- Reconstruct, validate and refine multi-level brain models at different levels of fidelity (via the Brain Simulation Platform [BSP])
- Search, analyse and cluster distributed clinical data (via the Medical Informatics Platform [MIP])
- Develop and access interactive supercomputing (via the High Performance Analytics and Computing [HPAC] Platform)
- Configure, train and operate neuromorphic computing systems (via the Neuromorphic Computing Platform [NCP])
- Couple brain models to virtual agents acting in virtual environments to perform *in silico* cognition and behaviour experiments (via the Neurorobotics Platform [NRP]).
- crowdsourcing of literature mining

Underneath the web-based portal the COLL is designed to support novel collaborations by providing a Service-oriented architecture (SOA) which supports:

1) instantaneous sharing of data, models, tools, theories, configurations, methods and applications,
2) tracking and crediting researchers for their contributions (provenance),
3) launching of collaborative projects with various levels of access control and user-defined tool sets.

The COLL will also be the primary means by which the HBP shares its scientific and technological advances with the scientific, medical and engineering communities. The COLL will therefore provide a platform for Strategic Partners to present their research projects, form collaborative projects, known as *collabs*, with other members of the Flagship Consortium, and share their progress in advancing or using the HBP ICT platforms. Finally, the current COLL implementation also allows researchers outside the HBP to join and create *collabs*. In its current implementation, it provides a software and service catalog for the HBP. In the next phase a universal search for *collabs*, software, service, documentation and data will be provided, thanks to the planned NIP integration.

The COLL is an open and standards-based architecture. As such, it is capable of integrating platforms for European and international collaborations, depositing and licensing IP, subscribing to HBP services, and developing knowledge streams, and also for accessing and managing educational services, providing feedback for Responsible Research and Innovation (RRI), and general administration and management of the HBP. The COLL will be designed to scale to very large numbers of researchers in science, medicine and engineering, providing a novel virtual environment for distributed, collaborative and multi-disciplinary research and development.

To achieve this goal, the COLL must:

4) Serve both technical Power Users and non-technical Casual Users,
5) Facilitate simple access models for high performance computing (HPC) and neuromorphic computing resources,
6) Facilitate simple discovery and access for rich multi-modal data sets.

In most cases the modellers, theoreticians and computer scientists are the ones building tools for inclusion in the COLL. These are the *Power Users*. However, it is often difficult to validate or use those tools without input or validation data. Biologists need computational

tools in their toolkit, but those tools must be easy to use. These are the *Casual Users*. It is necessary to serve both *Power Users* and *Casual Users* because the goals of the HBP can only be met by active collaboration between biologists, theoreticians, modellers and computer scientists.

Neuroscientific data are among the most complex (due to their multidimensionality, ontologies, and formats), and opaque in science. Part of the problem in understanding the human brain is finding and organising the data so that they can be searched and used to build models and other applications. By providing web-based tools for searching, viewing and analysing rich neuroscientific data sets, the HBP will shorten the distance between experiment and discovery and make possible worldwide data-centric collaboration.

In the current project phase (SGA1), the HBP has an increased awareness of the Platform linkages required to achieve goals 1-3 above. It is for this reason that this document contains extensive documentation of the architecture required to integrate the HPAC Platform and the NIP.

2.2 HBP-COLL: Refined Use Cases

The Use Cases below describe success scenarios for small numbers of actors. The scenarios describe high-level interaction with the HBP-COLL, NIP and HPAC Platform and their underlying services.

Each Use Case described in this document is attributed a unique identifier. "HBPCOLL" indicates that it relates to the COLL. "UC" indicates a Use Case, while "FR" denotes a Functional Requirement.

2.2.1 Roles

- Computational Scientific User (CSU) – A User with scientific development skills, comfortable launching command line HPC jobs.
- Biological Scientific User (BSU) – A User with scientific expertise, but limited technical computing skills.
- Scientific User (SU) – A scientific User, either a CSU or a BSU.
- Scientific Developer (SCIDEV) – A User who is developing software to directly realise the scientific objectives. This User usually works in close collaboration with scientists, both CSUs and BSUs.
- Developer (DEV) – A User who is developing software to realise engineering, operational and/or scientific objectives.
- Portal User (PU) –A User who accesses Platform functions through the Web GUI.
- Service User (ServU) – A User who accesses Platform functions through a programmatic Service Client API.
- Infrastructure Personnel (INFRA) - An Infrastructure System Administrator or Developer, typically responsible for deploying and monitoring Platform services that are offered directly to customers.

2.2.2 Low Volume Scientific Data Sharing, Single COLL Project with Upload (HBPCOLL-UC-001)

Abigail needs help analysing some morphologies so she creates a COLL Project to collaborate with another scientific User in a shared workspace.

Primary Actors: Two Scientific Users, Abigail and Bill.

Success Scenario:

1) Abigail has morphology geometry data on her local machine or on a public and permanent internet-accessible URL.
2) She creates a COLL Project, COLL Project 1, in the Portal to hold the morphologies. Abigail is the owner of COLL Project 1.
3) She uploads the morphology files or their URLs to COLL Project 1. The COLL will request that she add MINDS metadata (see Glossary) to the Artefacts on upload.
4) If she decides not to add MINDS metadata to any uploaded data, the UI will display those data files differently.
5) Some portions of the MINDS data can be extracted from the data, if the data type is known to the COLL.
6) If she attempts to use uploaded data that are not MINDS annotated in COLL analysis Tasks, she will be required to add MINDS metadata before the analysis is launched.
7) Abigail can also add a simple wiki entry to her collab and link it to the uploaded morphology. This wiki entry will help to give context to the uploaded morphology to help Users find it from the COLL search functionality.
8) A background process will index COLL Project 1 and all of its metadata into the NIP Search.
9) She adds Bill to the COLL Project team and gives him read and write access to the COLL Project.
10) Bill can now download (COLL Project read) or use Portal Tasks (COLL Project read and write) to analyse and model with the morphologies.

2.2.3 Low Volume Scientific Data Sharing, Multi COLL Project (HBPCOLL-UC-002)

Abigail and Chris created a COLL Project in a previous Use Case and now Bill needs additional help to review the output of one of his analysis. However, he doesn't want Chris, whom he asked for help, to see the original COLL Project.

Primary Actors: Three Scientific Users, Abigail and Bill and Chris.

Precondition:

- Abigail has created COLL Project 1 with Bill as described in HBPCOLL-UC-001.

Success Scenario:

1) Bill creates a new COLL Project, COLL Project 2.
2) Bill creates a COLL link, denoted L, in COLL Project 2 to an analysis output data file, denoted A, in COLL Project 1.
3) Bill then adds Chris to the COLL Project 2 as a COLL Project Administrator.
4) Bill then performs analysis on L, specifying COLL Project 2 as the output directory. The output of the analysis on L is denoted A'.
5) Chris (or any other User with read permissions in COLL Project 2) is able to see A'.

6) Users with read permissions in COLL Project 2 are not able to see the contents of A' (i.e.: A) unless they have been added to Abigail's COLL Project as a reader, or unless Abigail has made the data public to one of the sharing groups that Chris is a part of.

2.2.4 Data Release (HBPCOLL-UC-004)

Abigail has some data that she wants to share with a larger community.

Primary Actors: Two Scientific Users, Abigail and Bill.

Precondition:

- Abigail has created a COLL Project 2 with Bill as described in HBPCOLL-UC-001.

Success Scenario:

1) Abigail is confident that her latest cerebellum model in COLL Project 2 is a significant improvement over previous cerebellum models. She wants to make her model available to others to run simulations on, analyse and refine.
2) Abigail selects the folder in her COLL Project that contains the cerebellum model, named "model" and presses the Release button.
3) She is warned that the Release is an irreversible action and that her COLL Project will become read-only.
4) She selects a name for the new Release, "Cerebellum Release 1".
5) She is prompted to update the permissions of the Released model as a convenience. Abigail upgrades the visibility of the new Release to allow everyone in the HBP to see it. This grants access to all data in her new Release to anyone in the HBP.
6) The "model" folder is moved to the new release entity named Cerebellum Release 1. The original folder location in COLL Project 2 is replaced with a link the model folder in Cerebellum Release 1.
7) Releases can be published to the Knowledge Space if required.

2.2.5 Portal Developer Services and Component Reuse (HBPCOLL-UC-005)

Catherine wants to extend the data visualisation capabilities of the Portal.

Primary Actor: One Portal Developer, Catherine.

Success Scenario:

1) Portal Developer User Catherine has a Web UI extension to the HBP Portal that she would like to implement.
2) Catherine's application will be divided into HTML5 client-side logic and a REST service implemented in Python or Java.
3) For the browser client side of the application, Catherine will be able to take advantage of any HTML5 libraries she wants, or she can use the Angular widgets produced by the HBP COLL Portal team. She will then create the client side portion of her application. She will integrate the client-side connector library to allow her application to talk the HBP COLL Portal container.
4) For the server side of the application, Catherine will use a standardised authentication library to authenticate Users of her application securely against an HBP Central Authentication Service. The integration with the HBP Authentication Service will allow her application to access COLL REST APIs on behalf of the authenticated User.

2.2.6 Viewing of Data in HBP-COLL (HBPCOLL-UC-006)

Abigail is interested in some data that are found through a search interface, in a COLL Project, or that are viewed from a site supporting the COLL authentication data. She wants to view those data using a rich visualisation.

Primary Actor: Abigail, a Scientific User.

Success Scenario:

1) Abigail has used the Search App to find morphology data.
2) Abigail has the option to view her morphology data using an image rendering, a 3D geometry viewer or a provenance viewer, showing where the data came from.
3) She can add the data to a COLL Project where she can use the same viewer options to view the data.
4) The available viewers will be filtered based on semantic content types and can be used in Web UIs throughout the various platforms.

2.2.7 Collaborative Scientific Analysis (HBPCOLL-UC-007)

Bill wants some help analysing data. He recruits Abigail to his COLL Project and then shares the results with Chris.

Primary Actors: Three Scientific Users: Abigail with strong software development skills, and Bill and Chris with strong biological skills. Abigail and Bill are working on the same COLL Project. Chris does not share any COLL Projects with Abigail and Bill.

Success Scenario:

1) Bill has morphology geometry data in a COLL Project that he would like to understand quantitatively, but he needs additional expertise to write the analysis software and to analyse the data.
2) He adds Abigail to the COLL Project team and gives her access to the morphologies.
3) Abigail can now use Jupyter notebooks to analyse data and answer questions for Bill.
4) Abigail or Bill can execute notebooks, new and previously existing, on the data they share in the collab.
5) If Abigail decides that her new analysis notebook is useful to others, she can use the Portal (through UI or service interface) to make their analysis public for others to run.
6) Chris, who also works with morphologies, can now use Abigail's analysis notebook. The Portal tracks Chris's output data and knows that Abigail's analysis was used to generate it.

2.2.8 Scientific Developer Iterative Workflow Development (HBPCOLL-UC-008)

Daniel needs to update a Jupyter notebook workflow to integrate a new type of data constraint.

Primary Actor: One Scientific Developer, Daniel.

Success Scenario:

1) Scientific Developer User Daniel would like to add the use of synthesised morphologies to the circuit building workflow.

2) Daniel will download the notebook code for the workflow. He will also be able to download the container or VM definition for the COLL-hosted Jupyter notebook so he can run his local analysis with the same software dependencies
3) Finally, he modifies the circuit building workflow to include the synthesis in the appropriate location.
4) Once he has tested his workflow and container description locally, Daniel will upload and share his Jupyter notebook with other COLL users.

2.2.9 Visualisation Developer Component Reuse (HBPCOLL-UC-009)

A CAVE Visualisation Developer, Elisabeth wants to find data to view in her application.

Primary Actor: A Visualisation Developer, Elisabeth.

Success Scenario:

1) Visualisation Developer User Elisabeth would like to find cellular models for visualising inside her CAVE application.
2) Elisabeth's application uses the NIP Search API to search for cellular models matching certain metadata queries.
3) The application then uses the Collab Storage Service API to find a local path from which her application can load data.
4) If the data for the cellular model are not available on a locally accessible storage resource, the application can use the Data Transfer API to move the data from the remote location to a locally accessible storage resource.
5) The application can use standard filesystem APIs to load and visualise the cellular model in the CAVE.
6) If needed, her application can also use the Knowledge Graph REST client API to find source data from which a particular cellular model was constructed.

2.3 HBP-COLL: Functional Requirements

2.3.1 Authentication and Authorisation (HBPCOLL-FR-001)

1) Users must be authenticated against a central database.
2) Users must have access control based on COLL Project-specific groups of Users.
3) The Portal and its applications will be accessible only through the SSL protected https:// or wss:// protocols.
4) Single sign-on will allow a User to log in once for most applications.
5) Certain applications may require re-authentication to perform some privileged operations.
6) Authentication will be standards-based.
7) The Platform will allow limited delegation, i.e. the User will be able to restrict services in the Platform constellation from accessing certain services on their behalf.
8) Access to COLL Projects, Artefacts, Parameters, Tasks, Workflows and their metadata will be controlled by the authorisation system.

2.3.2 Search (HBPCOLL-FR-002)

1) Artefacts uploaded to the COLL will be searchable by diverse metadata.
2) Artefacts produced by the COLL will be searchable by diverse metadata.
3) Artefacts in the COLL will be searchable across the COLL and NIP.
4) Parameter contents will be searchable.
5) COLL Project wiki descriptions will be full-text searchable.
6) The search service must provide a common search API for searching NIP and COLL Databases for relevant metadata.
7) The search service must return the first page of search results in less than five seconds. Since it will take a series of incrementally refined searches to find a User's desired data, searches taking longer than five seconds to return the first page of 10-25 results will greatly reduce search utility.
8) Both the Search App and REST API must honour the ACLs of the services that have been indexed.

2.3.3 COLL Projects (HBPCOLL-FR-003)

1) It will be possible for the Portal User or Service User to attach a wiki to Entities.
2) Permissions are COLL Project-wide.
3) Read permissions are required to read in from the COLL Projects. Some metadata from the COLL Projects will be globally visible to enable public discovery. Read permissions are granted by assigned COLL Project Administrators.
4) Write permissions are required to add data, link data and modify metadata. A COLL Project Administrator grants Write permissions.
5) Administration permissions are required to modify User COLL Project permissions. A COLL Project Administrator grants Administration permissions.
6) The COLL must provide a mechanism for COLL Project Administrators to modify COLL User permissions for the COLL Projects that the Administrator owns.
7) The COLL must provide a mechanism for a User to create, move, copy and link a User readable data file in one COLL Project to a User readable data file in another COLL Project.
8) There is a COLL Storage Viewer that embeds content type-specific viewers.
9) The COLL Storage Viewer must provide controls for managing the Storage and Data Lifecycle.

2.3.4 Storage and Data Lifecycle Management (HBPCOLL-FR-004)

1) Users must be identifiable and have access permissions on all services in the Portal.
2) There must be a metadata service with functionality for deleting data.
3) The COLL expects to use the FeDaPP/FENIX Storage for storage of various low-density data files.
4) The COLL will use services in the FeDaPP/FENIX Storage API for accessing data that are not available in a local storage resource.
5) File storage in the collab should be allocated from a per-COLL Project storage quota.

6) File storage in FeDaPP/FENIX using the COLL Storage Browser app will be governed by HPAC site storage quotas. The COLL expects to delegate the management of storage quotas to the HPAC Platform and its per-site implementation.
7) There will need to be an FeDaPP/FENIX service that facilitates the reliable transfer of data between sites using high performance bulk transfer protocols.
8) The transfer service will need to be available to the COLL. This requirement suggests that the transfer service should be JSON REST-enabled. This does not mean that the actual transfer is done over REST, but that the transfer is initiated and managed over REST.
9) The FeDaPP/FENIX Transfer service should enforce quotas of limited transfer resources. The COLL expects to delegate the management of transfer quotas to FeDaPP/FENIX.

2.3.5 Common Service Interfaces (HBPCOLL-FR-005)

1) All services must use the HBP LDAP as their source of truth for providing User metadata and for authenticating Users. This authentication can be exposed through one of the authentication mechanisms described in Sections 2.3.2 and 2.3.3.
2) All services—web or otherwise—must log their system logs through the system-wide syslog Logging Service.
3) All Services must provide a method to determine service health for the system-wide Monitoring Service.

2.3.6 Web Services (HBPCOLL-FR-006)

1) Web Services will offer a REST binding using JSON.
2) REST services must conform to Platform standards.
3) REST services must provide web accessible documentation in a standard location.
4) REST services must have Platform-provided client libraries for Python.
5) Access to REST services must be authenticated via OpenID-Connect that uses HBP LDAP as its source of authentication authority.
6) Web Service Interfaces must log API accesses. Ideally this will provide per-User accounting.

2.3.7 Non-Web Services (HBPCOLL-FR-007)

1) Public facing non-Web Service interfaces must be authenticated with one of OpenSSH keys, Kerberos or X509 client certificates.

2.4 Components

Current URL: https://collab.humanbrainproject.eu

Development Standard: BBP Standard -> HBP Standard

Depends on Services: HBP Identity Service

Current TRL: TRL7 (minus SLA definition)

SGA1 Target TRL: TRL7 (minus SLA definition)

Further Details: TBD

2.4.1 App Components

2.4.1.1 COLL Content Apps

Description: A collection of Apps used to add, edit and view content in the COLL.

Current URL: https://collab.humanbrainproject.eu

Development Standard: BBP Standard -> HBP Standard

Depends on Services: HBP Identity Service, COLL Services

Current TRL: Mixed, TRL6-8 (minus SLA definition)

SGA1 Target TRL: TRL7+ for all (minus SLA definition)

Further Details:

The COLL provides a number of types of content pages which can be used by Collab teams to explain and contextualise the work they present in their *collabs.*

Currently there are 3 types, each with different use cases and user bases:

1) Markdown – Simple text markup. This is favoured by many technical users for efficient editing of content without having to worry about presentation.
2) Richtext – WYSIWYG editing of HTML content. This is favoured for cases where presentation is more important or the user is less technical.
3) Live documents – Realtime collaborative documents with simple formatting. This was recently added to support collaborative editing of content and being used in a wide number of use cases.

2.4.1.2 COLL Storage App

Description: Provides simple access to web based storage

Current URL: https://collab.humanbrainproject.eu

Development Standard: BBP Standard

Depends on Services: HBP Identity Service, Collab Storage Service

Current TRL: TRL7 (minus SLA definition)

SGA1 Target TRL: TRL8 (with SLA definition)

Further Details:

The COLL Storage App provides simple access to web-based storage associated with each collab. The system provides upload, download, rename, delete and move operations. Under this model ACLs are provided in a simple per-collab membership list.

In addition, early support is available for browsing FedAPP/FENIX storage using the same interface. In this mode the Collab Storage App defers ACLs to UNICORE and the per-site configuration.

Both backends, Collab storage and UNICORE are accessed through a standard API provided by the Collab Storage Service.

2.4.1.3 COLL Chat App

Description: Integrated Chat functionality for the COLL

Current URL: https://collab.humanbrainproject.eu

Development Standard: BBP Standard

Depends on Services: HBP Identity Service

Current TRL: TRL6

SGA1 Target TRL: TBD – alternative chat approaches are being considered to maximise utility of chat functionality while leveraging existing software effectively.

Further Details:

The COLL has an integrated chat system in the version released at the end of the Ramp-Up Phase. Each collab gets its own chat room. Chat is writable by anyone who can view a collab. This allows people who are not members of a public collab to interact with collab owners, while still restricting write permission on public collab content to collab members.

2.4.1.4 COLL Jupyter Notebook App

Description: Integrated Chat functionality for the COLL

Current URL: https://collab.humanbrainproject.eu

Development Standard: BBP Standard

Depends on Services: HBP Identity Service, conditional dependencies on other services depending on the notebook in question.

Current TRL: TRL6

SGA1 Target TRL: TRL8

Further Details:

This app integrates the web interface of a popular web-based collaborative software development tool, known as the Jupyter notebook, into the COLL. Interface modifications have been made to facilitate COLL-friendly editing, upload and download.

See the Jupyter HowTo Collab here:

https://collab.humanbrainproject.eu/#/collab/509/nav/11881

2.4.2 Service Components

2.4.2.1 COLL Service

Description: Service storing per-collab metadata

Current URL: https://collab.humanbrainproject.eu

Service API documentation: https://collab.humanbrainproject.eu/#/collab/54/nav/1878

Development Standard: BBP Standard

Depends on Services: HBP Identity Service

Current TRL: TRL7 (minus SLA definition)

SGA1 Target TRL: TRL8 (with SLA definition)

Further Details:

The goal of the COLL Service is to provide metadata and ACL information to support the Collab UI via a REST API. It is used as an authorisation mechanism for many Platform apps and services.

2.4.2.2 HBP Identity Service

Description:

Current URL: https://collab.humanbrainproject.eu

Service API documentation: https://collab.humanbrainproject.eu/#/collab/54/nav/4853

Development Standard: BBP Standard

Depends on Services: HBP Identity Service

Current TRL: TRL7 (minus SLA definition)

SGA1 Target TRL: TRL8 (with SLA definition)

Further Details:

Authentication is the act of confirming a User's identity by requesting a piece of information that only they should be able to provide. This can be a password or a number derived from a private key via a cryptographically irreversible process.

The COLL will include an Authentication Service that implements the OpenID-Connect standard to authenticate REST APIs and Web applications. This standard is used to allow web or native application Single Sign-On (SSO) for applications using REST APIs provided by the various HBP Platforms.

More information on OpenID-connect can be found here: http://openid.net/connect/.

2.4.2.3 COLL Storage Service

Description: REST API for accessing Storage through various backends

Current URL: https://collab.humanbrainproject.eu

Development Standard: BBP Standard

Documentation: https://collab.humanbrainproject.eu/#/collab/54/nav/2412

Depends on Services: HBP Identity Service

Current TRL: TRL7 (minus SLA definition)

SGA1 Target TRL: TRL8 (with SLA definition)

Further Details:

COLL Storage provides a REST service to access storage space in each collab which honours the ACLs of the collab. Collab members can read, write and rename files and their metadata. Both backends, Collab storage and UNICORE are accessed through a standard API provided by the Collab Storage Service.

2.4.2.4 COLL Jupyter Notebook Service

Description:

Current URL: https://collab.humanbrainproject.eu

Development Standard: BBP Standard

Documentation: https://collab.humanbrainproject.eu/#/collab/509/nav/11881

Depends on Services: HBP Identity Service

Current TRL: TRL5

SGA1 Target TRL: TRL7 (without SLA)

Further Details:

This app integrates the web interface of a popular web-based collaborative software development tool, known as the Jupyter notebook, into the COLL. Service modifications

have been made to facilitate HBP Identity Service integration, COLL-friendly editing, upload and download.

The service currently provides a Jupyter kernel with a standard software suite based on user requests: https://collab.humanbrainproject.eu/#/collab/509/nav/12747.

The system is currently based on the JupyterHub multi-user server system:

https://github.com/jupyterhub/jupyterhub

Documentation: http://jupyterhub.readthedocs.io/en/latest/

REST API Documentation:

http://petstore.swagger.io/?url=https://raw.githubusercontent.com/jupyter/jupyterhub/master/docs/rest-api.yml#/default

2.4.2.5 COLL XMPP Service

Description:

Current URL: https://collab.humanbrainproject.eu

Development Standard: BBP Standard

Documentation:

Depends on Services: HBP Identity Service

Current TRL: TRL6

SGA1 Target TRL: TBD – under consideration

Further Details:

A customised ejabberd implementation is used as the XMPP service behind the COLL Chat system. This is also used to power the notification system. Customisations were required for:

- Notifications
- Collab automatic chat groups
- HBP Identity authentication

2.5 HBP-COLL: Architecture

2.5.1 Architectural Principles

- Service Oriented Architecture – the COLL provides standard APIs to support a Service Oriented Architecture ecosystem. These are typically versioned REST APIs to support COLL integration and the handling of common authentication, authorisation, metadata and data operations.
- Permissions model – the COLL permissions are allocated in a very coarse-grained manner, and are intended to be uniform across all files in a particular COLL Project. Permission sets available are:
  - Public collabs:

    Members can read and write any authenticated user can read.

    Member can invite other COLL users to the collab.

    Visible in the “All Collabs” list.

  - Private collabs:

    Members can read and write; no one else can read or write.

    Members can invite other COLL users to the collab.

    Only visible to members of the collab in the “All Collabs” list.

  - Currently members are added one by one. There is a group mechanism in the HBP Identity Service. Adding and removing these groups will eventually be supported as members in the collab member list.

  - The permissions are opt-in for authorisation. App providers are free to choose their own authorisation model and may choose to use some or all of the collab permissions model or semantics in the app they provide.

- Provenance Tracking – Provenance tracking is the discipline of tracking where things come from and where they’ve gone. Historically in the COLL, this is the tracking of software versions and software execution environments and their data inputs and outputs. Provenance is used to establish reproducibility of computational Tasks, and to attribute credit for data production and tool development. This work is being revisited in SGA1 with a planned integration between the COLL and the NIP. Both implement a superset of PROV-DM to track links between Entities, Activities and Agents.

- Components – The COLL can be extended with the following components

  - Apps - web GUIs for certain services integrated into the COLL. In most cases these components will also use COLL services.

  - Services - web services, network file systems, SSH, source control (git), continuous integration, databases, configuration and deployment services. Software-as-a-Service, Platform-as-a-Service and Infrastructure-as-a-Service offerings.

  - Software - analysis libraries, simulators, and data access libraries, as well as thick client applications (desktop visualisation tools).

The COLL architecture is such that operators of services can deploy their services wherever they like while still preserving key integration characteristics inside the COLL ecosystem. This is primarily a consequence of architectural decisions of the HBP Identity service as well as the decision to provide a Microservice-based Service Oriented Architecture.

As a result, there is no coupling between deployment sites for the Apps visible in the COLL. The ensures that services and Apps can evolve independently and it allows a Federation of App providers across Europe. This can already be seen emerging in the Platform offerings demonstrated by the HBP at the end of the Ramp-Up Phase.

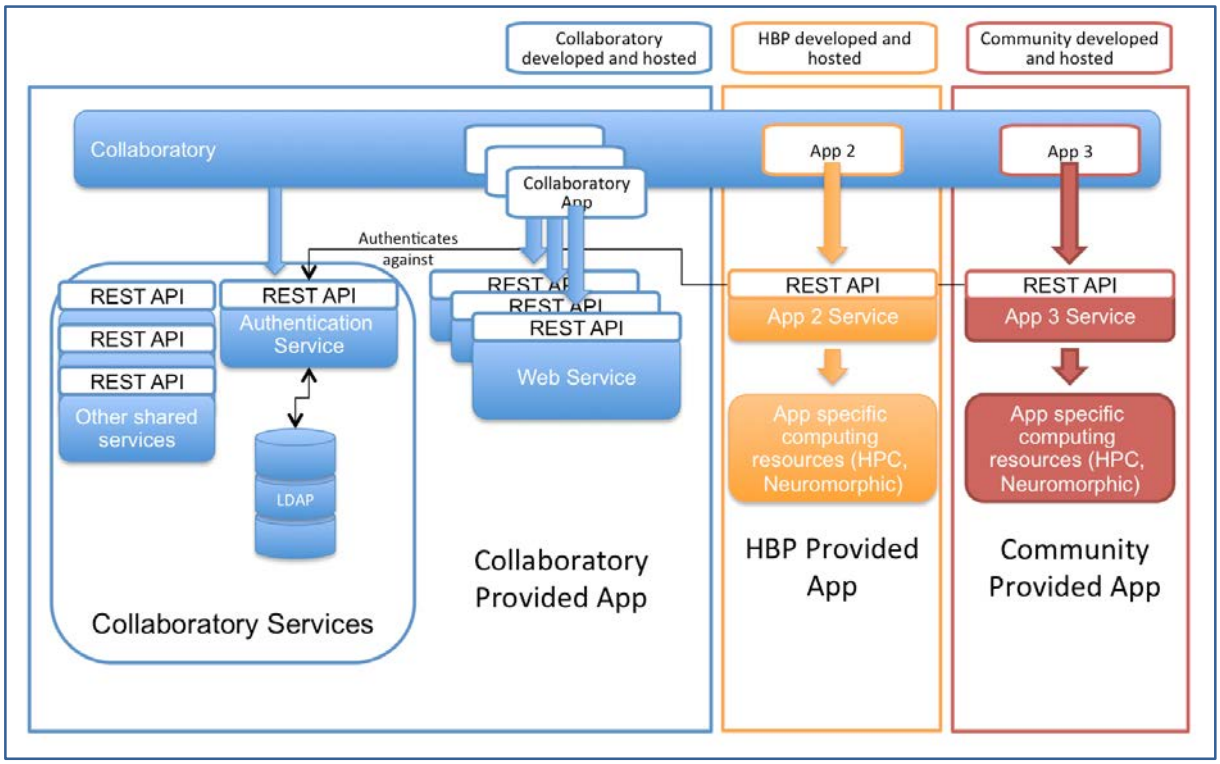


Figure 1: App Infrastructure Relationship

2.5.2 Standard App Component

For developers wishing to extend the COLL, App components are intended to allow them the ability to write standard web applications using any client-side HTML functionality, and any backend service. Typically, these apps would also integrate with several key COLL services, such as Authentication, Monitoring and Logging. COLL Apps would also be integrated with other COLL services. However, the level of integration with COLL services is entirely the decision of the App developer and should be decided on the basis of the App's provided functionality.

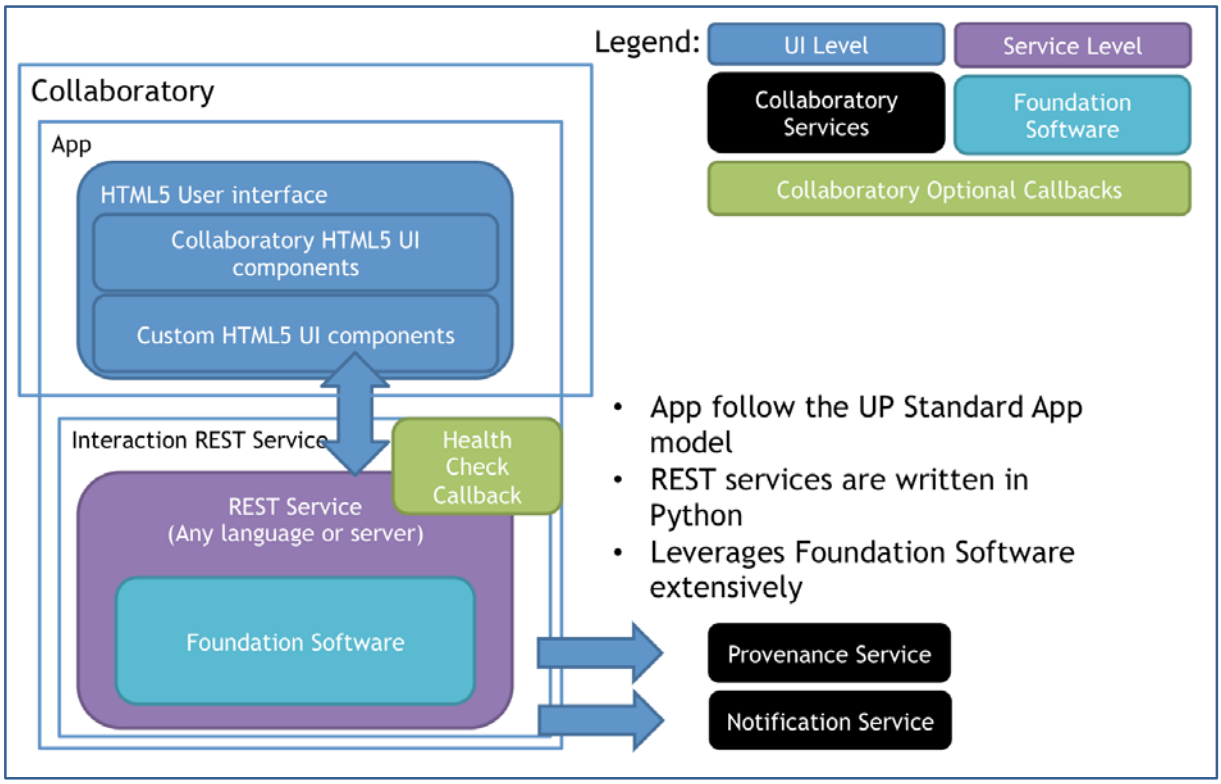


Figure 2: COLL App Architecture

The COLL Authentication system and Service-Oriented Architecture (SOA) allows loose coupling of HBP and third party apps to the COLL. This same approach also allows the Apps and their supporting web service to be deployed to any Base Infrastructure. This is key to the long-term federation strategy which will be developed in the Operational Phase.

2.5.3 Standard REST Web Service API

A COLL standard REST API has the following properties:

1) Python REST — implemented using BBP Tornado standard worker pooling behind an NGINX frontend.
   a) For database interaction, SQLAlchemy or Django persistence models are preferred.
2) Python, Java or Scala REST services include standard Rest API documentation, and a Python API used in service integration and load testing.

Official COLL REST APIs are expected to support client authentication using OpenID-Connect access tokens.

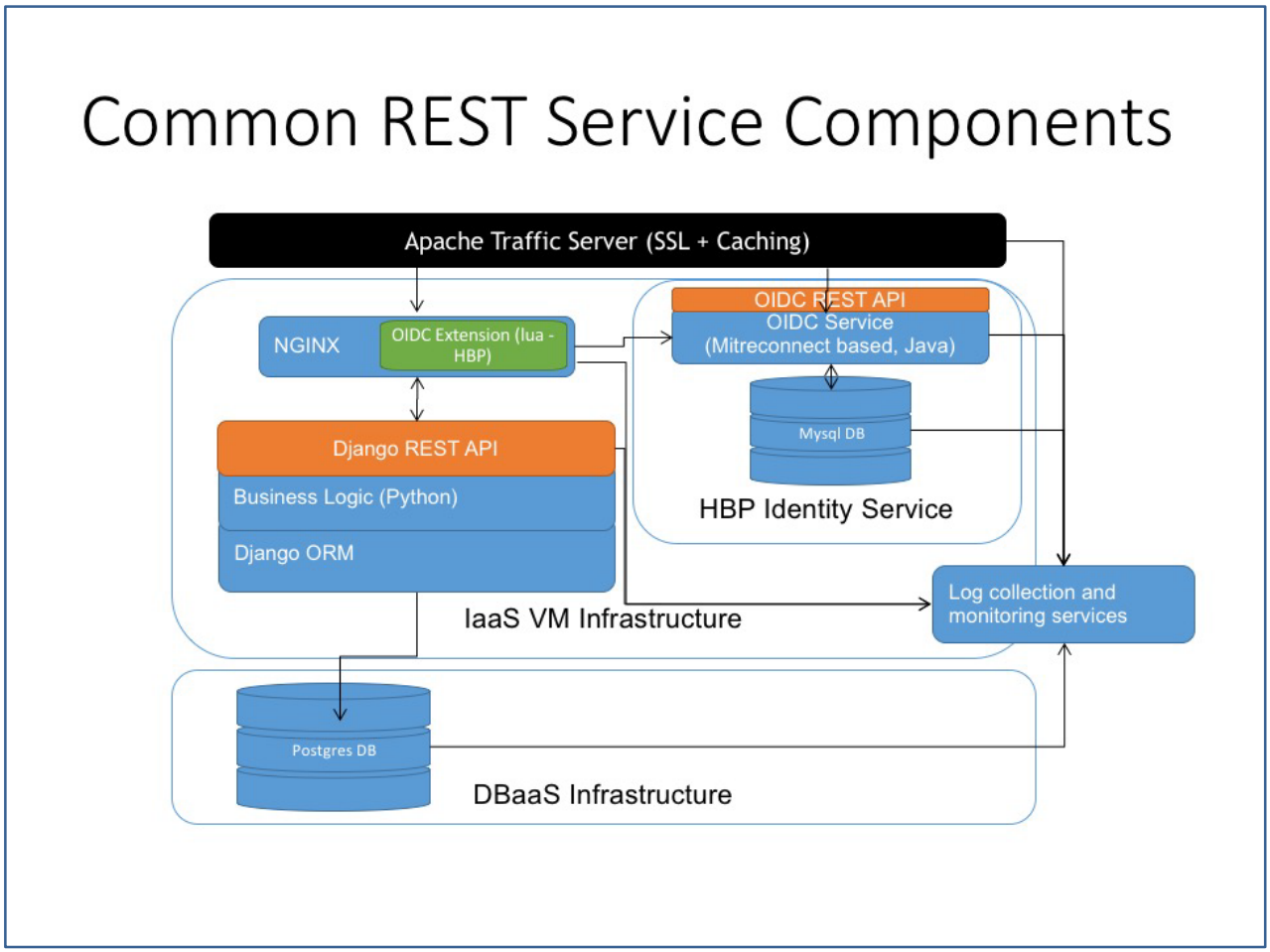


Figure 3: Common REST service architecture

The diagram above shows three main types of infrastructure needed to support a Service Oriented Architecture. They are:

1) PaaS – Proxy servers, Log collection and monitoring services, Databases (or DBaaS)

2) IaaS – Virtual machines, storage and network configurations.

In addition to these, efficient development of such systems requires an extensive developer service stack as well as disciplined developer processes to ensure that services are developed, tested, configured and deployed in an efficient and sustainable manner. See the section on Operations Standards for more details.

2.5.4 Physical Architecture and Infrastructure Dependencies

The infrastructure dependencies of the current COLL are outlined below. In the SGA1 phase, the COLL will migrate from running on infrastructure hosted by the Blue Brain Project at EPFL (described in the figure below) to a combination of commercial cloud providers operating with explicit SLAs and infrastructure operated by SP7 partners and developed in part under the FENIX project. This mix of service providers will allow the COLL team to optimise development and deployment costs while ensuring that the COLL can operate continuously despite planned and unplanned outages at a given site.

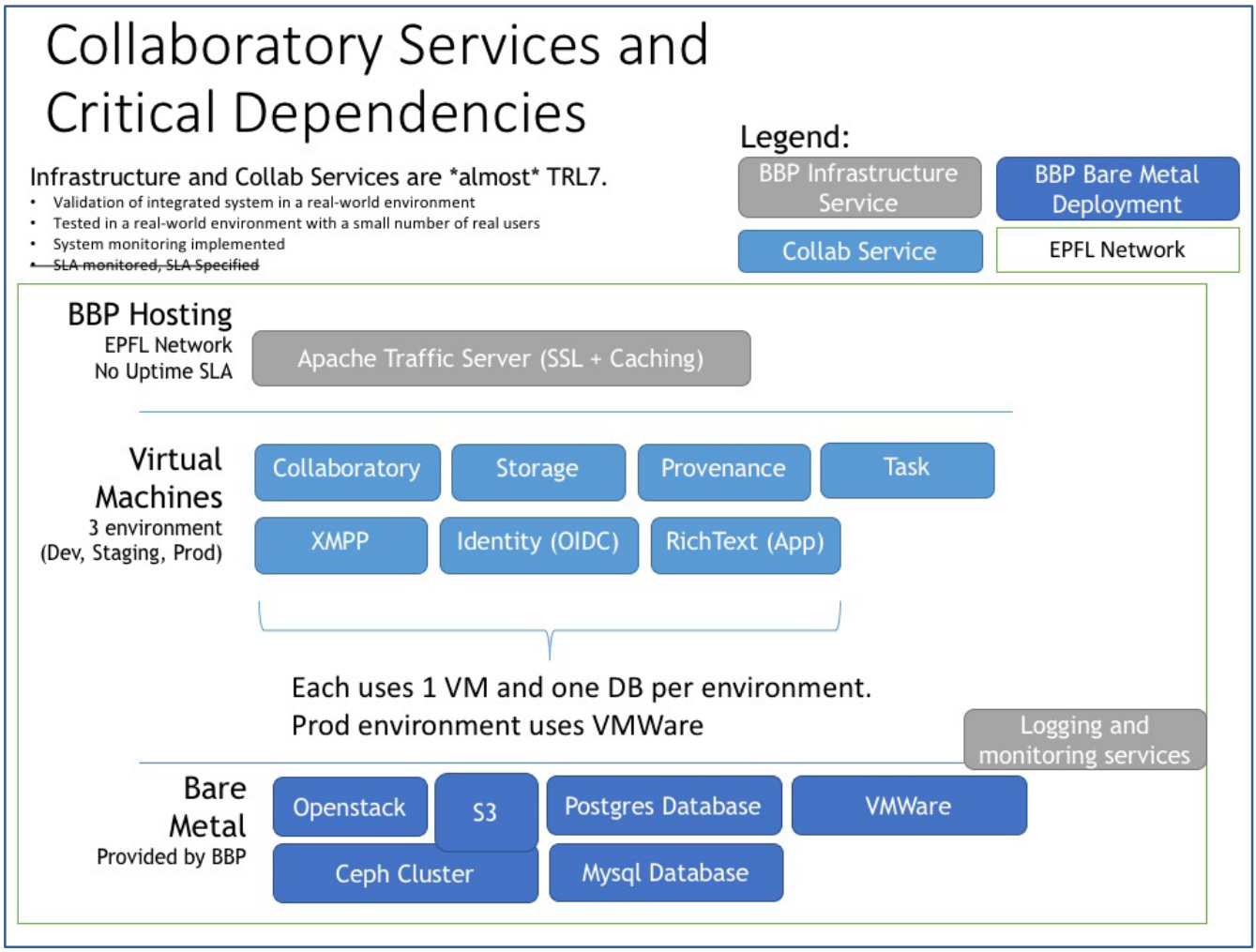


Figure 4: High-level Component Block Diagram

2.6 HBP-COLL: Dependencies

The service dependencies are well described in the architecture sections above, but this section discusses some of the other non-service, non-infrastructure dependencies which need to be addressed to ensure the COLL achieves its ambitious goals.

2.6.1 Required

1) Data management guidelines are needed to guide COLL (and other Platform) developers in the offering of data processing and data storage services.

2.6.2 Preferred

1) A number of COLL use cases would be well served by a pay-as-you-go computing access model. Currently such a model is only mature with commercial vendors, but the availability of such a model at HPAC partner sites would enable a number of interesting use cases.

3. Common Architecture

3.1 Considerations for Extensibility

3.1.1 Overview

3.2 For SaaS, PaaS and IaaS Software Development

In Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) tiers of the service hierarchy, software is a crucial component. All research activities in the HBP are heavy consumers and producers of complex software packages. A robust approach to software quality will be a critical factor in the success of the HBP.

Drawing on the experience of other large-scale software projects, the HBP identified the need to invest in software maturity early on.

These investments have continued and expanded to include the following best practices:

- Unit testing, typically with minimum test coverages (e.g.: 90% or higher on all COLL Team developments)
- Automated Integration testing suites (automated post-deployment system testing)
- Continuous integration
- Ticket management for project feature planning
- Scrum training for Agile software development (either with an Agile coach or an HBP practitioner)
- Coding standards
- Utilization of Code review by Developer teams.

Bringing all these pieces together results in a discipline called Continuous Delivery. All teams in the HBP use a variant of this approach with slightly different mixes of tools. These will be described in more detail in the upcoming D11.2.2 – HBP Software Engineering and Quality Assurance Approach delivered at SGA1 - Month 10.

3.3 Operations Standards

3.3.1 Overview

For SaaS, PaaS and IaaS tiers the reliable and repeatable deployment software is a crucial component of the overall operations strategy. Monitoring of services will be based on best practices from industry along with crucial expertise from HPAC providers in the HBP and Federated infrastructure providers throughout the EU.

The HPAC Platform team has well-defined deployment and monitoring practices which are informed by a long history of service provision. For the other Platform teams, no such history exists so best practices have been adopted from Agile software development. Platform teams in the HBP have been adapting these techniques and tools according to their team needs.

To this end a DevOps model is in the process of being pragmatically adopted across HBP:

- Deployment lifecycle w/ dev, staging (optional) and production environments for all services.

- Continuous integration – unit testing, integration testing, repeatable software builds and package releases.
- VM configuration development – source control and code review of configuration changes. Requires a programmable configuration system.
- VM configuration management – associate service configurations with specific VM resources.
- VM configuration deployment – deploy approved changes through an automated system.
- Object Storage – highly available, redundant storage for VMs and service data.
- Internet Gateway w/ Caching Proxy Server – services typically are not available directly via the internet. Best practice places a tuned caching proxy server between application servers and the open internet for reliability, flexibility and performance reasons.

3.3.2 Current Standard - BBP Standard DevOps Stack

The most widely used of the DevOps infrastructures, BBP Standard was developed by various teams at the BBP and serves as the basis for development, deployment, configuration and monitoring of the following Platforms:

- Collaboratory
- SP5 – Neuroinformatics Platform
- SP6 – Brain Simulation Platform
- SP10 – Neurorobotics Platform

Table 1: BBP Standard DevOps stack components

Service Category	Service	Provided by	Notes
Dev hosts	Openstack	BBP Core Services	
Staging hosts	Openstack	BBP Core Services	Optional environment
Production hosts	VMWare	BBP Core Services	
Continuous integration	Jenkins	BBP Core Services	Jenkins uses Openstack VMs for jobs
VM configuration development	Git + Gerrit	BBP Core Services	
VM configuration management	Foreman	BBP Core Services	Integrates with VMware and Openstack
VM configuration deployment	Puppet	BBP Core Services	
Object Storage	Ceph	BBP Core Services	Used by both Openstack VMs and COLL Storage Service
Internet Gateway w/ Caching Proxy server	Apache Traffic Server	BBP Core Services	
Monitoring	Icinga, Grafana, Kibana, syslogd, collectd	BBP Core Services	

3.3.2.1 BBP Standard Service deployment

The COLL and NIP currently follow the BBP Standard development and deployment model, as described in the HBP System Engineering documentation and represented in the figure below.

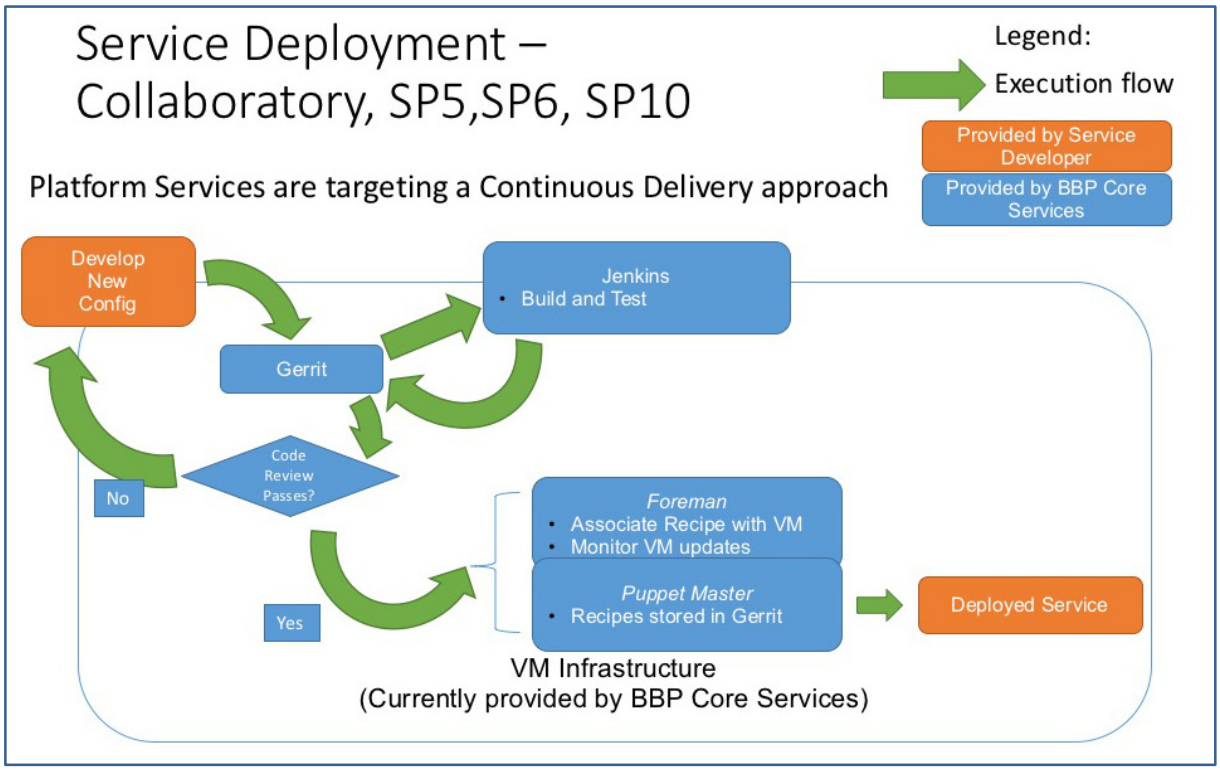


Figure 5: BBP Standard Continuous development and deployment

This process makes extensive use of the following open source tools:

Git – a leading distributed Version Control system.

Jenkins – a leading Java-based Continuous Integration (CI) system: https://jenkins.io/.

Gerrit – a leading Git and Java-based source code change review system: https://www.gerritcodereview.com/.

Puppet – a leading *devops* system configuration system: https://puppet.com/.

Foreman – associates Puppet recipes with particular hosts: http://theforeman.org/.

3.3.3 Future Standard - HBP Standard DevOps Stack

For Platforms moving towards serving HBP-wide needs, infrastructure must be soundly governed by an organisation with the mandate to provide services for and on behalf of HBP. The planned HBP Standard DevOps Stack serves as the basis for development, deployment, configuration and monitoring of the following Platforms:

- Collaboratory
- SP5 – Neuroinformatics Platform

Table 2: HBP Standard DevOps stack components

Service Category	Service	Provided by	Notes
Dev hosts	Cloud vendor – TBD	EPFL-HBPPCO procurement	
Staging hosts	Cloud vendor – TBD	EPFL-HBPPCO procurement	Optional environment
Production hosts	Cloud vendor and HPAC Platform	EPFL-HBPPCO procurement	
Continuous integration	Travis-CI	EPFL-HBPPCO procurement	
VM configuration development	Github and/or Phabricator	EPFL-HBPPCO	
VM configuration management	TBD	EPFL-HBPPCO	
VM configuration deployment	Puppet or Ansible - TBD	EPFL-HBPPCO	
Object Storage	Cloud vendor and FENIX	EPFL-HBPPCO	
Internet Gateway w/ Caching Proxy server	NGINX	EPFL-HBPPCO	
Monitoring	TBD	EPFL-HBPPCO	

3.3.3.1 HBP Standard Service deployment

The COLL and NIP will follow the HBP Standard development and deployment model, as described in the HBP System Engineering documentation and represented in Figure 5.

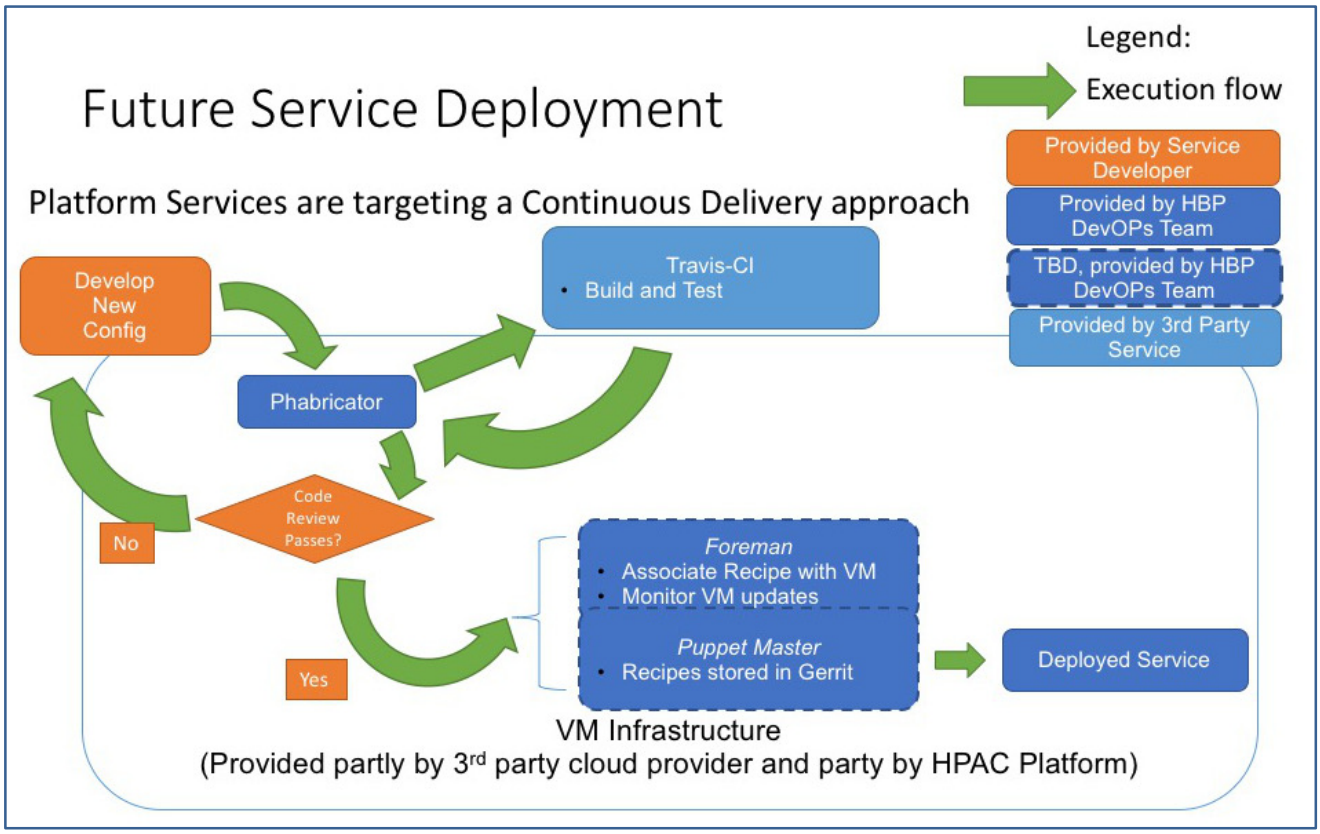


Figure 6: Planned HBP Standard Continuous development and deployment

This process makes extensive use of the following open source tools:

Git – a leading distributed Version Control system.

Travis-CI – a leading Java-based Continous Integration (CI) system: https://travis-ci.org/.

Phabricator – a leading code forge site with extensive project management feature set and code-review system change review system: https://www.gerritcodereview.com/.

Puppet – a leading *devops* system configuration system: https://puppet.com/.

Ansible – a serverless *devops* system automation framework: https://www.ansible.com/.

4. Extensibility and Platform Integration

4.1 Extensibility

As described in the HBP-COLL: Architecture section above, the HBP Collaboratory is designed from the ground-up to be extensible at all levels of the architecture. This has significant advantages. First, the HBP Collaboratory can be seen as self-extend to add each new feature addition. Secondly, it allows a tested extension model which can allow HBP Platform developers and third-party developers can use to add functionality to the Collaboratory. Finally, it requires HBP Collaboratory services and modules to be swapped out with alternative implementations readily as requirements change.

SP7 plays a key role in powering the Collaboratory and the HBP infrastructure architecture and should be considered a horizontal service provide both directly and indirectly through various HBP-COLL services. The details of this architecture are described in D5.6.2 “IT Architecture of the HBP Integrated System of Platforms”. Further details on the specifics of HBP-COLL extension can be found in the HBP-COLL developers guide, an attachment to this document.

For the remaining platforms, SP6, SP8, SP9 and SP10 have made heavy use of the Collaboratory to develop their solution verticals. The details of these interactions are described in the sections below.

4.2 Brain Simulation Platform

The Brain Simulation Platform (BSP) makes heavy use of Python-based Jupyter notebooks to document usage of their tools. These notebooks are offered through the HBP-COLL to allow users of the Platform to easily reproduce and customise model building, simulation and analysis task in an interactive fashion.

For less technically adept users, the BSP team is also building a collection of purpose-built web-based applications to increase the ease of use of certain workflows. These applications are largely build on the same foundations as the python notebooks above, but have been wrapped in web-based GUIs to ease understanding, configuration and execution of the workflows.

The two figures below describe the integration points between these two application paradigms and the HBP infrastructure. Case 1 describes the Jupyter notebook integration pattern and Case 2 describes the Collab Application integration pattern. It should be further noted that these two cases share similarities with the integration patterns of other HBP Platforms.

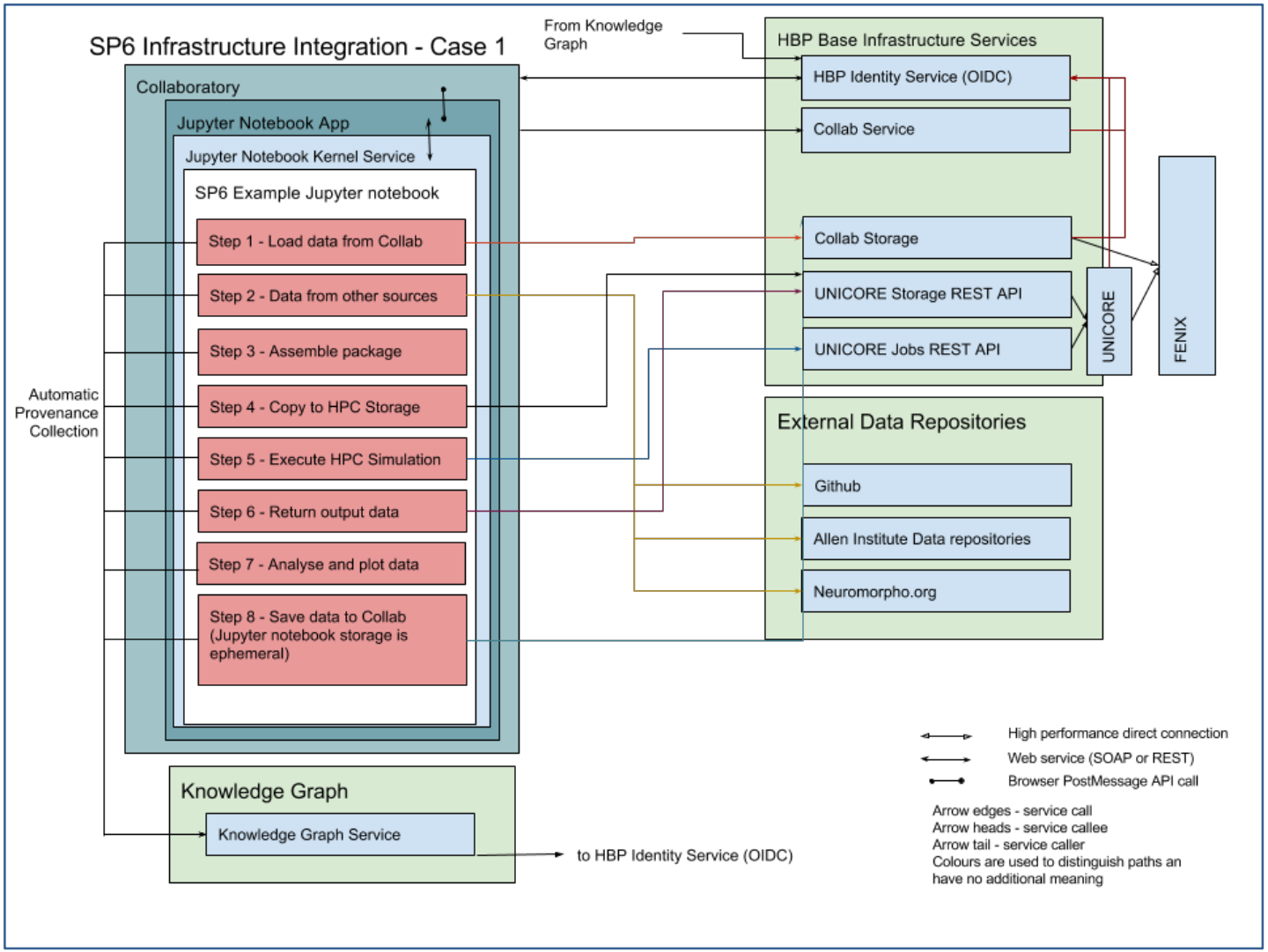


Figure 7: SP6 Infrastructure Integration - Case 1

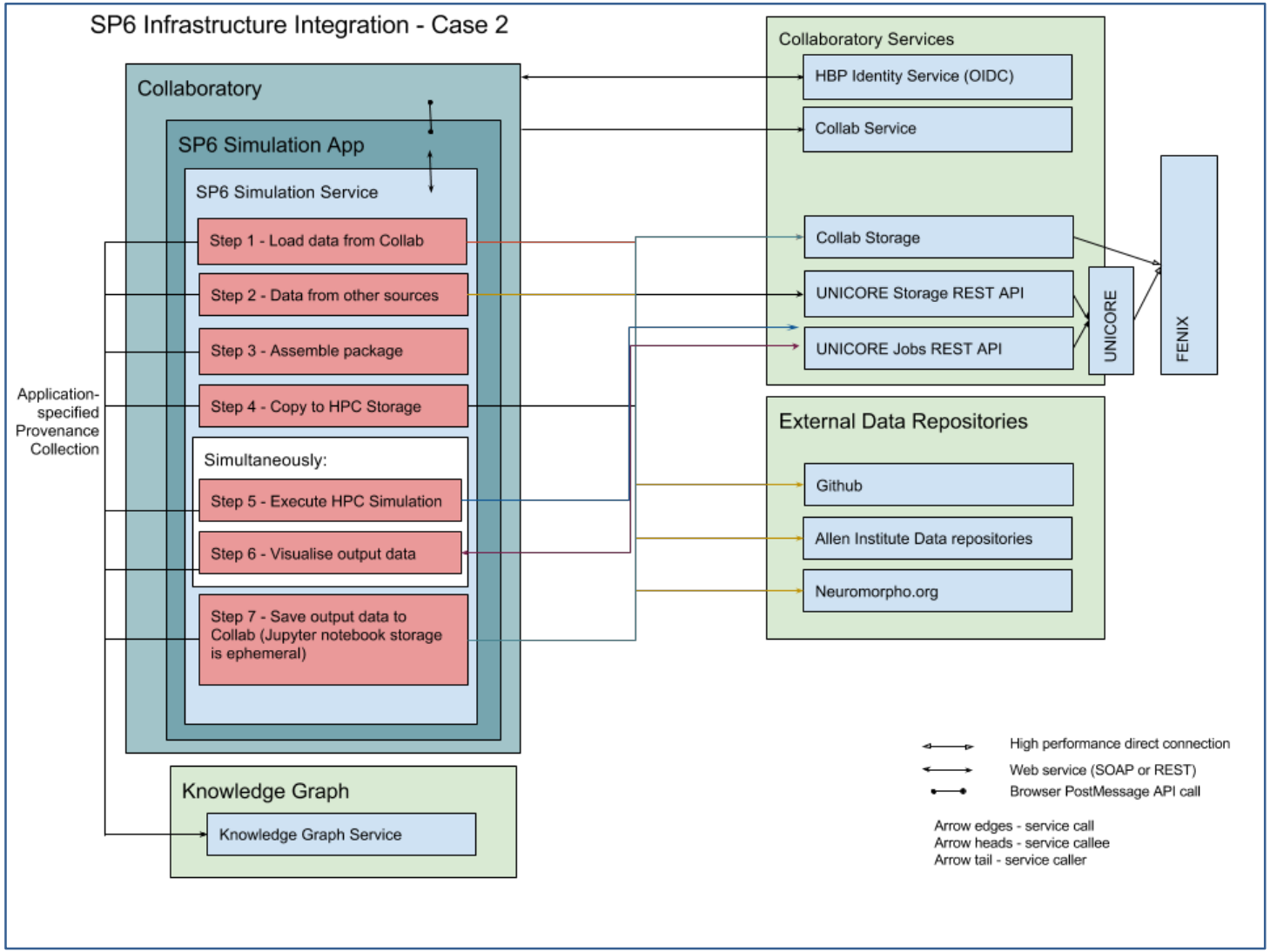


Figure 8: SP6 Infrastructure Integration - Case 2

4.3 Medical Informatics Platform

The Medical Informatics Platform (MIP) has different constraints on its authentication and authorisation practices due to privacy constraints which come when dealing with medical data. As a result, the MIP relies on external platform services much less than the other platforms. It is expected that Medical Informatics could integrate more services from other SPs as the constraints on human data are better reflected in HBP Data Policy and later implemented in various services.

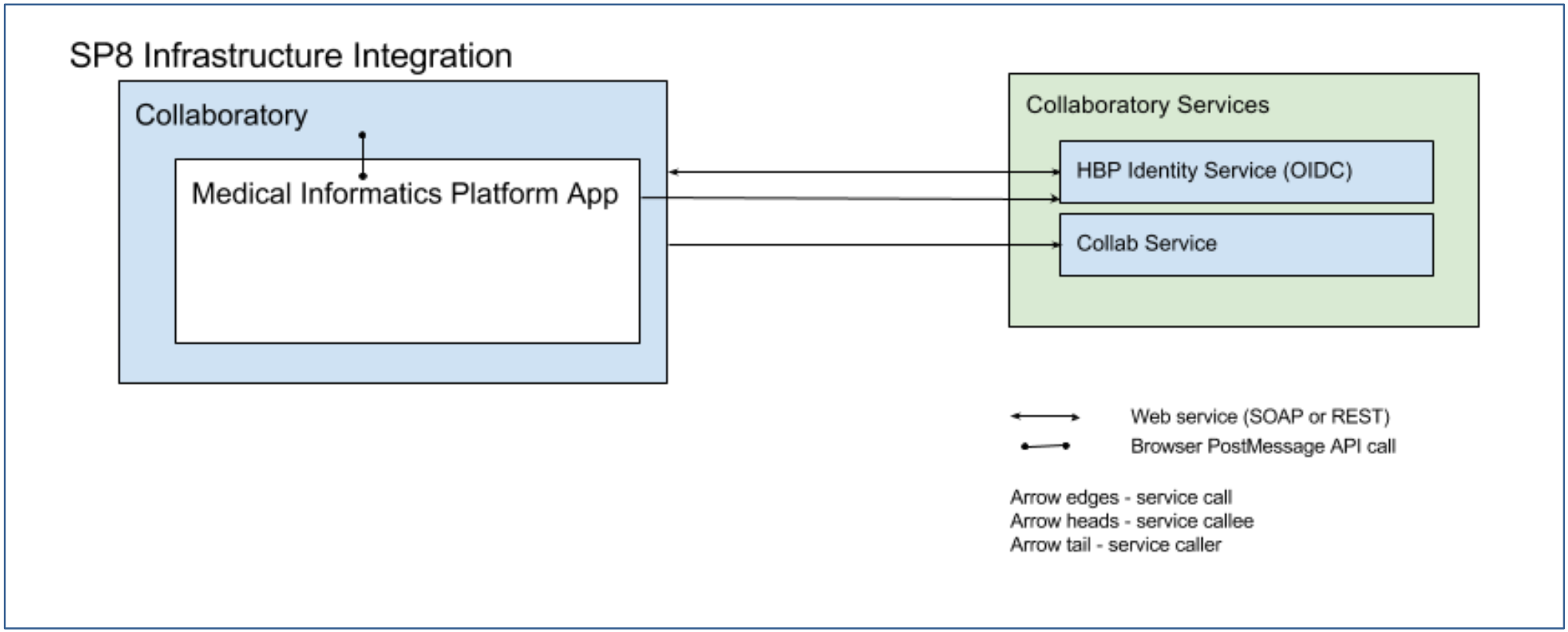


Figure 9: SP8 Infrastructure Integration

4.4 Neuromorphic Computing Platform

The Neuromorphic Computing Platform Collab App largely follows the model of the SP6 Case 2 Collaboratory App, but adds an additional dimension integrating the Neuromorphic job services. While not diagrammed as below, it is worth noting that the Neuromorphic execution services can also be used from the Collaboratory Jupyter notebooks, thanks to the OIDC token integration in the Jupyter notebooks.

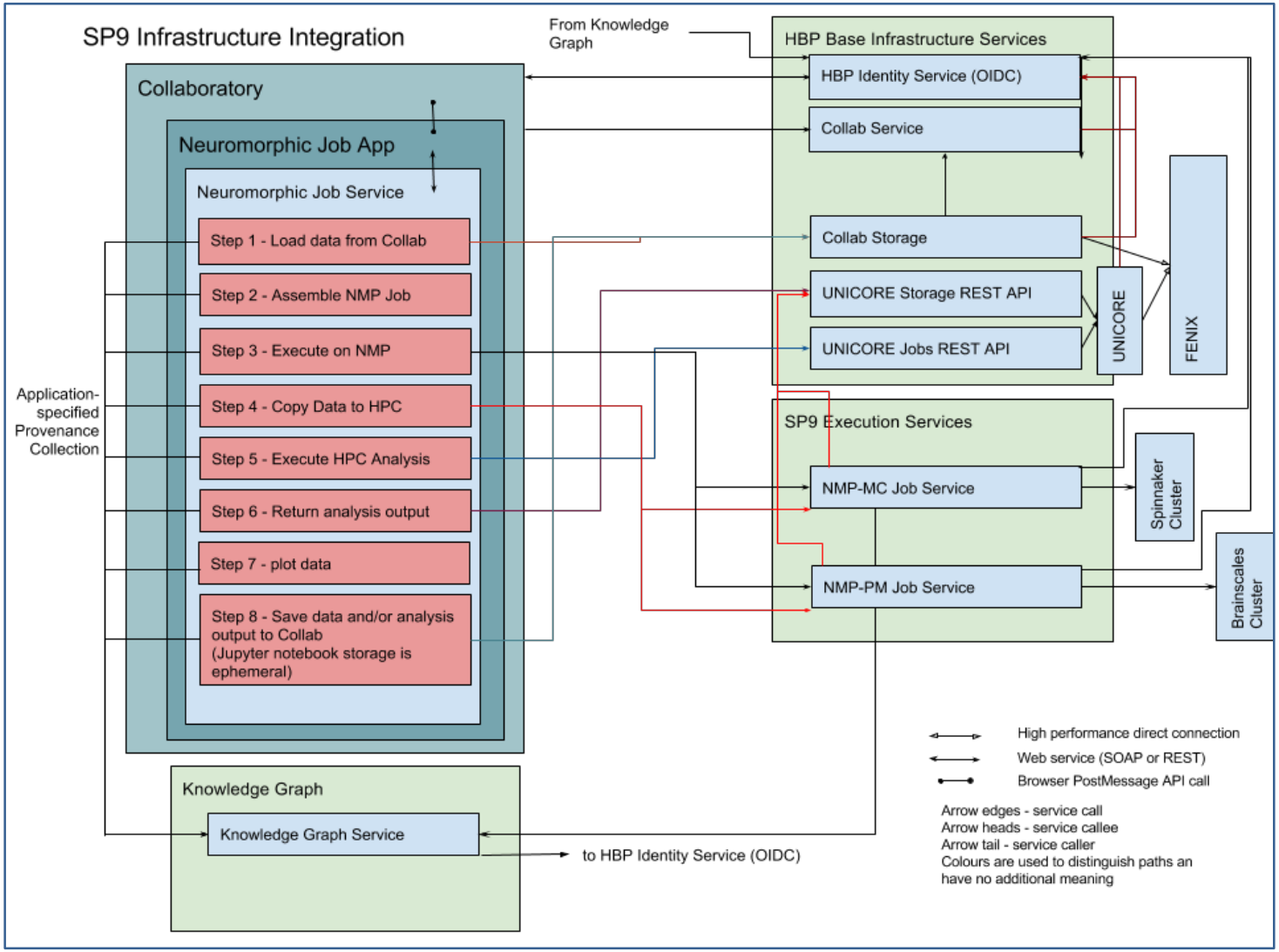


Figure 10: SP9 Infrastructure Integration

4.5 Neurorobotics Platform

The Neurorobotics Platform (NRP) also follows the SP6 Case 2 Collaboratory App integration model. This is expected to serve the HBP well in later versions of the NRP because the similarities in the integration patterns between the Neuromorphic Computing Platform (NCP) and the NRP will facilitate easy integration of NCP job execution services into the NRP simulation environment. This is a potential integration scenario which is not currently on the roadmap, but would be well supported and will only require having the NRP simulation backend collocated with the NCP for latency reasons.

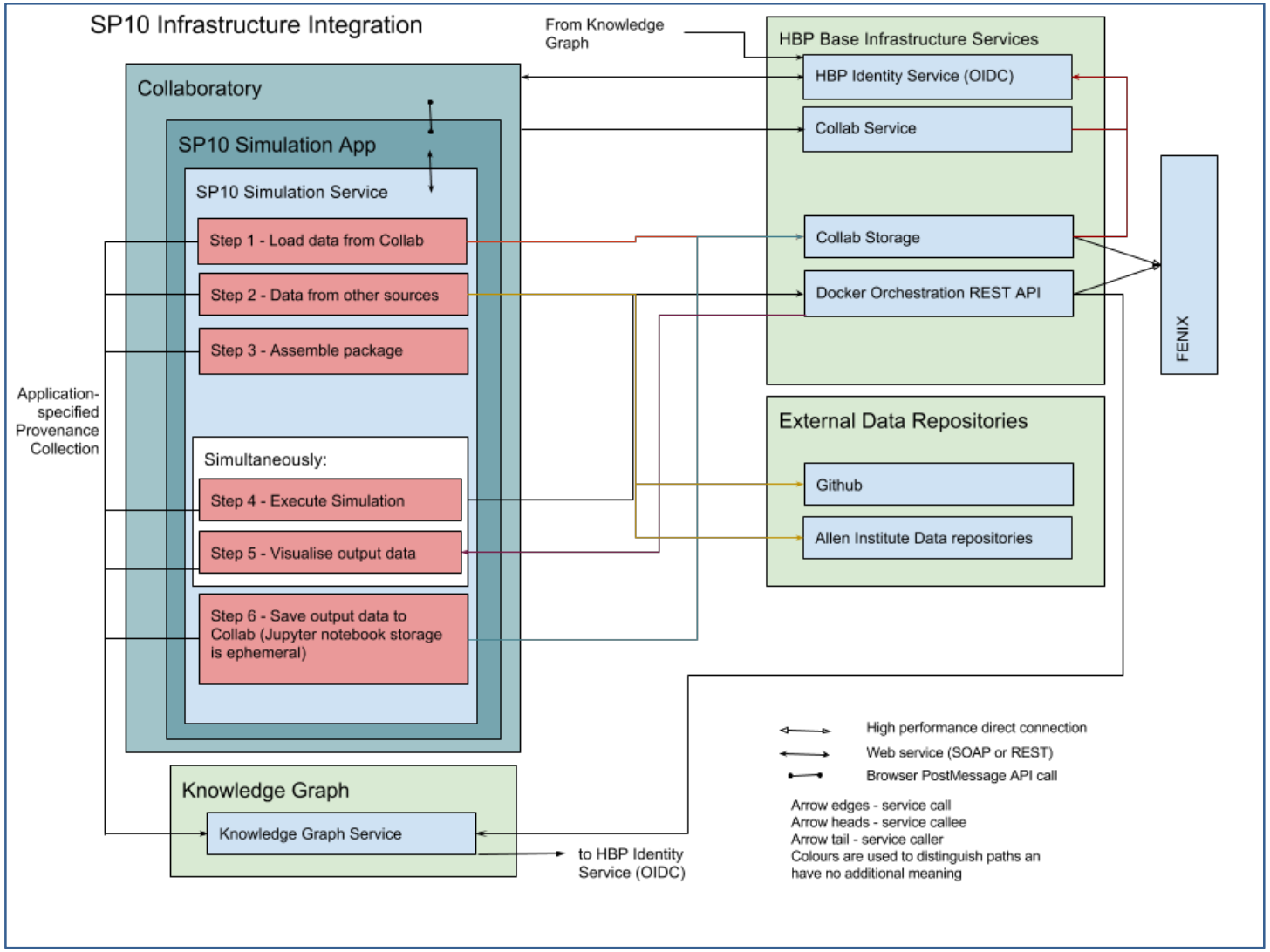


Figure 11: SP10 Infrastructure Integration

Annex A: Glossary

Term	Description
0-9	
2D Atlas	A 2D reference space, a collection of 2D parcellations, 2D images or a collection of registered data.
2D Parcellation	A collection of closed polygonal or spline boundaries at specific cut planes in 3D space. Each boundary is linked to one or more ontological elements.
2D Reference space	A collection of 2D subspaces, each aligned with a specific cut plane in 3D space. Each subspace has a single coordinate origin and affine transformation.
3D Atlas	A 3D reference space, a collection of 3D parcellations, 3D voxel volumes and a collection of registered data.
3D Parcellation	A collection of meshes that define 3D boundaries. Each boundary is linked to one or more ontological elements.
3D Reference space	A set of 3D basis vectors with a single coordinate origin and affine transformation.
A	
Anchor	A spatial location with orientation and scale or a semantic-spatial with optional orientation and scale.
Artefact	A high data-density discrete data element, primarily meant to denote a file larger than 10kB which is not human readable or editable.
Atlas	A 2D or 3D atlas.
B	
Biophysically realistic	Mathematical description of physical phenomena relevant to the biological processes of cellular behaviour. In particular, but not limited to, a model of the neuronal tree with the cable equation and ion channels by the phenomenological Hodgkin-Huxley formulation.
Brain System	A specific set of interacting brain regions.
C	
Capability supercomputing	Tightly integrated parallel supercomputer providing a high-speed and low-latency network between the computing nodes (in contrast to embarrassingly parallel computers).
Cockpit	Desktop, display wall or cave visualisation resource with a mechanism for good data locality.
Cognitive Architecture	A specific set of brain regions and interactions that are proposed to underlie specific cognitive capabilities.
Compute resource	A computer or collection of computers where a job can be executed.
Configurability	The ability of the simulator to provide extensibility for additional mathematical formulations of novel physical phenomena and integrate these mathematics in the compute-critical inner integration loop.

Term	Description
Continuous time spike interaction	Representation of the time point of an action potential as a floating-point number with double precision on a continuous time axis. The action potential is generated at the time point of threshold crossing of the sending neuron and is communicated with full precision to its target cells, where it affects the receiving neuron, acting at the point in time after the application synapse delay.
CRUD	A commonly used acronym for Create, Read, Update, Delete.
Curation	A manual or analytic process involving a human to make decisions about some property of the reconstruction or one of its components. Curation can be applied to everything from electrical channel models in the Hodgkin-Huxley model to ontology names for a particular neuron morphology class.
D	
Datatype	A datatype is a semantically enriched mimetype. For example, the mimetype of a particular data file might be XML, but the datatype would be CircuitML, implying that the data file can be interpreted in a richer way. This allows the selection of editing interfaces and input data in a much more user-friendly way than in the COLL.
Detailed model	The finest level of representation with full geometry.
E	
e-type	A short-hand form of the ne-type abbreviation defined below.
Entity	A COLL Project, a file or a folder. These exist as part of a hierarchy; they may have a parent and children. Each entity has a series of predefined key values associated to it (such as name, creation date, etc.) and can also have some custom Metadata associated to it.
Exact integration	A method applicable to the integration of sets of linear differential equations. The solution agrees to the mathematically exact solution. Often formulated in terms of a matrix exponential.
F	
File	Entities that are required to have a parent but cannot have any children. In addition to the standard attribute they also have a Content URL that defines how to access the content of the file.
Folder	Plain Entities that are required to have a parent.
Full scale	Representation of a network with the natural number of neurons and synapses per neuron as found in the biological system.
G	
Glial cell	Non-neuronal cells that function in homeostasis and energy usage, which provide support and protection for neurons. They can be divided into microglia and macroglia types.
H	
HBP Collaboratory (HBP-COLL)	The unified web interface through which the web-accessible components of the six HBP Platforms and all other HBP activities are made available.
MINDS	A minimum metadata specification. Similar in spirit to the Carmen MINI specification but tailored to the Use Cases of the HBP.

Term	Description
Hidden Entity	Entities can be hidden from the Document Service REST API and through the Web GUI. Hidden data are not visible by default in the COLL Project browser. Hidden data will still be optionally visible (though marked as hidden) to anyone it has been shared with. Hidden data are a separate function from true deletion. Deletion of data is a highly privileged operation that must be done by System Administrators on User request. See the Data Hiding Use Case for more information.
Host	A single operating system instance, running on virtualised or real hardware.
I	
Ionic conductance models	Models that represent ionic permeation through the plasma membrane. Both stochastic and deterministic approaches should be covered. Extension toward molecular models (WP6.4.1) is envisaged.
J	
Job	An instance of an execution of a Task on a compute resource. For some Tasks, the compute resource will be selected by the User in the COLL on job launch. For other Tasks the execution will be decided by the Task.
L	
Level of resolution	The choice of abstraction applied to the representation of the network. The level of resolution of MolSim corresponds to single neurons or synapses. The level of resolution of NetSim corresponds to single neurons and synapses. The level of resolution of CellSim equates to electrical compartments coupled by conductances.
M	
m-type	A short-hand form of the nm-type abbreviation below.
me-type	A short-hand form of the nme-type abbreviation below.
Macrocircuit	The definition of the whole brain as a set of brain regions connected through long-range fibre tracts - the whole brain.
Mesocircuit	The definition of the smallest collection of midrange interacting microcircuits through their intra-areal or regional arbours - a brain area or region.
Metabolism	The set of chemical transformations within the cells of living organisms that maintain life.
Microcircuit	The definition of the smallest collection of short-range interacting neurons through their local arbours.
Microcircuit models	Models that represent an entire microcircuit, including 3D geometrical architecture, synaptic connectivity and neuronal and synaptic models.
Microservice architecture	A variant of the Service Oriented Architecture ICT architecture where software functionality is provided by a collection of network accessible services, each with a well defined responsibility and minimal cross service dependencies. It is well described in various internet fora: (https://en.wikipedia.org/wiki/Microservices)
Molecular level models	Models that are structurally accurate at the subcellular level (organelles, intracellular and extracellular spaces) and that contain molecules that ultimately follow cell biological rules of production, transport, localisation and degradation as well as the environment-dependent thermodynamics and

	kinetics of their interactions. Both stochastic as well as deterministic versions will be covered.

Term	Description
Molecular Simulations	Numerical simulations at the atomistic or coarse-grained level used for predicting structures of molecular complexes and the estimation of kinetic and thermodynamic parameters for molecular interactions. Molecular Simulations are based on atomistic structures of proteins available either from the Protein Data Bank or from homology modelling.
Morphology	The geometric definition of the shape of a neuron.
Multi-constraint fitting	The process whereby one data parameter or property constrains other data parameters or properties.
N	
n-type	A class of brain cells or a particular instance, depending on context.
ne-type	Abbreviation for an electrophysiological type of cell. This abbreviation is used to refer to a class of cells or a particular instance, depending on context.
ng-type	Abbreviation for a genetic type of cell. This abbreviation is used to refer to a class of cells or a particular instance, depending on context.
nm-type	Abbreviation for a morphology type of cell. This abbreviation is used to refer to a class of cells or a particular instance, depending on context.
nme-type	Abbreviation for a morpho-electrophysiological combination type of cell. This abbreviation is used to refer to a class of cells or a particular instance, depending on context.
np-type	Abbreviation for a protein type of cell. This abbreviation is used to refer to a class of cells or a particular instance, depending on context.
Neuro-glia vasculature (NGV)	The three principal components in neural tissue, which function as a unit to regulate blood flow and metabolism.
NEURON	Open source simulator NEURON (http://www.neuron.yale.edu) developed by Michael L. Hines.
Neuron model	Implementation of neuron dynamics defined as a set of differential equations. The implementation solves the dynamics within a finite time span given the incoming spike events are supplied. Incoming synapses can be modelled as currents or conductances.
Neuropil	Any area in the nervous system composed of mostly unmyelinated axons, dendrites and glial cell processes that form a synaptically dense region containing a relatively low number of cell bodies.
P	
p-type	An abbreviation for projection type, a to-be-determined classification scheme for determining classes of projections between meso-scale brain regions.
Parameter	A low data-density discrete data element that is primarily meant to denote a value that one might enter into a single form element. It might also be used to refer to a richer configuration document containing a group of settings.
Parcellation	One or more spatial boundaries associated with a set of discrete semantic concepts. Usually developed by manual, semi-automated or automated image analysis of landmarks.

Platform	Software components: libraries, services, APIs and their documentation that are to be used to build portals or cockpits.
Predictive reconstruction	The process whereby multi-constraint solutions yield a hypothesis and hence a prediction of the data parameter space.

Term	Description
COLL Project	COLL Projects are Entities with no parent. In addition to the standard attribute and Metadata associated with Entities, COLL Projects have also ACL that define which Users can access their content.
R	
Reconstruction data	Data that is used to parameterise a model.
Reconstruction process	A workflow that uses a configuration of the data parameters and implements a set of fundamental biological principles to constrain and instantiate the model.
Reference space	In 2D, a collection of slices with an optional 2D *parcellation*. In 3D, a voxel volume with an optional 3D *parcellation*.
Registered data	A URL accessible data set with an *anchor*.
Resources	Parameters, Artefacts, services, or compute capacity.
REST	An acronym for Representational State Transfer, for a definition see http://en.wikipedia.org/wiki/Representational_state_transfer.
S	
s-type	Abbreviation for the type of synaptic connection. This abbreviation is used to refer to a class of cells or a particular instance depending on context.
Sa-type	Abbreviation for the anatomical type of synaptic connection. This abbreviation is used to refer to a class of synaptic connection or a particular instance depending on context.
Sp-type	Abbreviation for the physiological type of synaptic connection. This abbreviation is used to refer to a class of synaptic connection or a particular instance depending on context.
SAN	An acronym for Storage Area Network, http://en.wikipedia.org/wiki/Storage_area_network.
Semantic-spatial location	Association of semantic concept (e.g.: cerebellum) with a spatial boundary.
Service	A software function performed by a third party for a User or other Service. In the language of the COLL, Services consume Parameters, Artefacts and compute capacity. Services produce Artefacts and parameters.
Single neuron models	Models that represent entire neurons, including 3D structure, electroresponsiveness, synaptic activation and intracellular biochemical cascades (developed in WP6.4.1).
Site	A collection of hosts collected together in a single location. The grouping is potentially arbitrary. QIJ might be considered one site, LNMC another or one might consider EPFL a site unto itself.
Spatial location	2D or 3D location.

Synapse model	A model representing synaptic plasticity, such as spike timing dependent plasticity (STDP). The implementation solves the dynamic equation describing the evolution of the synaptic amplitude, typically formulated as a differential equation, given the spike times of the presynaptic and possibly the postsynaptic neuron.
Synaptic models	Models that represent processes of synaptic transmission, including neurotransmitter release and postsynaptic receptor activation. Both stochastic and deterministic approaches should be covered. Extension toward molecular models and molecular networks (WP6.4.1) of neuromodulation, synaptic plasticity and homeostasis is envisaged.

Term	Description
Systems Biology Markup Language (SBML)	A mark-up language for representing standardised reaction networks within compartments.
T	
Task	A logical software unit. A Task takes Artefacts and Parameters as input, and produces Artefacts and Parameters as output. It may or may not be visible as a Service. A Task identifies its dependencies and its default parameters. Concretely, it is a software component that combines: • A Python-based Task entry point • A git repository or Python package index URL for the Task • A repository revision or package content specified by sha1 • A requirements file specifying all required dependencies. Tasks can have dependencies in non-Python languages, but these dependencies must be packaged for reproducible deployment.
Task definition	The collection of data that defines an individual Task.
Task repository	A database of Task definitions.
V	
Validation data	Data that is used to validate a model.
Validation process	A workflow that compares results obtained in the model when experimental protocols used to obtain the validation data are applied to the model.
Vasoconstriction	Narrowing of blood vessels resulting from constricting of smooth muscle cells within the vessel walls.
Vasodilation	Widening of blood vessels due to relaxation of smooth muscle cells within the vessel walls.
Voxel	A 3D unit volume, the 3D analogue of an image pixel.
Voxel volume	A 3D volume made up of voxels. Typically, the voxels densely fill a rectangular prism spatial bounding volume.
W	
Workflow	A tree of decision structures and Tasks. A Workflow takes Artefacts and Parameters as input, and produces Artefacts and Parameters as output. It may or may not be visible as a Service.

Annex B: HBP Collaboratory Documentation (Release 1.9.3)

Human Brain Project

Co-funded by the European Union

HBP Collaboratory Documentation

Release 1.9.3

TBD

May 31, 2017

CONTENTS

Todo

This should be a very visual description with pointers to the different manuals and the introduction page.

Contents:

CHAPTER

ONE

INTRODUCTION

CHAPTER

TWO

USER MANUAL

The *HBP Collaboratory* User manual. Find out about the Collaboratory, in concepts and in practice.

Introduction

You will learn how the Collaboratory can enable your collaborations. You will find out about how to accomplish some common tasks in the collaboratory. You will find out where to get more help if you need it.

Intended Audience

This manual is intended for all users of the Collaboratory.

Document Status

This document is intended to accompany the Collaboratory with every release and be updated as the functionality of the Collaboratory is updated.

What is the Collaboratory?

“A [matchmaking] site for [scientific collaboration]” – Dr. Christine Aicardi, University College London

The Collaboratory is intended to be a a hub for scientific collaboration. More than this, the sharing and collaboration around data, software and services. Much of the functionality which is delivered through the Collaboratory is largely web based.

As part of the sharing strategy, the Collaboratory is architected to provide an ecosystem which can extended by people outside the Collaboratory developer team. If you are a software developer or you just like to program, you can extend the Collaboratory to better support how you want to work and collaborate with others.

User Interface

Concepts

Collaboratory - the central site where all HBP Platform functionality can be found.

collab - A collaborative project instance in the Collaboratory. It has a user editable navigation structure, one or more team members and a history of activities.

extension - software or services which extend Collaboratory or collab functionality. In most cases, users of the Collaboratory can contribute their own extensions.

apps - a user interface extension which displays itself in the workspace.

app links - binding an app to a point in the navigation tree.

Basic Elements

Navigation - organize the app links in your collab

Workspace - where apps are displayed

Collaboration - contact the people in the current collab team.

Showcase You can see the Collaboratory highlights in this collab [here](https://collab.humanbrainproject.eu/#/collab/143/nav/759)

Access Control

Access control for the Collaboratory is primarily enforced on the level of a *collab*.

A collab can be either *Public* or *Private*. All *collabs* have a list of members.

NOTE: HBP Collaboratory Apps should try to be as consistent with the Access Control models listed below. However, individual apps provided by others can allow more fine grained or course grained Access Control over their content inside the Collab Workspace.

Public collabs

- *Everyone* can create a public collab.
- *Everyone* who can login to the Collaboratory can view the collab.
- *Collab members* can edit the collab navigation items.
- *Collab members* can add files to the Collab storage.

Private collabs

- Only *HBP Members* can create a private collab.
- *Collab members* can view the collab.
- *Collab members* can edit the collab.

- *Collab members* can edit the collab navigation items.
- *Collab members* can add files to the Collab storage.
- *Collab members* can add/remove users from the Collab.
- *Everyone* else can not see the collab or content in the Collab storage.

At some point in the future the Collaboratory may support the following additional Access control options.

NOTE: The HBP Collaboratory team prioritizes features based on user feedback. If the features below are needed for your work please let us know in the HBP forum (https://forum.humanbrainproject.eu/), as it will help us prioritize future work.

Under Consideration for Future expansion

1. *Read-only* members of *Private collabs*. Such members would be able to read content in Private collabs, but not modify it or add/update/delete files in Collab storage.
2. Create a *Manage* role in both *Private and Public collabs*. Such members would be the only ones able to modify the *collab* member list of a given *collab*.
3. *Private collabs* for non-HBP members.

Software Catalog

Uploading Software Information

Uploading the software information for a new version of the tool can be done using *curl* command line or any other app able to send HTTP POST requests.

User token

To retrieve one of your tokens, you can use this page:

https://services.humanbrainproject.eu/oidc/manage/user/tokens

Just copy an existing token from here, it will have the value $MY_VERY_PERSONAL_TOKEN in the following steps.

Open a terminal and enter the following to create a very minimal version of 'awesome-software' software.

```
$ curl -0 -X POST https://services-dev.humanbrainproject.eu/software-catalog/v0/api/
 ↪version/ \
  -H "Authorization: Bearer $MY_VERY_PERSONAL_TOKEN" \
  --data "name=awesome-software" \
  --data "version=1.0.0" \
  --data-urlencode "title=Awesome Software" \
  --data "category=library"
```

CHAPTER

THREE

APP DEVELOPER MANUAL

The *HBP Collaboratory* App development guide for beginners to masters.

Introduction

Intended Audience

This manual is primarily targeted at platform developers with a software engineering background.

To develop Collaboratory application, it is assumed that you have some knowledge on how to build web applications: - Basic knowledge of what is a Rest Web Service to use our API - At least average Web Development skills for developing the frontend;

this includes being fluent with Javascript HTML and CSS

- Understanding Oauth2 basics

We will propose various toolkit for our standard stack, but you are definitly free to choose other technologies. Our stack includes: - Python Django for the backend server - AngularJS for the frontend scripting - Sass for the stylesheets on the frontend - NGinx as server

The tutorial is based on basic knowledge of Django. The starting guide of Django is probably sufficient.

Document Status

This documentation explains how to develop application for the Collaboratory. It is currently in development so there might be some rough edges right now. In the future the Collaboratory will supper easier ways of developing scientific applications, targeted at scientific developers.

The core of the documentation is the tutorial section. The API references are currently incomplete and will be improved over time.

Applications

An *Application* defines a set of HTML5 pages that enable participation in one HBP activity. They are context sensitive in the sense that they show data in the context of where they are instantiated, for example:

- A File Viewer application should open with the intended file already displayed. It is the same as opening a text document by clicking on the file directly as opposed to first opening the text editor application.
- A Wiki application should store and retrieve its content given the context in which it has been opened.

Applications are developed by the HBP platforms and HBP partners to enable wide access to their core business tools.

Backend

To achieve their goal, an *Application Developer* can use a wide range of RESTful web services. If HBP Collaboratory ensures the integration of scientific data and tools within the same working space, the interoperability of the various platforms is guaranteed by the underlying *Microservice Architecture*, which provides a directory of web services that are guaranteed to seemlessly work together.

An application will create its UI by using a combination of general web services (authentication, provenance, unicore, data registration, …) and specialized and/or custom web services (brain simulation builder, brain atlas, …).

Applications that are a good fit for Collaboratory

Any scientific application that enables access to data in order to visualize, query, or more generally interact with them could be present in the Collaboratory.

A scientific team should be able to organize its project material and tools within the HBP by using only the Collaboratory.

An application that cannot be accessed using a web interface without a lot of effort should still have at least a *Project Page* describing how to get to and work with the tool.

Any HBP management application is also a good fit as we want to assemble the tools that help a team to organise their activities within the Collaboratory.

Application Requirements

An application must provide at least one HTTPS URL that will be used to display its purpose. The application must be registered within the Collaboratory *Apps Manager* by its developer. An application is responsible for its hosting, data storage, security and access. The HBP Consortium is here to help find solutions of course.

Developing Applications

As of today, developing a Collaboratory application requires web development skills. Any application is composed of a web UI that will be loaded within the Collaboratory platform following the same kind of patterns as a Facebook iframe application.

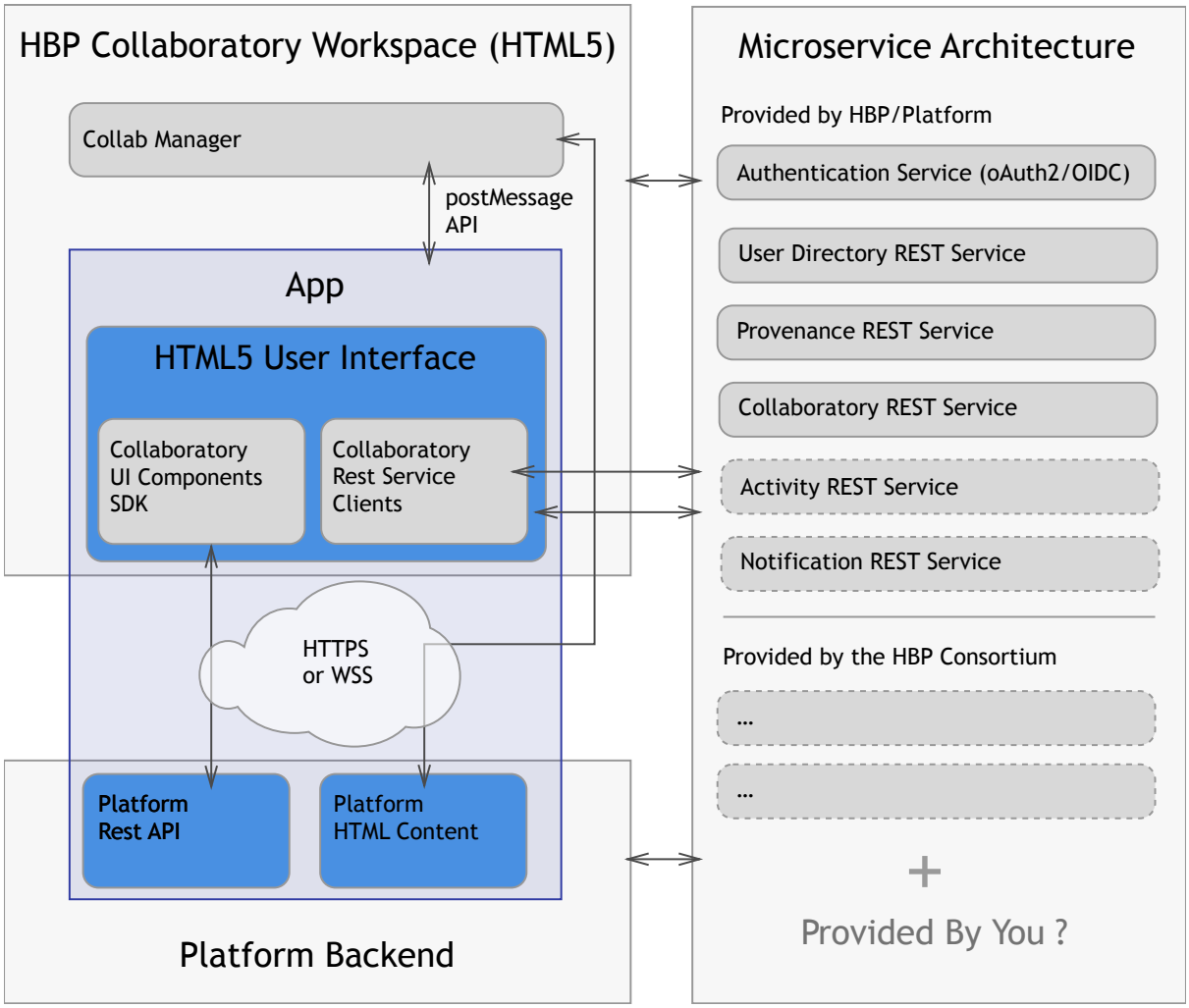
In the upcoming release, we will release alternative ways of creating applications to enable a broader community to create them. One of the main targets is to enable scientific developer to publish application that let scientists run code in the HBP computing centers. In those cases, the

web User Interface (UI) generation will be automated. A web developer will be able to step in later to replace the automatically generated UI with a more user-friendly custom UI.

Streamline Publication

One of the goals of the Collaboratory is to streamline the process of research and publication of new discoveries. Preparing an accompanying website for a publication, such as in the NMC Portal should become a matter of cleaning up a *Collab* instead of becoming a few months development project. As a result, any data that should be part of a publication should at least have a viewer application available in the Collaboratory.

Architecture Overview



An Application is composed from an HTML5 frontend and a backend coded with the technology of your choice (our current default is Python). The backend is typically in charge of serving the frontend and storing your Application data.

To develop an application quickly, we provide the Collaboratory SDK. It is composed of:

- angular-hbp-common (AngularCommon): delivers a set of common libraries for AngularJS. You can use them in an AngularJS based web application to connect to the Collaboratory core services.

- collaboratory-ui (: provides a set of AngularJS directives and filters of reusable components.
- collaboratory-theme (Collaboratory UI Theme) : provides a `CSS` or `SASS` template based on Twitter Bootstrap
- generator-collaboratory-app: a Yeoman generator to bootstrap a client application for Collaboratory

---

Todo

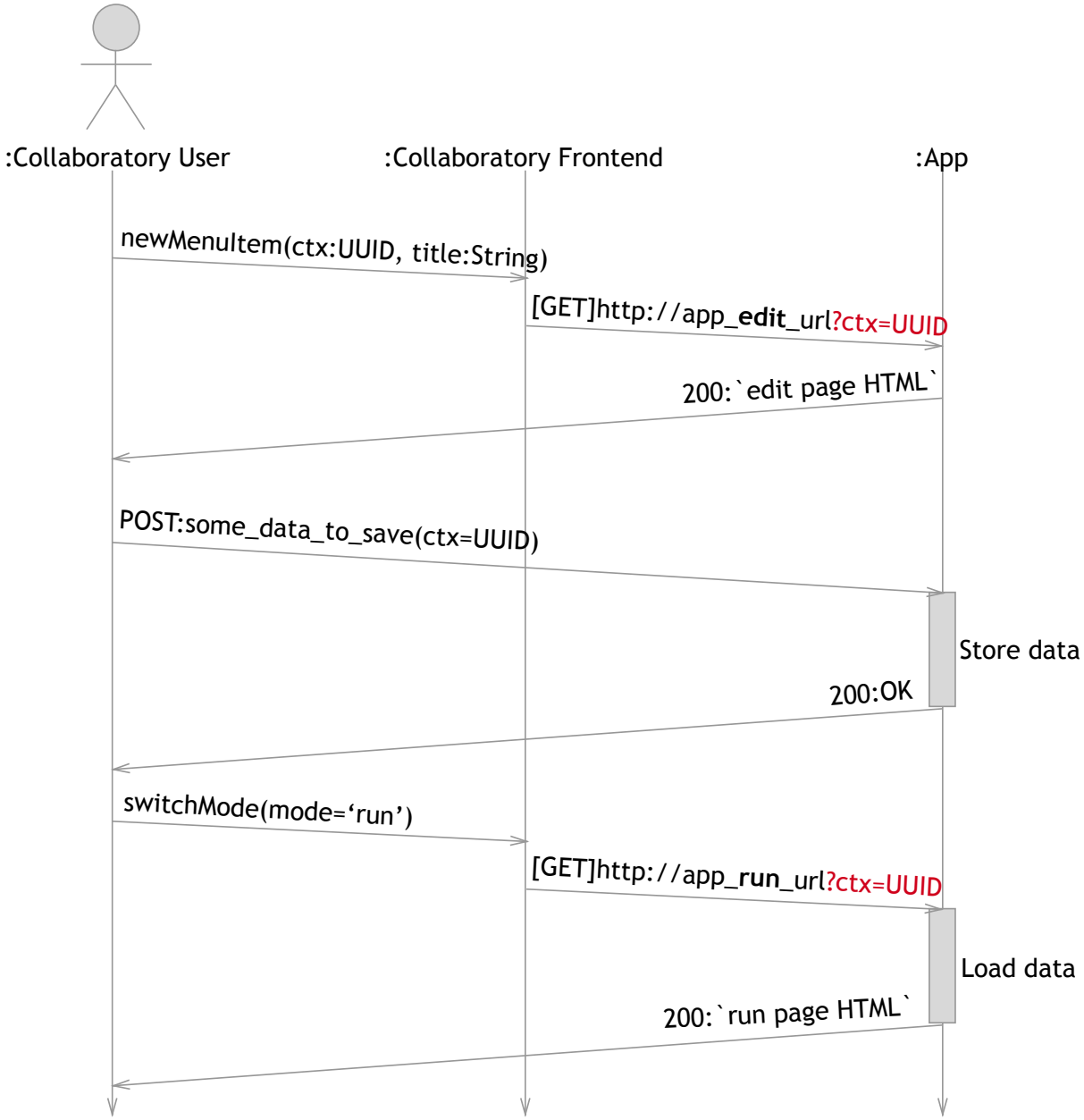
Point to specific documentation above

Basic user interaction with an App within Collaboratory

Your application can provide two access points to load the client:

- a `run` mode URL, accessible to all people who can access the Collab
- an `edit` mode URL, accessible only to Collab members

Note: Your application can further restrict the access to some data or actions but they should provide the user with a message of what is causing the restriction.



UML Sequence Diagram representing the interaction between a user and an app within the HBP Collaboratory.

1. The user adds a *Navigation Item* in the navigation. The *Navigation Item* is assigned a *Context UUID* and is bound to an *Application*.
2. When the item has been created, the *Edit View* is instantiated in the workspace iframe. To achieve that, the HBP Collaboratory opens the *Edit URL* defined by the application and appends `ctx=contextUUID` as a query parameter.
3. The application renders the *Edit View*, which should store user configuration data associated with the *Context UUID*.
4. At some point, the user is done with the editing and switches to the *Run View*.
5. The Collaboratory now opens the *Run URL* defined by the application and appends the same *Context UUID* to it.
6. The application uses the *Context UUID* to retrieve the stored data and display a view that is

specific to this *Navigation Item*.

General Concepts

Below we present the main concepts that describe an application.

Roles and ACL

All users need to be Authenticated to access the HBP Collaboratory. The authorization involves the creation of an HBP user account. The Collaboratory provides a few roles that an application can use to provide Authorization.

- *Collaboratory Administrator*: Global admin of the Collaboratory platform
- *Collab Member*: Someone who is member of the current collab team
- *Collab User*: Anybody with an access to the current collab

There is inheritance at play here: a *Collab Member* is a also *Collab User*. Likewise a *Collaboratory Administrator* is also a *Collab Member*

Edit/Run

As we have seen previously, an application should avoid a common view to privileged content in the context of the current *Collab* and *Context UUID*. For example, given a specific *Navigation Item*, an application *Run View* should display a specific document rather than a list of all available documents. To create this link between a *Navigation Item*, a *Context UUID* is sent to each view. The application should provide an Edit view that links this *Context UUID* to some relevant data.

To summarize:

- the *Edit View* provides a way to configure how the application is accessed within a particular *Context UUID*.
- the *Run View* provides a UI to work with the data defined by the current *Context UUID*.

Note: In most applications, the *Run View* will also have editing functionality, but the purpose is different. in a *Run View*, We are working with the application data, not the application configuration. For example, in a Brain Simulation application, the *Edit View* will let you choose a specific model while the *Run View* will let you configure the simulation parameters, run it, and display the results.

The access rights are not the same between both views. A *Collab Member* can access both the *Edit View* and the *Run View* where a *Collab User* can access only the *Run View*.

Note: This restriction is enforced only in the UI. The application should enforce the same access permission in its backend.

The run view can (and should) further restrict what is possible for a given user based on Collaboratory roles or application specific ACL. In this case, the application should always state why something cannot be done and explain how to enable the functionality.

Context UUID

This is an important concept to understand. HBP Collaboratory stores no application data.

Note: Why? Data is a very important citizen of the system. They need to be backed up, migrated, verified, accessed from various location. They are very heterogenous by nature. As an *Application Developer* it is your responsibilityto handle them with care. The HBP still plans to provide some common storage infrastructure and service to access data, but it is not part of the HBP Collaboratory mission to solve this concern.

Instead, the Collaboratory provides the application with a *Context UUID*, which respects the UUID v4 format specification. The application uses this context to match the view of a *Navigation Item* to its data. Of course, the application doesn't know at first how to map the context to a set of data, this is the purpose of the *Edit View*. At this point, a context UUID matches exactly one *Navigation Item*. This principle might be relaxed a bit in the future but at least the *Context UUID* should always point to a given data set. And it will be possible to resolve *Collaboratory Access Control* based on a *Context UUID*.

Note: It is possible to create a new *Navigation Item* from within an application. In this case, the application directly knows what data is bound to the *Context UUID*. In this case, the *Edit View* is not needed.

Getting Started

A Simple Example

The following example is here to show you what is a very simple application and what makes it a Collaboratory application. Consider it as the *Hello World* for a Collaboratory Application.

The example can be downloaded as an archive: `collaboratory-app-example.tar.gz`

The remaining part of this article explains what is in the application and how to load it within the Collaboratory.

Project Structure

File	Description
index.html	The main HTML page
app.js	This application javascript code
lib/hello.all.js lib/hbp.hello.js	The library HelloJS is used to manage authentication that handles user authorization. This library handles all the OpenID Connect authentication.
lib/jquery.js	jQuery is a popular javascript framework
lib/collaboratory.bootstrap.css	The Collaboratory CSS theme. This theme is based on Twitter Bootstrap. Using it ensures a consistent look and feel with the Collaboratory.
fonts/	Contains fonts required by the theme stylesheet
runserver	A Python script to run a local ssl server

The index.html file

```
<!DOCTYPE html>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<link rel="stylesheet" href="./lib/collaboratory.bootstrap.css">
<title>Hello World</title>

<header>
  <h1>My First Collaboratory Application</h1>
  <h2>Retrieve this collab informations</h2>
  <p><button class="btn btn-primary" href="#" retrieve-collab-info>Retrieve Collab␣
 ↪Informations</button></p>
</header>

<article>
  <h3 class="collab-title"></h3>
  <p class="collab-content"></p>
  <pre class="data-source" style="display: none;"></pre>
</article>

<script src="./lib/jquery-2.1.4.min.js" charset="utf-8"></script>
<script src="./lib/hello.all.js" charset="utf-8"></script>
<script src="./lib/hbp.hello.js" charset="utf-8"></script>
<script src="./app.js" charset="utf-8"></script>
```

A simple HTML5 file that contains a button to request collab informations and simple markup to hold the server response.

The app.js file

```
/*global $, document, window, hello*/
(function() {
  'use strict';

  var init = function() {
    // Setup OpenID connect authentication using the clientId provided
    // in the HBP OIDC client page.
    // https://collab.humanbrainproject.eu/#/collab/54/nav/1051
    hello.init({
      hbp: '2bc1364d-1039-495b-b51e-608108cbefce'
    });
    // If the user is not authenticated, it will redirect to the HBP auth server
    // and use an OpenID connect implicit flow to retrieve an user access token.
    hello.login('hbp', {display: 'page', force: false});

    $(document).ready(function() {
      $('*[retrieve-collab-info]')
      .on('click', retrieveCurrentContext);
    });
  }

  // Extract the context UUID from the querystring.
  var extractCtx = function() {
    return window.location.search.substr(
      window.location.search.indexOf('ctx=') + 4,
```

```
      36 // UUID is 36 chars long.
    );
  };

  var retrieveCurrentContext = function() {
    var ctx = extractCtx();

    // Retrieve the user auth informations
    var auth = hello.getAuthResponse('hbp');
    if (auth && auth.access_token) {
      var token = auth.access_token;
      // Query the collaboratory service to retrieve the current context
      // related collab and other associated informations.
      $.ajax('https://services.humanbrainproject.eu/collab/v0/collab/context/' + ctx + '/
↪', {
        headers: {
          Authorization: 'Bearer ' + token
        }
      })
      .done(function(data) {
        // Update the DOM with the context object retrieved by the web service.
        $('.collab-title').html(data.collab.title);
        $('.collab-content').html(data.collab.content);
        $('.data-source').html(JSON.stringify(data, null, 2)).show();
      })
      .fail(function(err) {
        $('.data-source').html(JSON.stringify(err, null, 2)).show();
      });
    } else {
      $('.collab-title').html('Not Authenticated');
      $('.collab-content').html('Please login first');
    }
  };

  init();
}());
```

This app.js file contains the javascript code that will authenticate the user and request the current collab context information.

The `lib/` folder

The `lib` folder contains the `hbp.hello.js` script, that has been downloaded from the hbp.hello.js git project. This library handle all the OpenID Connect authentication flow and is used by our main scripts.

Make this example work

To make this example work as a Collaboratory application, you need to:

1. create an OpenID Client
2. modify the code to use your OpenID Connect Client ID
3. make the application accessible through HTTPS

4. register the application in the Collaboratory
5. create a new Navigation Item using this application

Create an OpenID Connect client

The *How to develop apps* Collab has a navigation item called OpenID Connect Client Manager that lets you register new applications.

The user will have to authorize your application in order:

- for him to use it
- for the application to use other API on its behalf

In this example, we use the OpenID Connect user access token to get access to the current Collab information and display it.

Modify the code to use your OpenID Connect Client ID

in `app.js`, look for the string *YOUR_CLIENT_ID* and replace it by the the one provided in the previous step.

Access through HTTPS

Any way that will let you serve a static page over HTTPS is acceptable. Two ways are described in this example, using Divshot free service or a local Python server.

Using a local Python server

For testing purpose, you can run a local https server to test the application.

Note: Please Note that only you will be able to access your application so don't release anything public this way. Add a file called runserver in the root of the website and paste this content

---

```
#!/usr/bin/env bash
HTTPS_PORT=4443
HOST_SERVED=localhost
CWD=$(pwd)
CERT_PATH=$CWD/server.pem
APP_ROOT=.

# move to app root folder
cd $APP_ROOT

# generate self-signed cert the first time
if [ ! -f $CERT_PATH ]; then
    openssl req -new -x509 -subj "/C=CH/ST=Geneve/L=Geneve/O=lol/CN=localhost" -keyout
 ↪$CERT_PATH -out $CERT_PATH -days 365 -nodes
fi
```

```
echo "launching HTTPS server on port $HTTPS_PORT"
python -c "import BaseHTTPServer, SimpleHTTPServer, ssl;\
    httpd = BaseHTTPServer.HTTPServer(('${HOST_SERVED}', ${HTTPS_PORT}), SimpleHTTPServer.
↪SimpleHTTPRequestHandler);\
    httpd.socket = ssl.wrap_socket (httpd.socket, certfile='${CERT_PATH}', server_
↪side=True);httpd.serve_forever()"
```

You can then run the server using the following command.

```
$ . ./runserver
```

During the first launch, it will generate a local https certificate. Before going to the next step, ensure your computer trust your self signed certificate (see Trust your certificate) ; otherwise the webpage will not be display.

Your page will be available at the URL: https://localhost:4443/

Register the application in the Collaboratory

To make your application accessible in the Collaboratory, you need to be able to add it to a Collab navigation. To do so, it is necessary to register your application using the Apps Manager page in the How to develop apps Collab.

1. Click the **Register App** button
2. Enter your application **title**
3. Enter a **description** for your application
4. Enter the **main URL** for your application, which is the URL obtained in the previous step
5. Leave the **edit URL** blank
6. Leave the default entry label blank
7. Use *Prototype* as **category**
8. Ensure the **visibility** of the app is *private* because you don't want everybody to use your example application.
9. Hit the **save** button.

Create a new Navigation Item using this application

You can now navigate to one of your collab and create a new instance of your application. When you access your application instance for the first time, you should see the authorization page. Once approved, a page with a single button will be displayed. Once the user hits the button, the collab title, description and the raw result of the query should be displayed as well.

What's Next

You did the quick tour, now is the time to: - start developing your own application using your preferred web stack. You might be interested by our various web service documentation there. - Create Your First Application using Django as a backend - get in touch with the Collaboratory team for more informations

Install and Setup

Create a local SSL Certificate

To see how your application behaves within Collaboratory, you will have to run your local server using an SSL certificate. The following guide explains how to create a self signed SSL certificate.

Note: Security is an important issue. You should set up even your local server to work over an HTTPS connection for various reasons. The most important being that a token will be passed to your local server and HTTPS is an important security measure to keep it private. Moreover, any content loaded into the Collaboratory iframe will eventually be accessed through HTTPS.

The quickest way to achieve this on a development box is to use a self-signed certificate that you will trust on your own development computer. Since you will have to tell your OS to trust it, it is very important that you don't share this certificate or commit it to a any public repository.

Ensure you have OpenSSL installed on your computer:

```
$ which openssl
# /usr/bin/openssl
```

Install Open SSL

OS	Installer
Mac OSX	`brew install openssl` (using Homebrew)
RedHat	`yum install openssl`
Ubuntu	`apt-get install openssl`

Generate a private, self-signed, SSL certificate

Inside a private folder, for example `$HOME/.ssl`, issue the following commands.

Note: The documentation will use that path as a default so if you decide to use something else, you might need to adapt the examples.

Generate a self-signed SSL certificate:

This command generates a signing key, a certificate signing request and then a signed certificate, using its own key.

```
openssl genrsa -des3 -passout pass:x -out server.pass.key 2048
openssl rsa -passin pass:x -in server.pass.key -out server.key
rm server.pass.key
# Ensure to use ``localhost`` as the Common Name
openssl req -new -key server.key -out server.csr
openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt
```

Trust your certificate

In order for your browser to accept a self-signed certificate, you must tell your OS to trust it. This operation is OS dependent:

Todo

Complete command to trust the ca cert on RedHat and Ubuntu.

OS	Trust
Mac OSX (Yosemite)	issue the command `open server.crt`. The certificate will automatically be added to your login keychain.
RedHat	TBD
Ubuntu	TBD

On Mac OSX Yosemite, to trust the local certificate, open it in the Keychain Access app. You then have to restart your browsers to refresh their internal keystore.

There are no strong requirements on how to build an application since one can be defined just as a title grouped to two HTTPS URLs declared in the *Apps Manager*. However, to build a rich interactive application the HBP provides a Collaboratory SDK that developers can leverage to quickly get a working application. This SDK requires few tools to be present in your development environment:

- NodeJS: NodeJS is used to compile the HTML5 assets and retrieve other javascript development dependencies.
- GruntJS: A NodeJS task runner used as a build tool.
- Bower: A package manager for the web.
- AngularJS: Javascript MVW framework.
- Python: At least to run the quickstart guide mini-server.

Install Dependencies

NodeJS

- Install via package manager
- Download binaries or source

Depending on your environment you might need a symlink or an alias for `node` to point to `nodejs` executable.

Grunt and Bower

Install globally via the NPM (NodeJS Package Manager)

```
npm install -g grunt-cli bower
```

Python

OS	Installer
Mac OSX	`brew install python` (using Homebrew_)
RedHat	`yum install python`
Ubuntu	`apt-get install python`

Or Download binaries or source.

OpenSSL

OS	Installer
Mac OSX	`brew install openssl` (using Homebrew_)
RedHat	`yum install openssl`
Ubuntu	`apt-get install openssl`

Create Your First Application

Bootstrap the Django application

The first step is to create a Django application that you can start using SSL.

To create the Django project, issue the following command in your project parent folder:

```
$ django-admin startproject miniki
```

This create a standard Django layout in the *miniki* subfolder. It is a good practice to use Python virtualenv to manage dependencies and avoid conflict.

Virtual Environment

Create a virtual environment and activate it with the following commands:

```
$ virtualenv .venv
$ source .venv/bin/activate
```

Dependencies Using Pip

Create a requirements.txt file that will contains all dependencies we need to achieve this tutorial.

Listing 3.1: miniki/requirements.txt

```
Django >=1.8.2, <1.9
django-sslserver == 0.15
django-bower == 5.0.4
bleach == 1.4.1
Markdown == 2.6.2
hbp-app-python-auth==0.1.*
django-jsonify==0.3.0
python-social-auth==0.2.21
```

You can then install those dependencies using pip:

```
$ pip install -r requirements.txt
```

Edit settings.py

Modify the list of used application in *settings.py*

Listing 3.2: miniki/miniki/settings.py

```
INSTALLED_APPS = (
    'sslserver',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'miniki',
)
```

Also comments the X-Frame-Options middleware as the application will be loaded inside an iframe.

Listing 3.3: miniki/miniki/settings.py

```
MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    # 'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django.middleware.security.SecurityMiddleware',
)
```

Launch Server

Run the database migrations and launch the SSL server:

```
$ ./manage.py migrate
$ ./manage.py runsslserver

System check identified no issues (0 silenced).
June 18, 2015 - 07:31:27
Django version 1.8.2, using settings 'miniki.settings'
Starting development server at https://127.0.0.1:8000/
Using SSL certificate: /usr/local/lib/python2.7/site-packages/sslserver/certs/development.
 ↪crt
Using SSL key: /usr/local/lib/python2.7/site-packages/sslserver/certs/development.key
Quit the server with CONTROL-C.
```

If you access https://localhost:8000/admin, your browser will display a warning about an unsecure connection. This is because *django-sslserver* use a self signed certificate that your computer don't trust. The solution is to relaunch the server using a trusted certificate. You can generate one for your machine using *Create a local SSL Certificate*. If you follow the tutorial, you should be able to relaunch the server with this command:

```
$ ./manage.py runsslserver --certificate /path/to/certificate.crt --key /path/to/key.key
```

If you refresh https://localhost:8000/admin, you should now be able to display a web page without anymore warning from your browser. In the next step, we will create the wiki page *run* view and the wiki page *edit* view.

Edit and Run view

Most HBP Collaboratory application are composed of a *run view* and an *edit view*. In this step we will create both view and reference them in the Django *urls.py* file.

Todo

Link to the User Manual run/edit mode when it exists.

Create HTML Views

In this step we will create view placeholder and link them to the correct URL.

Create the *miniki/miniki/templates* directory and add the following three templates:

- *layout.html*: a global layout for all our views
- *show.html*: will display a wiki page content
- *edit.html*: will display a form to edit a wiki page

For now, those files will contains minimal content.

Listing 3.4: miniki/miniki/templates/layout.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Tutorial Application</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
  </head>
  <body>
    {% block content %}{% endblock %}
  </body>
</html>
```

Listing 3.5: miniki/miniki/templates/show.html

```
{% extends 'layout.html' %}

{% block content %}
```

```
<main class="page">
    <header class="page-header">
        <h1>Show Template</h1>
    </header>

    <section role="content">
        <code>{{context}}</code>
    </section>
</main>
{% endblock %}
```

Listing 3.6: miniki/miniki/templates/edit.html

```
{% extends 'layout.html' %}

{% block content %}
<main class="page">
    <header class="page-header">
        <h1>Edit Template</h1>
    </header>

    <section role="content">
        <code>{{context}}</code>
    </section>
</main>
{% endblock %}
```

Bind the templates to views

Create associated views in a new file called *views.py*

Listing 3.7: miniki/miniki/views.py

```
'''Views'''

from django.shortcuts import render_to_response
from uuid import UUID


def show(request):
    '''Render the wiki page using the provided context query parameter'''
    context = UUID(request.GET.get('ctx'))
    return render_to_response('show.html', {'context': context})


def edit(request):
    '''Render the wiki edit form using the provided context query parameter'''
    context = UUID(request.GET.get('ctx'))
    return render_to_response('edit.html', {'context': context})
```

link URL to the created views

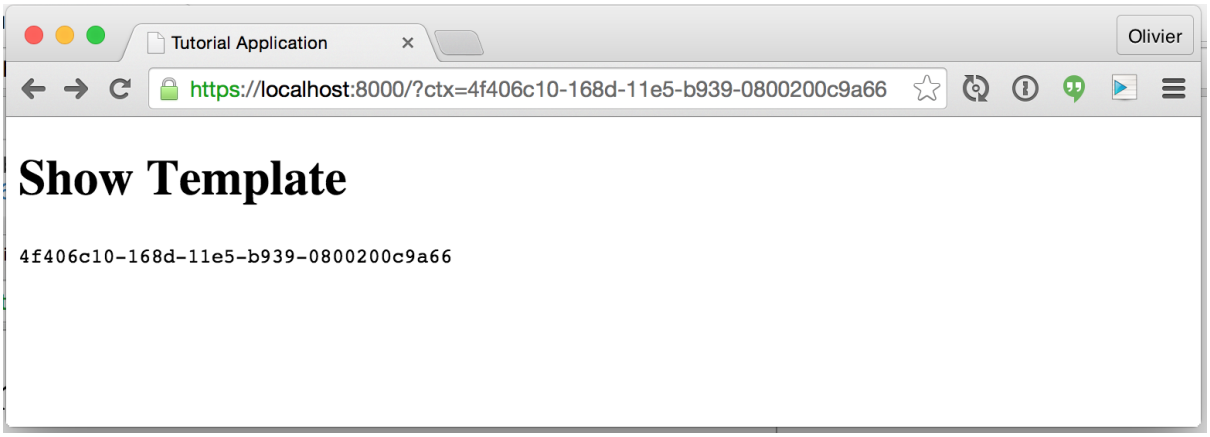
Edit *urls.py* to register the view to proper URL pattern file:

Listing 3.8: miniki/miniki/urls.py

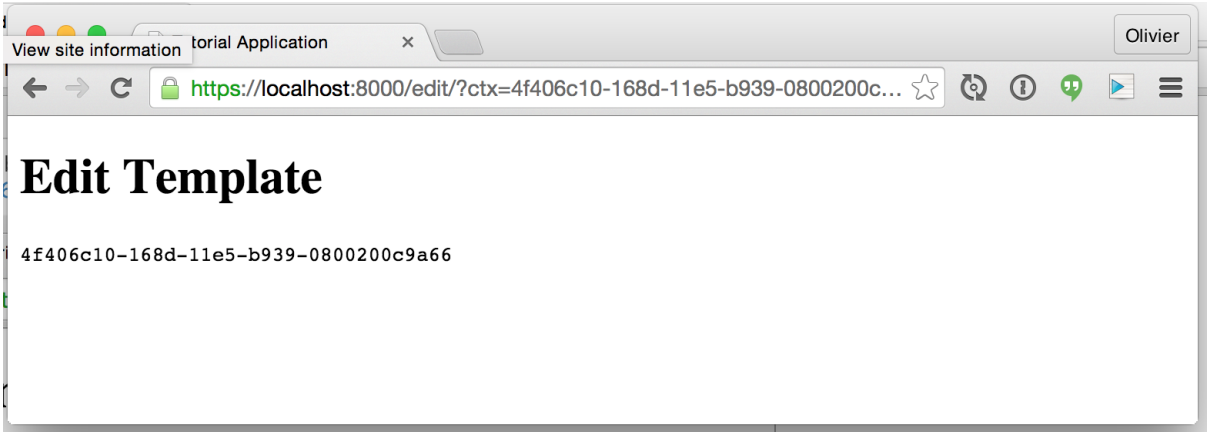
```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'^$', 'miniki.views.show', name='wiki_page_show'),
    url(r'^edit/$', 'miniki.views.edit', name='wiki_page_edit'),
]
```

A wiki page should be accessible at https://localhost:8000/?ctx=4f406c10-168d-11e5-b939-0800200c9a66.



This same page will be editable at https://localhost:8000/edit?ctx=4f406c10-168d-11e5-b939-0800200c9a66.

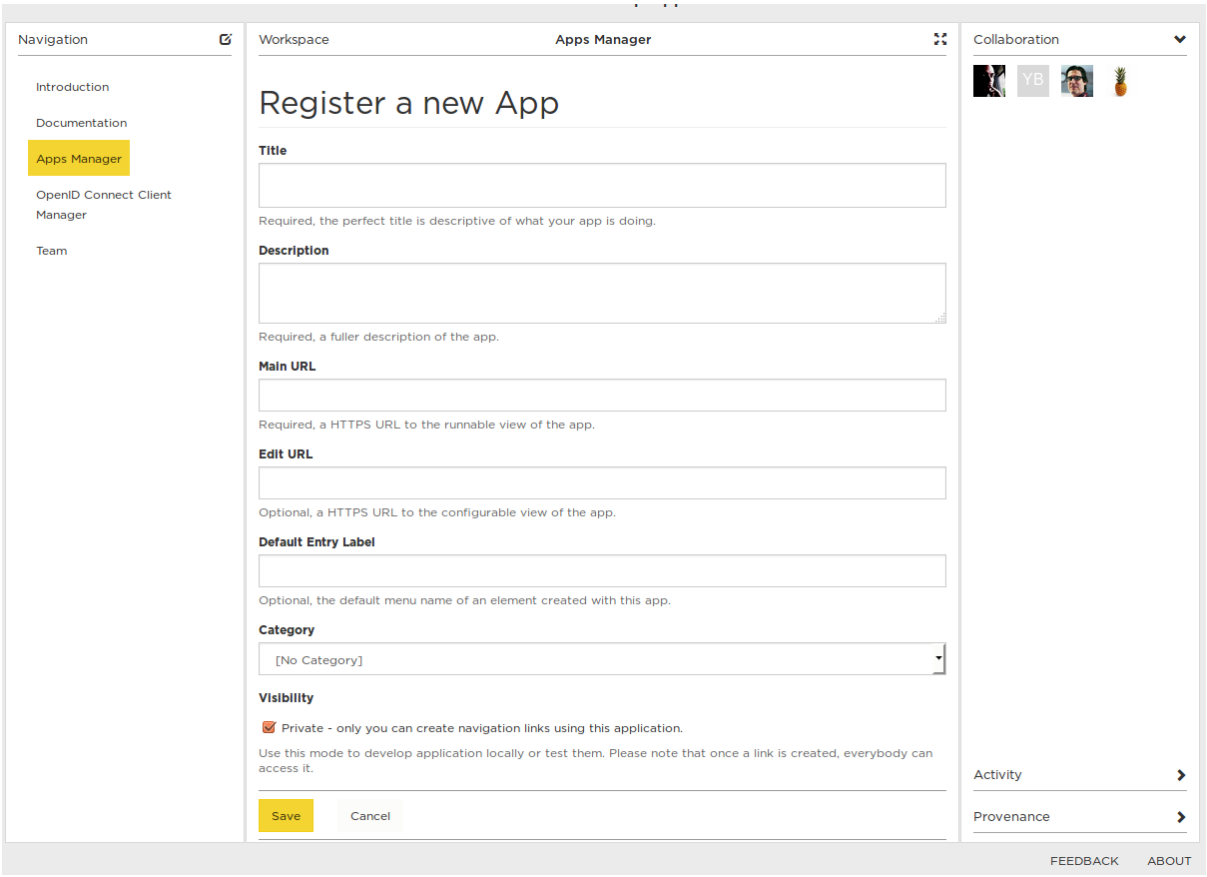


In the following steps, we will register the local server as an application within the HBP Collaboratory and create the WikiPage model.

Register Your Application

In this step we will register our development server as a private application. A private application can only be instantiated by its owner. It's purpose is to not pollute the public area while developing a new application.

To register an application, go to the Apps Manager app in the How to develop Apps collab and press 'Register App' button.



Enter the following parameters:

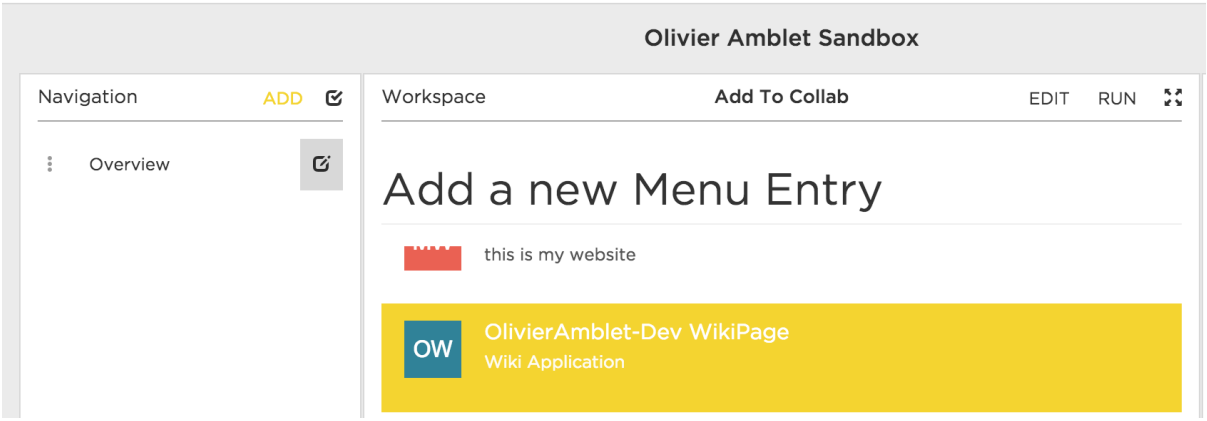
Name	Value	Description
Title	JohnDoe-Dev Wiki Page	Application name must be unique
Description	Wiki Application	A description that is displayed to collaborators
Main URL	https://localhost:8000/	The URL that will be used in *Run* mode
Edit URL	https://localhost:8000/edit	The URL that will be used in *Edit* mode
Default Entry Label		Leave empty, this is used to define a default title for the nav items
Category		Leave it empty, apps optionally can be grouped by categories
Private	[checked]	A private application can only be instantiated by the application owner

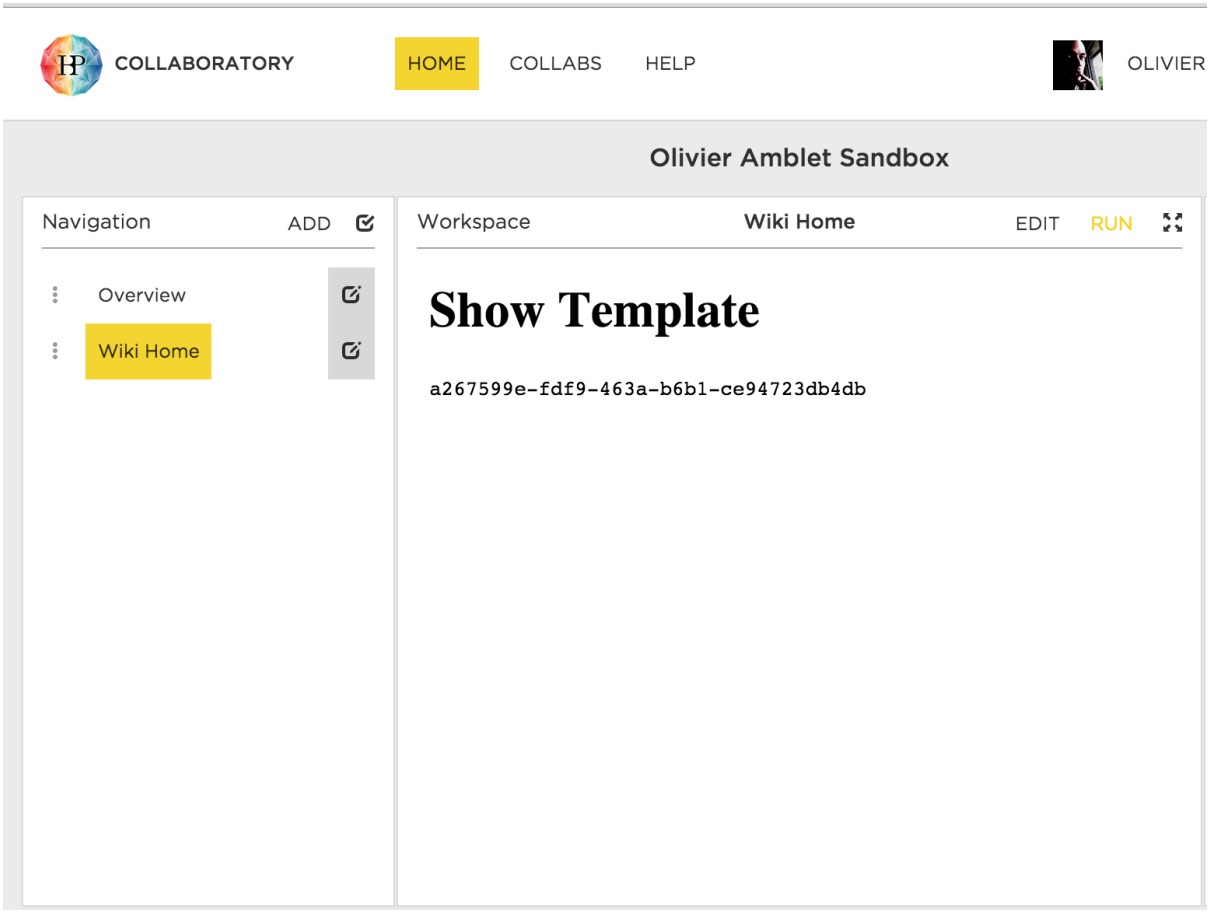
Then hit *save* to register your new application.

Loading the application

You can test the application from your sandbox collab, if you add a new item in the navigator, you can now choose your application in the list of available apps.

If your local server is running, you should be able to view the same page as in the previous step, but loaded within Collaboratory this time.


Olivier Amblet Sandbox
Navigation
ADD
Overview
Workspace
Add To Collab
EDIT
RUN
Add a new Menu Entry
this is my website
OW
OlivierAmblet-Dev WikiPage
Wiki Application


COLLABORATORY
HOME
COLLABS
HELP
OLIVIER
Olivier Amblet Sandbox
Navigation
ADD
Overview
Wiki Home
Workspace
Wiki Home
EDIT
RUN
Show Template
a267599e-fdf9-463a-b6b1-ce94723db4db

In the next steps, we will build the Django model representing a wiki page and use the Collaboratory theme to quickly build up a nice looking edit form.

Create WikiPage Model

Create a models.py file and add the WikiPage model to it.

Listing 3.9: miniki/miniki/models.py

```
from django.db import models
from django.core.urlresolvers import reverse


class WikiPage(models.Model):
    """A wiki page"""

    title = models.CharField(max_length=1024)
    text = models.TextField(help_text="formatted using ReST")
    # This field stores the UUID added as an argument by the Collaboratory.
    ctx = models.UUIDField(unique=True)
    created_on = models.DateTimeField(auto_now_add=True)

    def __unicode__(self):
        return self.title

    # UUIDField is not supported by automatic JSON serializer
    # so we add a method that retrieve a more convenient dict.
    def as_json(self):
        return {
            'title': self.title,
            'text': self.text,
            'ctx': str(self.ctx),
        }

    @models.permalink
    def get_absolute_url(self):
        return reverse('wiki_page_show', args=[str(self.ctx)])
```

The ctx field

This model may contains any fields corresponding to a wiki page. The only difference is that there is a HBP Collaboratory *Context* associated to it. The context is added to the URL of every requests made by the Collaboratory to your application and let you bind a Collaboratory location - an item of the navigation - to a specific context within your application. This linkage is most of the time the purpose of your application *edit view*.

Database Migration

Quit the Django server, run migration and relaunch.

```
$ ./manage.py migrate
$ ./manage.py runsslserver
```

Next

In the next step, we will use the HBP Collaboratory theme to create a nice looking edit form.

Create a Simple Edit Form

In this step we will create a Django Form and use it to edit our wiki page instances. We will use the HBP Collaboratory Theme to have a UI that is correctly integrated within the portal.

Create the Django form

Create the file named 'miniki/miniki/forms.py' and use the content below

Listing 3.10: miniki/miniki/forms.py

```
from django import forms
from .models import WikiPage

class WikiPageForm(forms.ModelForm):
    """Wiki Page edition form"""

    class Meta:
        model = WikiPage
        fields = ['title', 'text', 'ctx']
        widgets = {
            'ctx': forms.HiddenInput(),
            'title': forms.TextInput(attrs={
                'class': 'form-control',
                'ng-model': 'wikiPage.title',
            }),
            'text': forms.Textarea(attrs={
                'class': 'form-control form-control-editor',
                'ng-model': 'wikiPage.text',
            }),
        }
```

Using the Collaboratory Theme

We will use our first frontend dependencies. Collaboratory use bower as a package manage for frontend depdencies. *django-bower* is a library that provides tools to deal with bower dependencies (it has been declared in your requirements.txt file).

Require bower dependencies

Add the following definitions to *settings.py* file

Listing 3.11: miniki/miniki/settings.py

```
INSTALLED_APPS = (
    'sslserver',
    'django.contrib.admin',
```

```
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'miniki',
    'djangobower',
)

STATIC_URL = '/static/'

STATICFILES_FINDERS = (
    'django.contrib.staticfiles.finders.FileSystemFinder',
    'django.contrib.staticfiles.finders.AppDirectoriesFinder',
    'djangobower.finders.BowerFinder',
)

BOWER_COMPONENTS_ROOT = BASE_DIR

BOWER_INSTALLED_APPS = (
    'hbp-collaboratory-theme',
    'angular-hbp-common',
)
```

Add a *.bowerrc* at the project root which point to the HBP Bower Repository.

Listing 3.12: miniki/.bowerrc

```
{
    "directory": "bower_components",
    "registry": "http://bbpteam.epfl.ch/repository/bower"
}
```

You can then install or update dependencies using the following command:

```
$ ./manage.py bower install
```

Load the Collaboratory theme

We can now load the Collaboratory theme in *layout.html*

Listing 3.13: miniki/miniki/templates/layout.html

```
{% load static %}

<!DOCTYPE html>
<html>
  <head>
    <title>Tutorial Application</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">

    {# Bower Dependencies #}
    <link rel="stylesheet" href="{% static 'hbp-collaboratory-theme/dist/css/bootstrap.css
↪' %}">
  </head>
```

```
  <body>
    {% block content %}{% endblock %}

    {# Bower Dependencies #}
    <script type="text/javascript" src="{% static 'jquery/dist/jquery.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'lodash/dist/lodash.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular/angular.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-bootstrap/ui-bootstrap-tpls.
 ↪min.js' %}"></script>
  </body>
</html>
```

Update Edit View

Let's replace our dummy edit view by a more complete version that actually handle a wiki page instance.

Listing 3.14: miniki/miniki/views.py (edit function)

```
'''Views'''

from django.shortcuts import render_to_response, render, redirect
from django.core.urlresolvers import reverse
from uuid import UUID

import bleach

from .forms import WikiPageForm
from .models import WikiPage


def edit(request):
    '''Render the wiki edit form using the provided context query parameter'''

    context = UUID(request.GET.get('ctx'))
    # get or build the wiki page
    try:
        wiki_page = WikiPage.objects.get(ctx=context)
    except WikiPage.DoesNotExist:
        wiki_page = WikiPage(ctx=context)

    if request.method == 'POST':
        form = WikiPageForm(request.POST, instance=wiki_page)
        if form.is_valid():
            wiki_page = form.save(commit=False)
            # Clean up user input
            wiki_page.text = bleach.clean(wiki_page.text)
            wiki_page.save()
    else:
        form = WikiPageForm(instance=wiki_page)

    return render(request, 'edit.html', {'form': form, 'ctx': str(context)})

def _reverse_url(view_name, context_uuid):
    """generate an URL for a view including the ctx query param"""
```

```
    return '%s?ctx=%s' % (reverse(view_name), context_uuid)
```

The new *edit.html* template feature a form using the Collaboratory Theme which is based on Twitter Bootstrap.

Listing 3.15: miniki/miniki/templates/edit.html

```
{% extends 'layout.html' %}

{% block content %}
<form action="" class="wiki-form" method="POST">
    <div class="wiki-form-fields">
        {% csrf_token %}
        {{ form.non_field_errors }}
        {{ form.ctx }}
        <div class="form-group">
            {{ form.title.label_tag }}
            {{ form.title }}
            <p class="help-text">{{ form.title.errors }}</p>
        </div>
        <div class="form-group form-group-editor">
            {{form.text.label_tag}}
            {{form.text}}
            <p class="help-text">{{ form.text.errors }}</p>
        </div>
    </div>

    <div class="navbar navbar-form">
        <button type="submit" class="btn btn-primary">Save</button>
    </div>
</form>
{% endblock %}
```

Update Run View

We can replace the show view as well in order to have the real content displayed.

Listing 3.16: miniki/miniki/views.py (edit function)

```
'''Views'''

from django.shortcuts import render_to_response, render, redirect
from django.core.urlresolvers import reverse
from uuid import UUID
from markdown import markdown

import bleach

from .forms import WikiPageForm
from .models import WikiPage


def show(request):
    '''Render the wiki page using the provided context query parameter'''
    context = UUID(request.GET.get('ctx'))
```

```
    try:
        wiki_page = WikiPage.objects.get(ctx=context)
        content = markdown(wiki_page.text)
    except WikiPage.DoesNotExist:
        wiki_page = None
        content = ''
    return render_to_response('show.html', {'wiki_page': wiki_page, 'content': content})

def edit(request):
    '''Render the wiki edit form using the provided context query parameter'''

    context = UUID(request.GET.get('ctx'))
    # get or build the wiki page
    try:
        wiki_page = WikiPage.objects.get(ctx=context)
    except WikiPage.DoesNotExist:
        wiki_page = WikiPage(ctx=context)

    if request.method == 'POST':
        form = WikiPageForm(request.POST, instance=wiki_page)
        if form.is_valid():
            wiki_page = form.save(commit=False)
            # Clean up user input
            wiki_page.text = bleach.clean(wiki_page.text)
            wiki_page.save()
    else:
        form = WikiPageForm(instance=wiki_page)

    return render(request, 'edit.html', {'form': form, 'ctx': str(context)})

def _reverse_url(view_name, context_uuid):
    """generate an URL for a view including the ctx query param"""
    return '%s?ctx=%s' % (reverse(view_name), context_uuid)
```

The new *show.html* template feature the wiki page title and text content, formatted using markdown.

Listing 3.17: miniki/miniki/templates/show.html

```
{% extends "layout.html" %}

{% block content %}
<main class="page">
    {% if wiki_page %}
    <header class="page-header">
        <h1>{{wiki_page.title}}</h1>
    </header>

    <section role="content">
        {{content|safe}}
    </section>
    {% else %}
    <header class="page-header">
        <h1>This page does not exists</h1>
    </header>
    <section role="content">
        <p>Go to edit mode to create it.</p>
```

```
    </section>
    {% endif %}
</main>
{% endblock %}
```

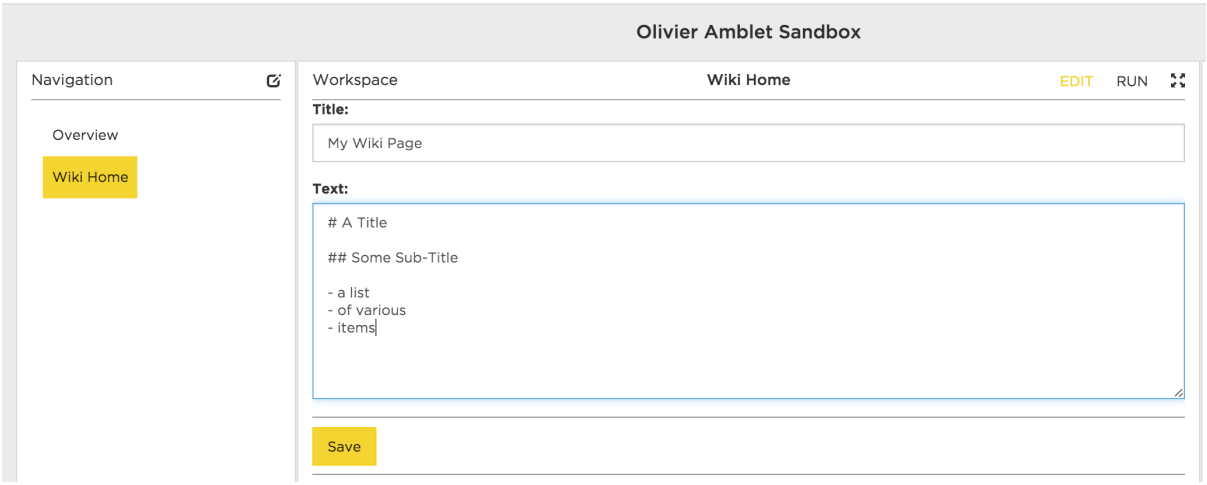


Fig. 3.1: The edit view

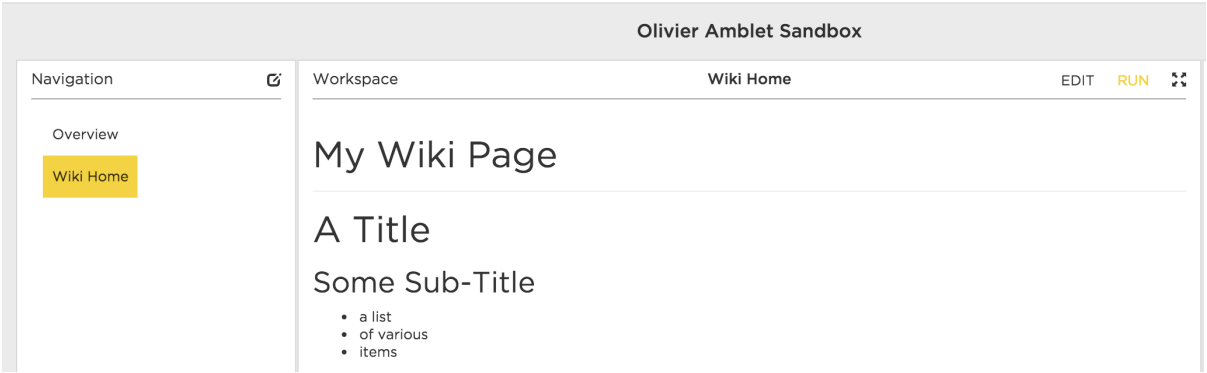


Fig. 3.2: The run view once the content has been saved.

Next

In the next steps we will add authentication and authorization to the wiki page resource, using the HBP OIDC service.

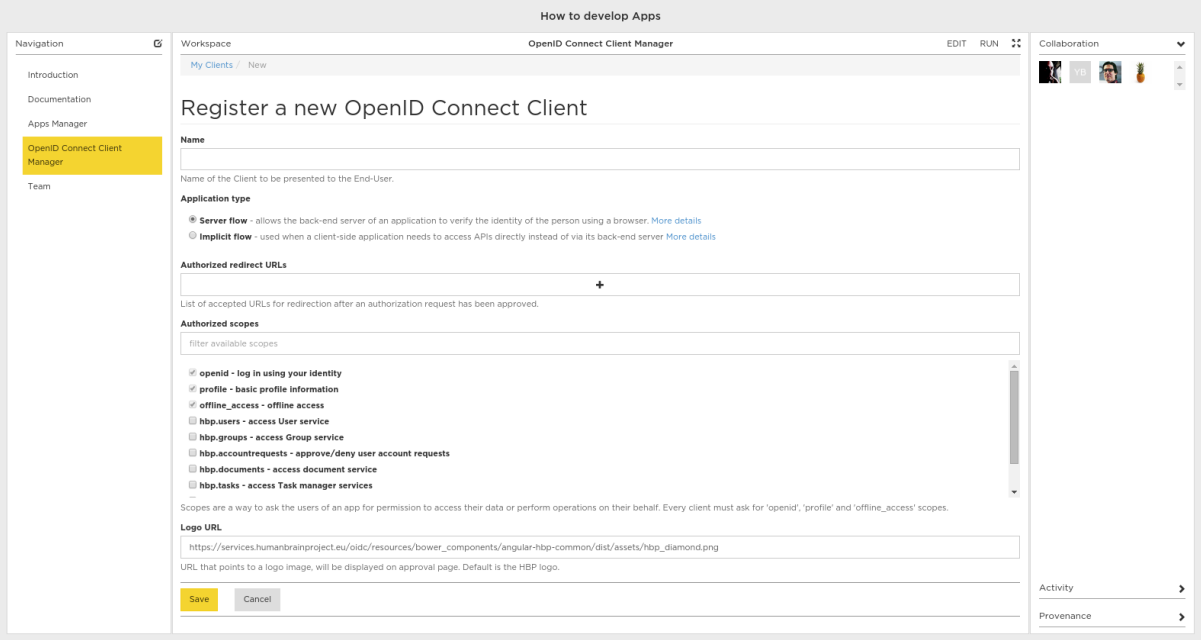
Authentication

Our wiki works but it is open to everyone to read and write content (which can be seen as a condition to be called a wiki). In our case we want to prevent access to members outside of the HBP and only authorize the members of a Collab to edit wiki pages written in this collab.

In this section, we will discuss how we manage the authentication part using the HBP Authentication Service based on OpenID Connect protocol.

OIDC application

To use the OpenID Connect protocol on the server, you need to create an OIDC client. To register a client, go to the OIDC Client Manager app in the *How to develop Apps* collab and press 'Create new Client' button.



Enter the following parameters:

Name	Value	Description
Name	JohnDoe-Dev Wiki Page	Client name will be shown in the client approval page
Application type	check *Server flow*	Authentication will be performed server side
Authorized redirect URL	https://localhost:8000/complete/hbp	The authentication token will be sent only to this url
Authorized scopes	check *hbp.collab*	This defines the set of apis that the app will request access to
Logo URL	(default value)	This logo will be shown in the approval page, it must be a public URL

Then press 'Save' to create the client. In the following page, you will find the Client ID and the Client Secret that you will need in the next steps of the tutorial.

python-social-auth plugin

We will use the python-social-auth plugin in order to implement authentication with HBP OIDC server. We first need to load the plugin and configure it with a few constants.

Private settings

Client ID and Client Secret are private data and should not be committed in your version control system. Create the config file in *miniki/miniki/config.py* and ensure it is ignored by your version

control (eg: add it to the .gitignore file).

Listing 3.18: miniki/miniki/config.py (authentication)

```
import hbp_app_python_auth.settings as auth_settings
auth_settings.SOCIAL_AUTH_HBP_KEY = '...provided-client-id...'
auth_settings.SOCIAL_AUTH_HBP_SECRET = '...provided-client-secret...'
```

Name	Description
SOCIAL_AUTH_HBP_KEY	Your Client ID
SOCIAL_AUTH_HBP_SECRET	Your Client Secret (keep it secret)

Public settings

Listing 3.19: miniki/miniki/settings.py (authentication)

```
from config import *

INSTALLED_APPS = (
    'sslserver',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'miniki',
    'djangobower',
    'social.apps.django_app.default',
    'hbp_app_python_auth',
)

MIDDLEWARE_CLASSES = (
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.auth.middleware.SessionAuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    # 'django.middleware.clickjacking.XFrameOptionsMiddleware',
    'django.middleware.security.SecurityMiddleware',
    'social.apps.django_app.middleware.SocialAuthExceptionMiddleware',
)

AUTHENTICATION_BACKENDS = (
    'hbp_app_python_auth.auth.HbpAuth',
    'django.contrib.auth.backends.ModelBackend',
)
```

Add the Python Social Auth URL

Python social auth needs a few endpoints to handle the login, logout and other redirection which are part of oAuth 2.

Listing 3.20: miniki/miniki/urls.py (authentication)

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url('', include('social.apps.django_app.urls', namespace='social')),
    url('', include('hbp_app_python_auth.urls', namespace='hbp-social')),
    url(r'^$', 'miniki.views.show', name='wiki_page_show'),
    url(r'^edit/$', 'miniki.views.edit', name='wiki_page_edit'),
]
```

Protect the views

Now that our authentication layer is configured properly, we can actually announce that both our views are protected by annotating them. Edit the *views.py* file as follow:

Listing 3.21: miniki/miniki/views.py (authentication)

```
from django.contrib.auth.decorators import login_required

@login_required(login_url='/login/hbp')
def show(request):
    '''Render the wiki page using the provided context query parameter'''
    context = UUID(request.GET.get('ctx'))
    try:
        wiki_page = WikiPage.objects.get(ctx=context)
        content = markdown(wiki_page.text)
    except WikiPage.DoesNotExist:
        wiki_page = None
        content = ''
    return render_to_response('show.html',
                              {'wiki_page': wiki_page, 'content': content})


@login_required(login_url='/login/hbp')
def edit(request):
    '''Render the wiki edit form using the provided context query parameter'''

    if not _is_collaborator(request):
        return HttpResponseForbidden()

    context = UUID(request.GET.get('ctx'))
    # get or build the wiki page
    try:
        wiki_page = WikiPage.objects.get(ctx=context)
    except WikiPage.DoesNotExist:
        wiki_page = WikiPage(ctx=context)

    if request.method == 'POST':
        form = WikiPageForm(request.POST, instance=wiki_page)
        if form.is_valid():
            wiki_page = form.save(commit=False)
            # Clean up user input
```

```
            wiki_page.text = bleach.clean(wiki_page.text)
            wiki_page.save()
    else:
        form = WikiPageForm(instance=wiki_page)

    return render(request, 'edit.html', {'form': form, 'ctx': str(context)})
```

Edit also the layout.html template to include a snippet code responsible for ensuring that the logged in user is the same as the Collaboratory user where the app is running.

Listing 3.22: miniki/miniki/templates/layout.html

```
{% load static %}
{% load oidc_session %}

<!DOCTYPE html>
<html>
  <head>
    <title>Tutorial Application</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">

    {# Bower Dependencies #}
    <link rel="stylesheet" href="{% static 'hbp-collaboratory-theme/dist/css/bootstrap.css
 ↪' %}">
  </head>
  <body>
    {% hbp_oidc_session_handler request.user %}

    {% block content %}{% endblock %}

    {# Bower Dependencies #}
    <script type="text/javascript" src="{% static 'jquery/dist/jquery.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'lodash/dist/lodash.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular/angular.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-bootstrap/ui-bootstrap-tpls.
 ↪min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-bbp-config/angular-bbp-config.
 ↪js' %}"></script>
    <script type="text/javascript" src="{% static 'bbp-oidc-client/angular-bbp-oidc-
 ↪client.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-ui-router/release/angular-ui-
 ↪router.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-resource/angular-resource.min.
 ↪js' %}"></script>
    <script type="text/javascript" src="{% static 'marked-hbp/marked.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-hbp-common/dist/angular-hbp-
 ↪common.min.js' %}"></script>

    {# App Specific #}
    <script type="text/javascript" src="{% static 'scripts/app.js' %}"></script>
  </body>
</html>
```

The next time you will access a wiki page from within the Collaboratory, HBP Authentication will ask you to authorize *miniki* app to access your user informations. If you are not logged in Collaboratory already, a new authentication challenge will be provided as well.

At this point, the access to *miniki* is restricted to authenticated users. In the next step, we will see how to restrict it further.

Todo

link to HBP Authentication Service

Adding Authorization

Now that we can authenticate our users, it is time to ensure they have the correct access level to access a resource. Authentication is the responsability of the *HBP Authentication Service* but Authorization is every app duty. We will ensure that only HBP Collaboratory members can edit a wiki page. To achieve this task we will use the Collaboratory REST service.

Define Collab Service Settings

Add the following constant declaration to *settings.py*.

Listing 3.23: miniki/miniki/settings.py (authorization)

```
HBP_COLLAB_SERVICE_URL = 'https://services.humanbrainproject.eu/collab/v0/'
```

Check Permissions

Edit *views.py* to add a permission check function that will call the web service to retrieve the collab for the current context and the user authorization on the current collab. If the current user is a member of the collab, then we will authorize access to the edit page. Otherwise, we return a 403 error.

Listing 3.24: miniki/miniki/views.py (authorization)

```
from django.conf import settings
from django.http import HttpResponseForbidden

from hbp_app_python_auth.auth import get_access_token, get_token_type, get_auth_header
import hbp_app_python_auth.settings as auth_settings


@login_required(login_url='/login/hbp')
def edit(request):
    '''Render the wiki edit form using the provided context query parameter'''

    if not _is_collaborator(request):
        return HttpResponseForbidden()

    context = UUID(request.GET.get('ctx'))
    # get or build the wiki page
    try:
        wiki_page = WikiPage.objects.get(ctx=context)
    except WikiPage.DoesNotExist:
        wiki_page = WikiPage(ctx=context)
```

```
    if request.method == 'POST':
        form = WikiPageForm(request.POST, instance=wiki_page)
        if form.is_valid():
            wiki_page = form.save(commit=False)
            # Clean up user input
            wiki_page.text = bleach.clean(wiki_page.text)
            wiki_page.save()
    else:
        form = WikiPageForm(instance=wiki_page)

    return render(request, 'edit.html', {'form': form, 'ctx': str(context)})


def _is_collaborator(request):
    '''check access depending on context'''

    svc_url = settings.HBP_COLLAB_SERVICE_URL

    context = request.GET.get('ctx')
    if not context:
        return False
    url = '%scollab/context/%s/' % (svc_url, context)
    headers = {'Authorization': get_auth_header(request.user.social_auth.get())}
    res = requests.get(url, headers=headers)
    if res.status_code != 200:
        return False
    collab_id = res.json()['collab']['id']
    url = '%scollab/%s/permissions/' % (svc_url, collab_id)
    res = requests.get(url, headers=headers)
    if res.status_code != 200:
        return False
    return res.json().get('UPDATE', False)
```

If you try to edit a wiki page with a context parameter belonging to a collab that did not belongs to you, it will now retrieve a 403 Forbidden response.

More

- Collab REST Service API Documentation

Next

In the next section, we will add a Markdow preview using AngularJS and the angular-hbp-common library. It will let us bootstrap our application to support modern reactive Web Application where most of the work is done in the client.

Preview Markdown with AngularJS

In this step we will bootstrap AngularJS with all needed configurations to enable all the functionality available in angular-hbp-common. This library give you access to common UI components, filters and service clients of the HBP platform (see angular-hbp-common API Documentation).

Environement

To work properly, the angular-hbp-common library needs access to environments data. It uses the angular-bbp-config library to retrieve them (see angular-bbp-config API Documentation). This library read the configuration from a global variable *window.bbpConfig*. We have to write a script that will set this variable before launching the Angular application.

To retrieve the proper environments, we will reuse the one define at https://collab.humanbrainproject.eu/config.json and inject the user token in it.

To do that, we need to configure a new view that will build this json reprensentation and create a javascript launch script that will bootstrap our angular application.

Define the /config.json URL

Edit *urls.py* to configure the new URL:

Listing 3.25: miniki/miniki/urls.py (add config.json URL)

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url('', include('social.apps.django_app.urls', namespace='social')),
    url('', include('hbp_app_python_auth.urls', namespace='hbp-social')),
    url(r'^$', 'miniki.views.show', name='wiki_page_show'),
    url(r'^edit/$', 'miniki.views.edit', name='wiki_page_edit'),
    url(r'^config.json$', 'miniki.views.config', name='config'),
]
```

Define the config view

Define a new *config* function that will load the official config.json and tailor it to fit this application. The parameters that needs to be set are defined below:

Parameter name	Description
auth.client_id	This application OpenID Connect identifier
auth.token	The user token describe as an object containing the following keys: access_token, token_type, expires_in and scopes

Define the URL of the HBP environment URL in *settings.py*:

Listing 3.26: miniki/miniki/settings.py (add config.json URL)

```
HBP_ENV_URL = 'https://collab.humanbrainproject.eu/config.json'
```

The new view is written in *views.py*:

Listing 3.27: miniki/miniki/views.py (add config function)

```
@login_required(login_url='/login/hbp')
def config(request):
    '''Render the config file'''

    res = requests.get(settings.HBP_ENV_URL)
    config = res.json()

    # Use this app client ID
    config['auth']['clientId'] = settings.SOCIAL_AUTH_HBP_KEY

    # Add user token informations
    request.user.social_auth.get().extra_data
    config['auth']['token'] = {
        'access_token': get_access_token(request.user.social_auth.get()),
        'token_type': get_token_type(request.user.social_auth.get()),
        'expires_in': request.session.get_expiry_age(),
    }

    return HttpResponse(json.dumps(config), content_type='application/json')
```

Bootstrap Script

The bootstrap script load */config.json* and then set the result to *window.bbpConfig* accordingly to *angular-bbp-config* specification. Once done, it boostraps an AngularJS application.

Listing 3.28: miniki/miniki/static/scripts/app.js (bootstrap code)

```
(function() {

// Define the miniki application, currently doing nothing.
angular.module('miniki', ['hbpCommon']);

// Bootstrap function
angular.bootstrap().invoke(function($http, $log) {
  $http.get('/config.json').then(function(res) {
    window.bbpConfig = res.data;
    angular.element(document).ready(function() {
      angular.bootstrap(document, ['miniki']);
    });
  }, function() {
    $log.error('Cannot boot miniki application');
  });
});

}());
```

Load dependencies

Load all necessary dependencies in the layout page:

Listing 3.29: miniki/miniki/templates/layout.html (js dependencies)

```
{% load static %}
{% load oidc_session %}

<!DOCTYPE html>
<html>
  <head>
    <title>Tutorial Application</title>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">

    {# Bower Dependencies #}
    <link rel="stylesheet" href="{% static 'hbp-collaboratory-theme/dist/css/bootstrap.css
 ↪' %}">
  </head>
  <body>
    {% hbp_oidc_session_handler request.user %}

    {% block content %}{% endblock %}

    {# Bower Dependencies #}
    <script type="text/javascript" src="{% static 'jquery/dist/jquery.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'lodash/dist/lodash.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular/angular.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-bootstrap/ui-bootstrap-tpls.
 ↪min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-bbp-config/angular-bbp-config.
 ↪js' %}"></script>
    <script type="text/javascript" src="{% static 'bbp-oidc-client/angular-bbp-oidc-
 ↪client.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-ui-router/release/angular-ui-
 ↪router.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-resource/angular-resource.min.
 ↪js' %}"></script>
    <script type="text/javascript" src="{% static 'marked-hbp/marked.min.js' %}"></script>
    <script type="text/javascript" src="{% static 'angular-hbp-common/dist/angular-hbp-
 ↪common.min.js' %}"></script>

    {# App Specific #}
    <script type="text/javascript" src="{% static 'scripts/app.js' %}"></script>
  </body>
</html>
```

The AngularJS Application

Time to write the AngularJS controller that will handle the displying of a HTML preview of the Markdown formatted wiki text.

The WikiPageFormCtrl

Angular controller handle application states. We can add the *WikiPageFormCtrl* to *app.js*:

Listing 3.30: miniki/miniki/static/scripts/app.js (WikiPageFormCtrl)

```
(function() {

// Define the miniki application
angular.module('miniki', ['hbpCommon'])
.controller('wikiPageForm', function($scope) {
  // The form controller that manage the displays of preview.
  $scope.isPreviewOpen = false;
  $scope.togglePreview = function () {
    $scope.isPreviewOpen = !$scope.isPreviewOpen;
  };
});


// Bootstrap function
angular.bootstrap().invoke(function($http, $log) {
  $http.get('/config.json').then(function(res) {
    window.bbpConfig = res.data;
    angular.element(document).ready(function() {
      angular.bootstrap(document, ['miniki']);
    });
  }, function() {
    $log.error('Cannot boot miniki application');
  });
});

}());
```

Update edit template

We can now edit the template to integrate the backend with the frontend.

Listing 3.31: miniki/miniki/templates/edit.html (angular markdown preview)

```
{% extends 'layout.html' %}

{% load jsonify %}

{% block content %}
<form action="" class="wiki-form" method="POST"
      ng-controller="wikiPageForm" ng-init='wikiPage = {{form.instance.as_json|jsonify}}'>
    <div class="wiki-form-fields"
         ng-show="!isPreviewOpen">
        {% csrf_token %}
        {{ form.non_field_errors }}
        {{ form.ctx }}
        <div class="form-group">
            {{ form.title.label_tag }}
            {{ form.title }}
            <p class="help-text">{{ form.title.errors }}</p>
        </div>
        <div class="form-group form-group-editor">
            {{form.text.label_tag}}
            {{form.text}}
```

```
            <p class="help-text">{{ form.text.errors }}</p>
        </div>
    </div>

    <div class="wiki-form-preview"
         ng-show="isPreviewOpen">
        <header class="page-header">
            <h1 ng-bind="wikiPage.title"></h1>
        </header>

        <section role="content" ng-bind-html="wikiPage.text|hbpMarkdown"></section>
    </div>

    <div class="navbar navbar-form">
        <button type="submit" class="btn btn-primary">Save</button>
        <button type="button" class="btn btn-default"
            ng-click="togglePreview()"
            ng-bind="isPreviewOpen ? 'Editor' : 'Preview'"></button>
    </div>
</form>
{% endblock %}
```

Load the jsonify library in Django by adding it to the list of applications:

Listing 3.32: miniki/miniki/settings.py (add config.json URL)

```
INSTALLED_APPS = (
    'sslserver',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'miniki',
    'djangobower',
    'social.apps.django_app.default',
    'jsonify',
)
```

Results

If you access a wiki page and switch to edit mode, you can know enable a preview using the *Preview* button and switch back to the editor. Moreover, you can use all the existing services which requires client side authentication as the backend token is registered in the frontend as well.

Next

You probably owe yourself a little break here. Feel free to send any comments about this tutorial at the following address: bbp-ou-platformdev@groupes.epfl.ch

Overview

As an example, we will create a minimal Django Wiki application that integrates in the HBP Collaboratory.

This tutorial shows how to create a very minimal Wiki and how to integrate into HBP Collaboratory using Django on the backend, AngularJS on the frontend. The first part is a condensed version of the work to create any Django application while the remaining dive into Collaboratory specific features like binding resources to a context, authentication and authorization.

We'll assume you have Django 1.8+ installed already. You can tell Django is installed and which version by running the following command:

```
$ python -c "import django; print(django.get_version())"
```

Note: This tutorial assume you have basic knowledge of Django. The following guide will reproduce all the basic steps needed to setup a new Django project without further explanation. You should have completed the Django Tutorial to understand this tutorial.

The Getting Started guide is divided in two parts:

A simple example demonstrates the "hello world" application and can be completed in about 30 minutes.

Create your first application is a bit longer and showcase the making of a Django application that will use more integration point and is a good starting point for a real world application. One should expect one day of work to complete it.

Security

Todo

Explains how to secure backend and frontend

Scopes

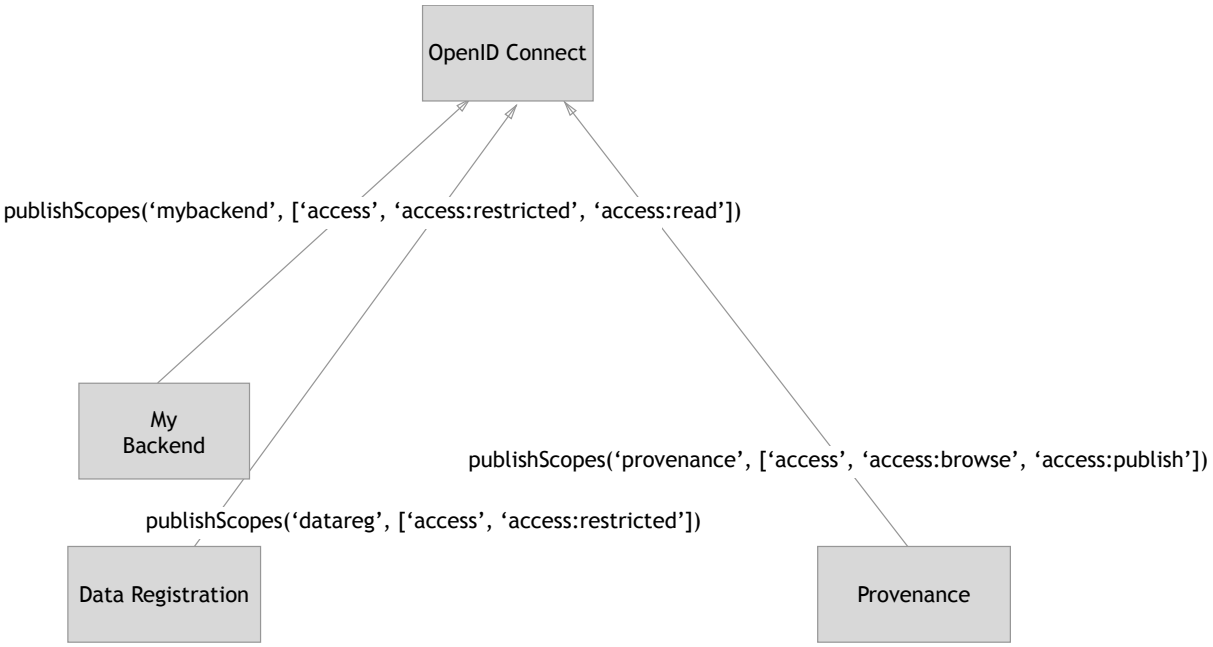
HBP Stream API Documentation

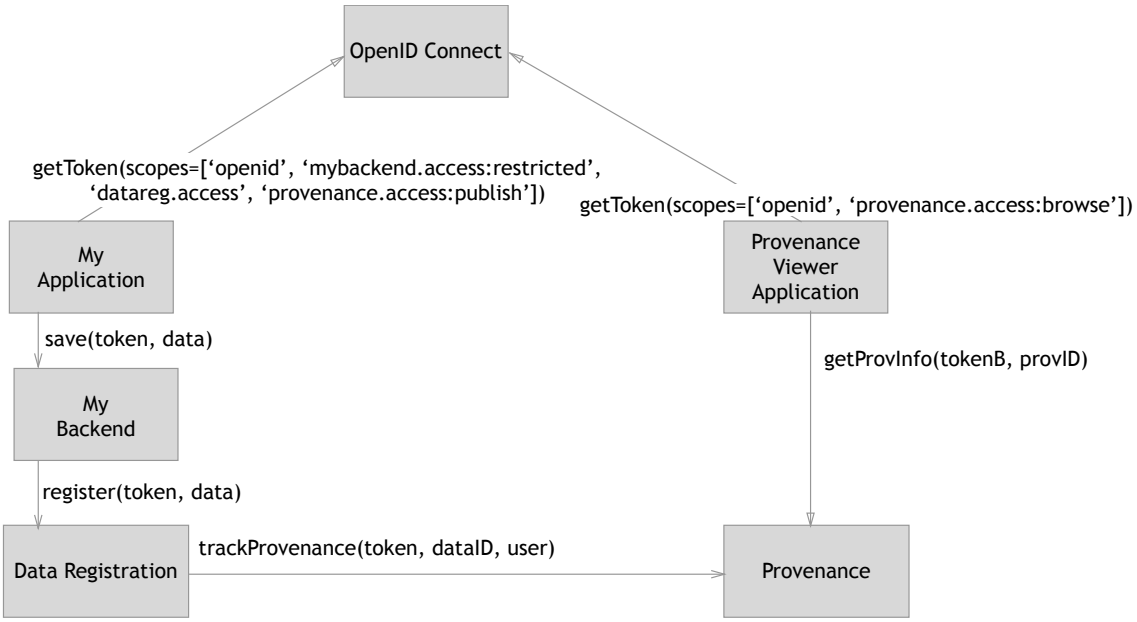
Contents:

Notification API

Any HBP Application can use the Notification REST API to send notifications targeted to users.

The goal of the *Notification API* is to inform a specific user of activities that they should be aware of as quickly as possible, either because they need to take an action or because this is potentially important for their work.


OpenID Connect
publishScopes('mybackend', ['access', 'access:restricted', 'access:read'])
My Backend
publishScopes('provenance', ['access', 'access:browse', 'access:publish'])
publishScopes('datareg', ['access', 'access:restricted'])
Data Registration
Provenance


OpenID Connect
getToken(scopes=['openid', 'mybackend.access:restricted', 'datareg.access', 'provenance.access:publish'])
getToken(scopes=['openid', 'provenance.access:browse'])
My Application
Provenance Viewer Application
save(token, data)
getProvInfo(tokenB, provID)
My Backend
register(token, data)
Data Registration
trackProvenance(token, dataID, user)
Provenance

Each application has the responsability to carefully choose which activities are important to send to a specific user. For more generic activities, the *Activity API* will provide a more verbose interface for general information aggregation and distribution to user feeds.

Is it an Activity or a Notification?

> A notification system commonly shows activity related to your account. Whereas an activity stream shows activity by the people you follow *[CIT001]*.

This good definition might translate in Collaboratory as: a notification system commonly shows activity related to the user. Whereas an activity stream show activity by the people in the collabs you are part of. As a generale rule of thumb, notify when people needs to take an action or that an asynchronous event occurs because of a user action.

OpenID Connect Scopes

To use the *Notification API*, an application needs to generate a token with specific scopes (`hbp.notification.self` or `hbp.notification.all`). More information about how to generate such a client is available in *Manage OpenID Connect Clients*.

- `hbp.notification.self`: This scope allow the application to notify the current user only. It is less permissive and ensure that the application cannot spread unwanted messages to anybody on the behalf of the current user. Any application can request this scope by accessing *Manage OpenID Connect Clients*.
- `hbp.notification.all`: This scope allows the application to notify any HBP users. One must be very careful when sending notifications to avoid flooding users with unwanted messages. Consider using activities instead of notifications where appropriate. To obtain this scope for your client, please follow the procedure describe in *Ask for restricted scopes*

Create a new Notification

You can easily create a new notification using the following REST call:

```
POST https://stream.humanbrainproject.eu/api/v0/notifications/

Authorization:Bearer HERE_COME_THE_USER_TOKEN
Accept: application/json
Content-Type: application/json


{
  "summary": "Ping"
  "targets": [{
    "type": "HBPUser",
    "id": "233444"
  }],
  "object": {
    "type": "HBPCollaboratoryContext",
    "id": "24497C98-C399-4FDE-B5FA-28B15E79598D"
  }
}
```

This will send a notification with the text "Ping" to the user with id "233444". A link will be provided to the user to navigate to the notification object ; in this case a Collaboratory page identified by its context. You can use HBPGroup as a "type" for target and group name as "id" to send notification to everybody from group. `hbp.notification.all` scope is required in this case.

Activity Stream Service

Any HBP Application can use the Activity Stream REST API to report activities and retrieve activity streams.

What is an Activity

Activity describes some kind of action. It has the following attributes:

- **Actor** - primary actor for the activity
- **Verb** - identifies the action that the activity describes
- **Object** - describes the primary object of the activity
- **Time** - when activity happened
- **Summary** - message describing the activity
- **Target** - describes the indirect object, or target, of the activity. Will often be the object of the English preposition "to". Optional

ObjectReference

An object reference describes objects living somewhere within the HBP. ObjectReference is used as the type for Actor, Object or Target for the Activity. An application that works with object reference should know how to retrieve most of them, and support not knowing some of them. If an application doesn't know how to resolve an object reference to the real instance, it should handle the exception properly and avoid to block the application for this. ObjectReference Attributes:

- **type** - String, some examples are HBPUser, HBPContext, HBPApp
- **id** - String
- **state** - String that an application use to display an object reference in the most appropriate form. The object type should advertise what is the format of the state parameter.

What is an Activity Stream

Activity Stream is a set of activities related to some context. For example, Collab Activity Stream will contain all activities related to a particular *Collab*. Activity Stream is defined by *type* and *id* where

- **type** - free form string like HBPUser or HBPCollab. HBP prefix is used for generic Collaboratory types like HBPUser, HBPCollab or HBPCollabContext
- **id** - id of the corresponding entity, for example, user id or collab id

Activity Propagation

When some Activity is reported in the system, it automatically propagates to the corresponding Activity Streams. For example, Activity like

```
UserA added UserB to collab Collab
```

will appear in Activity Streams for UserA, UserB and Collab.

Also Activity is automatically propagated to every collab member in case *object* or *target* has type **HBPCollab**. For **HBPCollabContext** type of *object* or *target* activity automatically propagated to corresponding collab stream.

Streams propagate Activities employing an asynchronous job. It means there can be delay between posting Activity and the time it appears in the corresponding streams.

Summary preprocessing

There is special support for placeholders in Activity Summary. The main reason for it is that sometimes there is not enough information on the service side to form user-friendly summary text. For example, on a service side, there is only user id for the actor. It does not make sense to send a request to Identity APIs to get the corresponding user id. Instead, it can be done asynchronously in Activity Service. Next placeholders are supported - {{actor}}, {{object}} and {{target}}. Before saving activity to the database these placeholders will be automatically substituted by proper text which is:

- @username for **HBPUser** type
- collab title for **HBPCollab** type
- context name for **HBPCollabContext** type
- type<id> for all other types

During placeholders substitutions, the corresponding indices saved to provide a way to do custom rendering on frontend side. For the details check /apidoc/stream-api

Access Control

For some known types, like User or Collab, an appropriate access control is implemented during stream access. When the stream type is not understandable by the system, no access control guarantees are provided.

Aggregation

TODO

API

Detailed API description can be found /apidoc/stream-api

API Documentation

API Endpoints

API Models

Activity Stream API Endpoints

Register Activity

Example request

```
POST https://services.humanbrainproject.eu/stream/v0/api/activity/

Authorization:Bearer HERE_COME_THE_USER_TOKEN
Accept: application/json
Content-Type: application/json

{
    "summary": "{{actor}} uploaded DocumentB to CollabC",
    "actor": {"type": "HBPUser", "id": "240343"},
    "verb": "UPLOAD",
    "object": {"type": "HBPDocument", "id": "uuid_here"},
    "target": {"type": "HBPCollab", "id": "123"},
    "time": "2016-05-23T15:39:15.048284Z"
}
```

Example response

```
HTTP/1.1 201 Created
```

Retrieve Activity Stream

Example request

```
GET https://services.humanbrainproject.eu/stream/v0/api/stream/HBPUser:240243/

Authorization:Bearer HERE_COME_THE_USER_TOKEN
Accept: application/json

:query page_size (optional): page size, default is 100
:query from (optional): get only activities after given date (format YYYY-MM-DD)
:query to (optional): get only activities before given date (format YYYY-MM-DD)
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
```

```
        {
            "actor": {
                "id": "240243",
                "state": null,
                "type": "HBPUser"
            },
            "object": {
                "id": "226241",
                "state": null,
                "type": "HBPUser"
            },
            "references": {},
            "summary": "240243 added new member `226241` to collab `yury testtt`",
            "target": {
                "id": "2003",
                "state": null,
                "type": "HBPCollab"
            },
            "time": "2016-05-23T15:39:15.048284Z",
            "verb": "ADD"
        },
        {
            "actor": {
                "id": "240243",
                "state": null,
                "type": "HBPUser"
            },
            "object": {
                "id": "123",
                "state": null,
                "type": "HBPUser"
            },
            "references": {
                "actor": {
                    "indices": [
                        0,
                        7
                    ]
                },
            },
            "summary": "@brukau uploaded DocumentB to CollabC",
            "target": {
                "id": "1",
                "state": null,
                "type": "HBPCollab"
            },
            "time": "2016-05-19T13:32:20.875589Z",
            "verb": "ADD"
        }
    ]
}
```


Activity Stream Heatmap

Return heatmap for last 90 days by default

Example request

```
GET https://services.humanbrainproject.eu/stream/v0/api/heatmap/HBPUser:240243/?days=5

Authorization:Bearer HERE_COME_THE_USER_TOKEN
Accept: application/json

:query days(optional): number of days for the heatmap, default is 90
```

Example response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "details": [
        {
            "date": "2016-07-15",
            "value": 4
        },
        {
            "date": "2016-07-16",
            "value": 1
        },
        {
            "date": "2016-07-17",
            "value": 2
        },
        {
            "date": "2016-07-18",
            "value": 0
        },
        {
            "date": "2016-07-19",
            "value": 3
        }
    ],
    "total": 10
}
```

Getting Started

As a start, we will train the HBP Notification API. This API lets an HBP Application send their own notifications to a specific user.

In order to use it, you will need a valid **OIDC client**, your **user token** and your **user id**.

User Token

As a first, we generate a user token which has the ability to send notifications to the user.

```
POST https://stream.humanbrainproject.eu/token
```

Django Template

There is a template that will help you bootstrap a Collaboratory Application based on the Django framework.

Usage

```
$ pip install cookiecutter
$ cookiecutter git+https://$USER@bbpcode.epfl.ch/code/platform/hbp/hbp-app-django-template
```

This will ask a few question about the awesome new project and generate:

- A Django application compatible with Python 2.6
- A Puppet recipe to deploy the application on BBP infrastructure

Next Step

- at the root of the Django project, run `make dev_install`
- Ask for a gerrit repository
- Ask for a new PostgreSQL or MySQL database instance
- puppet recipe: fill in the blank, encrypt and publish in the puppet repository

OpenID Connect Client

The HBP OpenID Connect Service let you create clients. A client is a recognized HBP entity that a user can authorize to work with its credentials in order to complete some tasks. The amount of power given to the application is defined by a contract, formalized by a list of scopes that an application requires from the user. The user has the choice to accept or refuse the contract.

In the latter case, the application won't be able to authenticate the user.

Manage OpenID Connect Clients

Create a new Client

Todo

Just write it

Manage OpenID Connect Clients

Todo

Just write it

Ask for restricted scopes

Some scopes give you great power on the behalf of the users, so they cannot be added automatically. You have to send a support request to the Collaboratory team in order to see wether this scope makes sense for your application. List of restricted scopes:

- hbp.notification.all
- hbp.accountrequests

Deep Linking

Introduction

An application can share its state with the Collaboratory using the postMessage API. The Collaboratory will update its current URL to contain the application state so that when a user copies and pastes its browser URL, the application can receive the state in its URL and go to the specific view a user wanted to share.

An application that uses the Deep Linking API is more usable because it:

- lets the user share internal state naturally; and
- lets the notification API focus on the internal part of the application.

Raw Access

This is the raw level access, without using any Javascript helper.

Reading the current state

When the application is loading, it should check if a state has been set and refresh the page accordingly.

The state is available in the URL, like the context UUID:

```
https://my-application.com/runURL?ctx=UUID&**ctxstate=myapplicationstatestring
```

The `ctxstate` query parameter contains the current state.

You can also ask for the full context using the postMessage API.

```
window.parent.postMessage({
  eventName: 'workspace.context',
  ticket: 111
}, 'https://collab.humanbrainproject.eu/');

window.addEventListener('message', function(event) {
  console.log(event);
  // Print the message if it is the answer to the context query.
```

```
  var msg = event.data;
  if (!msg || msg.origin !== 111) {
    // another answer
    return;
  }

  if (msg.eventName === 'error') {
    // unexpected error
    console.error('Cannot retrieve context', msg);
    return;
  }

  // Manage event response

  console.log('Current Context is:', msg.data.ctx);
  console.log('Current Mode is:', msg.data.mode);
  console.log('Current State is:', msg.data.state);
}, false)
```

Send the current state

Every time the application state changes, a signal should be sent to the HBP Collaboratory in order for the URL to be updated and, as a result, to enable the deep linking feature.

You can post the following message to the parent window in order to achieve that.

```
window.parent.postMessage({
  eventName: 'workspace.context',
  ticket: 112,
  data: {
    state: 'MyStateAsAPlainString like' + JSON.stringify([1, 2])
  }
}, 'https://collab.humanbrainproject.eu/');

Please note that the mode and context attributes are ignored. Once you run this
code, the browser URL will be replaced by a new one, including a state
parameter with your string.
```

Security

Your application MUST support corrupted messages and even malicious ones. The state is basically a user defined string passed to your application. It might be invalid for example because of the partial copy/paste of a link. In this case, the intended behaviour is to render the default view (when the state is empty) rather than to display an error message.

Moreover this state is read and stored by a cache router so it should not contain any sensitive information or information that can render differently over time.

We discourage use of the app's URL without careful verification, otherwise a malicious user might create a link which can point to another site location, outside of your control. Storing a paremater value or a query string is preferred to the full URL for this reason.

Finally, never ever evaluate an expression in the state as Javascript code (JSON is OK).

Javascript Client

A Javascript client which wraps the code described in the previous section is available, but the API is unstable at this point. Follow the discussion on Github:

https://github.com/HumanBrainProject/hbp-collaboratory-app-toolkit

angular-hbp-collaboratory

An AngularJS tool to develop web application based on the HBP Collaboratory

Contents

Module: `clb-app`

Local Navigation

- *Children*
- *Description*

Children

Namespace: `clbApp`

Member Of *Module: clb-app*

Local Navigation

- *Children*
- *Description*
  - *Usage*
- *Function:* `emit`
- *Function:* `context`
- *Function:* `open`
- *Typedef:* `HbpCollaboratoryContext`
  - *Properties*
- *Examples*

Children

Description

An AngularJS service to interface a web application with the HBP Collaboratory. This library provides a few helper to work within the Collaboratory environment.

Usage

- *Function: context* is used to set and retrieve the current context.
- *Function: emit* is used to send a command to the HBP Collaboratory and wait for its answer.

Function: `emit`

Send a message to the HBP Collaboratory.

`emit`(*name*, *data*)

Arguments

- **`name`** (`string`) – name of the event to be propagated
- **`data`** (`object`) – corresponding data to be sent alongside the event

Return Promise resolve with the message response

Function: `context`

Asynchronously retrieve the current HBP Collaboratory Context, including the mode, the ctx UUID and the application state if any.

`context`(*data*)

Arguments

- **`data`** (`object`) – new values to send to HBP Collaboratory frontend

Return Promise resolve to the context

Function: `open`

Open a resource described by the given ObjectReference.

The promise will fulfill only if the navigation is possible. Otherwise, an error will be returned.

`open`(*ref*)

Arguments

- **`ref`** (`ObjectReference`) – The object reference to navigate to

Return Promise The promise retrieved by the call to emit

Typedef: `HbpCollaboratoryContext`

Properties

- `string mode`: the current mode, either 'run' or 'edit'
- `string ctx`: the UUID of the current context
- `string state`: an application defined state string

Examples

Listing 3.33: Retrieve the current context object

```
clbApp.context()
.then(function(context) {
  console.log(context.ctx, context.state, context.collab);
})
.catch(function(err) {
  // Cannot set the state
});
```

Listing 3.34: Set the current state in order for a user to be able to copy-paste its current URL and reopen the same collab with your app loaded at the same place.

```
clbApp.context({state: 'lorem ipsum'})
.then(function(context) {
  console.log(context.ctx, context.state, context.collab);
})
.catch(function(err) {
  // Cannot set the state
});
```

Description

`clb-app` module provides utilities to retrieve current HBP Collaboratory Context in an app and to communicate with the current Collaboratory instance.

This module must be bootstraped using `angular.clbBootstrap` function as it needs to load the global environment loaded in CLB_ENBIRONMENT angular constant.

Module: `clb-automator`

Local Navigation

- *Children*
- *Description*

Children

Namespace: Tasks

Member Of *Module: clb-automator*

Local Navigation

Children

Description

Document a list of available tasks.

Function: createCollab

Create a collab defined by the given options.

createCollab(*descriptor*, *descriptor.name*, *descriptor.description*[, *descriptor.privacy*][, *after*])

Arguments

- **descriptor** (object) – Parameters to create the collab
- **descriptor.name** (string) – Name of the collab
- **descriptor.description** (string) – Description in less than 140 characters of the collab
- **descriptor.privacy** (string) – 'private' or 'public'. Notes that only HBP Members can create private collab
- **after** (Array) – descriptor of subtasks

Return Promise

- promise of a collab

Function: createNavItem

Create a new nav item.

createNavItem(*descriptor*, *descriptor.name*, *descriptor.collabId*, *descriptor.app*[, *context*][, *context.collab*])

Arguments

- **descriptor** (object) – a descriptor description
- **descriptor.name** (string) – name of the nav item
- **descriptor.collabId** (Collab) – collab in which to add the item in.
- **descriptor.app** (string) – app name linked to the nav item
- **context** (object) – the current run context
- **context.collab** (object) – a collab instance created previously

Return Promise promise of a NavItem instance

Function: overview

Set the content of the overview page. If an 'entity' is specified, it will use the content of that storage file If an 'app' name is specified, it will use that app for the overview page

The collab is indicated either by an id in *descriptor.collab* or a collab object in *context.collab*.

overview(*descriptor*[, *descriptor.collab*][, *descriptor.entity*][, *descriptor.app*], *context*[, *context.collab*][, *context.entities*])

Arguments

- **descriptor** (object) – the task configuration
- **descriptor.collab** (object) – id of the collab
- **descriptor.entity** (string) – either a label that can be found in context.entities or a FileEntity UUID
- **descriptor.app** (string) – the name of an application
- **context** (object) – the current task context
- **context.collab** (object) – the collab in which entities will be copied
- **context.entities** (object) – a list of entities to lookup in for descriptor.entiry value

Return object created entities where keys are the same as provided in config.storage

Function: storage

Copy files and folders to the destination collab storage.

storage(*descriptor*, *descriptor.storage*[, *descriptor.collab*], *context*[, *context.collab*])

Arguments

- **descriptor** (object) – the task configuration

- **descriptor.storage** (object) – a object where keys are the file path in the new collab and value are the UUID of the entity to copy at this path.
- **descriptor.collab** (object) – id of the collab
- **context** (object) – the current task context
- **context.collab** (object) – the collab in which entities will be copied

Return object created entities where keys are the same as provided in config.storage

Namespace: clbAutomator

Member Of *Module: clb-automator*

Local Navigation

Children

Class: Task

Member Of *Namespace: clbAutomator*

Local Navigation

Children

Description

Instantiate a task given the given *config*. The task can then be run using the *run()* instance method.

Function: `run`

Launch the task.

`run`(*context*)

Arguments

- **`context`** (`object`) – current context will be merged into the default one.

Return Promise promise to return the result of the task

Function: `runSubtasks`

Run all subtasks of the this tasks.

`runSubtasks`(*context*)

Arguments

- **`context`** (`object`) – the current context

Return Array all the results in an array

Description

clbAutomator is an AngularJS factory that provide task automation to accomplish a sequence of common operation in Collaboratory.

How to add new tasks

New tasks can be added by calling `clbAutomator.registerHandler`.

You can see a few example of tasks in the *tasks* folder.

Evaluate the automator

From the root of this project, you can start a server that will let you write a descriptor and run it.

```
gulp example
```

Function: `task`

Instantiate a new Task intance that will run the code describe for a handlers with the give `name`.

The descriptor is passed to the task and parametrize it. The task context is computed at the time the task is ran. A default context can be given at load time and it will be fed with the result of each parent (but not sibling) tasks as well.

`task`(*name[, descriptor][, descriptor.after][, context]*)

Arguments

- **`name`** (`string`) – the name of the task to instantiate
- **`descriptor`** (`object`) – a configuration object that will determine which task to run and in which order
- **`descriptor.after`** (`object`) – an array of task to run after this one
- **`context`** (`object`) – a default context to run the task with

Return Task

- the new task instance

Function: `run`

Directly generate tasks from given description and run them.

`run`(*descriptor*[, *context*])

Arguments

- **`descriptor`** (`object`) – description of the tasks to run
- **`context`** (`object`) – the initial context

Return Promise promise of the top level task result

Function: `createSubtasks`

Create an array of tasks given an array containing object where the key is the task name to run and the value is the descriptor parameter.

`createSubtasks`(*after*)

Arguments

- **`after`** (`object`) – the content of `descriptor.after`

Return Array/Task array of subtasks

Function: `missingDataError`

Return a HbpError when a parameter is missing.

`missingDataError`(*key*, *config*)

Arguments

- **key** (`string`) – name of the key
- **config** (`object`) – the invalid configuration object

Return HbpError a HbpError instance

Function: ensureParameters

Ensure that all parameters listed after config are presents.

ensureParameters(*config*)

Arguments

- **config** (`object`) – task descriptor

Return object created entities

Function: extractAttributes

Return an object that only contains attributes from the *attrs* list.

extractAttributes(*config*, *attrs*)

Arguments

- **config** (`object`) – key-value store
- **attrs** (`Array`) – a list of keys to extract from *config*

Return object key-value store containing only keys from attrs found in *config*

Examples

Listing 3.35: Create a Collab with a few navigation items

```
// Create a Collab with a few navigation items.
angular.module('MyModule', ['clb-automator'])
.run(function(clbAutomator, $log) {
  var config = {
    title: 'My Custom Collab',
    content: 'My Collab Content',
    private: false
  };
  clbAutomator.task(config).run().then(function(collab) {
      $log.info('Created Collab', collab);
  });
})
```

Listing 3.36: Create a Collab with entities and navigation items

```
clbAutomator.run({
  "collab": {
    "title": "Test Collab Creation",
```

```
    "content": "My Collab Description",
    "private": true,
    "after": [
      {
        "storage": {
          "entities": {
            // Use one of your file UUID here.
            "sample.ipynb": "155c1bcc-ee9c-43e2-8190-50c66befa1fa"
          },
          "after": [{
            "nav": {
              "name": "Example Code",
              "app": "Jupyter Notebook",
              "entity": "sample.ipynb"
            }
          }]
        }
      },
      {
        "nav": {
          "name": "Empty Notebook",
          "app": "Jupyter Notebook"
        }
      },
      {
        "nav": {
          "name": "Introduction",
          "app": "Rich Text Editor"
        }
      }
    ]
  }
}).then(function(collab) {
  $log.info('Created Collab', collab);
});
```

Listing 3.37: Create a Collab with a pre-filled overview

```
clbAutomator.run({
  "collab": {
    "title": "Test Collab With Pre Filled Overview",
    "content": "Test collab creation with  a pre filled overview",
    "private": true,
    "after": [{
      "overview": {
        // Use one of your HTML file UUID here.
        "entity": "155c1bcc-ee9c-43e2-8190-50c66befa1fa"
      }
    }]
  }
}).then(function(collab) {
  $log.info('Created Collab', collab);
});
```

Description

clb-automator module provides an automation library for the Collaboratory using the AngularJS service *clbAutomator*. It supports object describing a serie of actions that have to be run either concurrently or sequentially.

It is used for example to script the creation of new custom collab in the *Create New Collab* functionality in *collaboratory-extension-core*.

Module: `clb-collab`

Local Navigation

- *Children*
- *Description*
- *Function:* `ClbCollabModel`
- *Function:* `ClbContextModel`

Children

Class: `App`

Local Navigation

- *Children*
- *Description*

Children

Description

client representation of an application

Function: `toJson`

Transform an App instance into an object reprensentation compatible with the backend schema. This object can then be easily converted to a JSON string.

`toJson`()

Return object server representation of an App instance

Function: `App.fromJson`

Create an app instance from a server representation.

App.**`fromJson`**(*json*)

Arguments

- **`json`** (`object`) – converted from the server JSON string

Return App the new App instance

Namespace: `clbCollabApp`

Local Navigation

Children

Description

clbCollabApp can be used to find and work with the registered HBP Collaboratory applications.

Function: `list`

`list`()

Return Promise promise of the list of all applications

Function: `getById`

Retrieve an App instance from its id.

`getById`(*id*)

Arguments

- **`id`** (`number`) – the app id

Return Promise promise of an app instance

Function: `findOne`

`findOne`(*params*)

Arguments

- **`params`** (`object`) – query parameters

Return Promise promise of an App instance

Namespace: `clbCollabNav`

Local Navigation

Children

Class: `NavItem`

Member Of *Namespace: clbCollabNav*

Local Navigation

Children

Description

Client representation of a navigation item.

Function: `toJson`

Return a server object representation that can be easily serialized to JSON and send to the backend.

`toJson`()

Return object server object representation

Function: `update`

`update`(*attrs*)

Arguments

- **`attrs`** (`object`) – NavItem instance attributes

Return NavItemt this instance

Function: `ensureCached`

`ensureCached`()

Return NavItem this instance

Function: `NavItem.fromJson`

Build an instance from the server object representation.

`NavItem.`**`fromJson`**(*collabId*, *json*)

Arguments

- **`collabId`** (`number`) – collab ID
- **`json`** (`string`) – server object representation

Return NavItem new instance of NavItem

Description

clbCollabNav provides tools to create and manage navigation items.

Function: `getRoot`

Retrieve the root item of the given collab.

`getRoot`(*collabId*)

Arguments

- **`collabId`** (`number`) – collab ID

Return Promise promise the root nav item

Function: `getNode`

`getNode`(*collabId*, *nodeId*)

Arguments

- **`collabId`** (`number`) – collab ID
- **`nodeId`** (`number`) – node ID

Return NavItem the matching nav item

Function: `getNodeFromContext`

`getNodeFromContext`(*ctx*)

Arguments

- **`ctx`** (`str`) – The context UUID

Return Promise The promise of a NavItem

Function: addNode

addNode(*collabId*, *navItem*)

Arguments

- **collabId** (number) – collab ID
- **navItem** (number) – the NavItem instance to add to the navigation

Return Promise promise of the added NavItem instance

Function: deleteNode

deleteNode(*collabId*, *navItem*)

Arguments

- **collabId** (number) – collab ID
- **navItem** (NavItem) – the NavItem instance to remove from the navigation

Return Promise promise of an undefined item at the end

Function: update

update(*collabId*, *navItem*)

Arguments

- **collabId** (number) – collab ID
- **navItem** (NavItem) – the instance to update

Return Promise promise the updated instance

Function: insertNode

Insert node in the three.

A queue is used to ensure that the insert operation does not conflict on a single client.

insertNode(*collabId*, *navItem*, *parentItem*, *insertAt*)

Arguments

- **collabId** (int) – id of the collab
- **navItem** (NavItem) – Nav item instance
- **parentItem** (NavItem) – parent item
- **insertAt** (int) – add to the menu

Return Promise a promise that will return the update nav item

Namespace: `clbCollabTeamRole`

Local Navigation

- *Children*
- *Description*

Children

Description

Namespace: `clbCollabTeam`

Local Navigation

- *Children*
- *Description*

Children

Description

Angular client to access Collab Team REST endpoint.

Namespace: `clbCollab`

Member Of *Module: clb-collab*

Local Navigation

- *Children*
- *Description*

Children

Description

Provide a javascript client to query the Collab REST service.

Namespace: `clbContext`

Member Of *Module: clb-collab*

Local Navigation

Children

Description

Function: `get`

`get`(*uuid*)

Arguments

- **`uuid`** (`string`) – UUID of the context

Return Promise Resolve to the ClbContextModel instance

Description

Contain services to interact with collabs (e.g.: retriving collab informations or team members).

Function: `ClbCollabModel`

Representation of a Collab.

`ClbCollabModel`([*attributes*])

Arguments

- **`attributes`** (`object`) – initial values

Function: `ClbContextModel`

Representation of a Collab Context.

`ClbContextModel`()

Module: `clb-context-data`

Local Navigation

Children

Description

Provides a key value store where keys are context UUID and values are string.

Module: `clb-env`

Local Navigation

Children

Description

`clb-env` module provides a way to information from the global environment.

Function: `clbEnv`

Get environement information using dotted notation with the *clbEnv* provider or service.

Before being used, clbEnv must be initialized with the context values. You can do so by setting up a global bbpConfig variable or using angular.clbBootstrap.

`clbEnv`(*$injector*)

Arguments

- **`$injector`** (`object`) – AngularJS injection

Return object provider

Module: `lodash`

Local Navigation

- *Children*
- *Description*

Children

Description

Fix some compatibility issues with previous angular-hbp-common.

Module: `hbpCollaboratoryCore`

Local Navigation

- *Children*
- *Description*

Children

Module: `hbpCollaboratoryUI`

Member Of *Module: hbpCollaboratoryCore*

Local Navigation

- *Children*
- *Description*
- *Typedef:* `UUID`

Children

Namespace: `hbpCollaboratory`

Member Of *Module: hbpCollaboratoryUI*

Local Navigation

- *Children*
- *Description*

Children

Description

`hbpCollaboratory` module is a shell around various AngularJS modules that interface with the HBP Collaboratory. It loads both the core modules and the UI modules, as well as the backward compatibility modules.

Description

Module to load the UI part of angular-hbp-collaboratory. Try to use the sub-modules instead.

Typedef: `UUID`

A string formatted as a valid UUID4

Description

Module to load all the core modules. Try to use the sub-modules instead.

Module: `clb-rest`

Local Navigation

- *Children*
- *Description*

Children

Namespace: `clbResultSet`

Local Navigation

- *Children*
- *Description*

Children

Class: `ResultSet`

Member Of *Namespace: clbResultSet*

Local Navigation

Children

Description

Build a result set with internal support for fetching next and previous results. Member: `next`: Retrieve the next result page.

Description

Member: `ResultSetEOL`: error thrown when module:clb-rest.ResultSet is crawled when at an extremity.

Member: `get`: Return a promise that will resolve once the result set first page is loaded.

The promise contains the *instance* of the result set as well.

Description

`clb-rest` module contains util for simplifying access to Rest service.

Module: `clb-storage`

Local Navigation

Children

Namespace: `clbStorage`

Member Of *Module: clb-storage*

Local Navigation

Children

Description

The `clbStorage` service provides utility functions to ease the interaction of apps with storage.

Function: `getEntity`

Get an entity (e.g.: a project, a file or a folder) using a locator. The only accepted locator at this time is the entity UUID.

- the entity UUID
- an entity representation with `{_uuid: ENTITY_UUID}`

- the entity related context `{ctx: CONTEXT_UUID}`
- the entity collab ID `{collab: COLLAB_ID}`
- the entity absolute path

getEntity(*locator*)

Arguments

- **locator** (any) – Describe the entity to retrieve (see description).

Return Promise Return a module-clb-storage.EntityDescriptor when fulfilled or reject a module-clb-error.ClbError

Function: getAbsolutePath

Return the absolute path of the entity

getAbsolutePath(*entity*)

Arguments

- **entity** (object|UUID) – UUID or descriptor

Return Promise return a path string when fulfilled.

Function: runOnce

Ensure there is only one async *fn* run named *k* at once. subsequent call to runOnce with the same *k* value will return the promise of the running async function.

runOnce(*k*, *fn*)

Arguments

- **k** (string) – The key
- **fn** (function) – The function that retrieve a Promise

Return Promise Resolve to the function result

Function: getEntityByUUID

getEntityByUUID(*uuid*)

Arguments

- **uuid** (string) – Entity UUID

Return Promise Resolve to the entity Descriptor

Function: query

Query entities by attributes or metadata.

query(*params*)

Arguments

- **params** (object) – Query Parameters

Return Promise Return the results

Function: metadataKey

Retrieve the key to lookup for on entities given the ctx

metadataKey(*ctx*)

Arguments

- **ctx** (string) – application context UUID

Return string name of the entity attribute that should be used

Function: addMetadata

Add metadata to the provided entity and returns a promise that resolves to an object containing all the new metadata. The promise fails if one of the metadata already exists.

addMetadata(*entity*, *metadata*)

Arguments

- **entity** (object) – Entity Descriptor
- **metadata** (object) – key/value store where keys are the metadata name to set

Return Promise Resolves after the operation is completed

Function: deleteMetadata

Delete metadata keys in input from the provided entity and returns a promise that resolves to an object containing all the remaining metadata. The promise fails if one of the metadata doesn't exist.

deleteMetadata(*entity*, *metadataKeys*)

Arguments

- **entity** (object) – Entity Descriptor
- **metadataKeys** (array) – Array of metatdata keys to delete

Return Promise Resolve to the metadata

Function: create

Create a new entity.

create(*type*, *parent*, *name*, *options*)

Arguments

- **type** (string) – Entity Type (e.g.: file, folder, project)
- **parent** (string|object) – Parent UUID or entity descriptor
- **name** (string) – File name
- **options** (object) – Extend the entity descriptor with those data

Return Promise Resolve once done

Function: copy

Copy a file to a destination folder

copy(*srcId*, *destFolderId*)

Arguments

- **srcId** (string) – UUID of the entity to copy
- **destFolderId** (string) – UUID of the target directory

Return Promise Resolves when done

Function: getContent

Retrieves the content of a file given its id.

getContent(*id*[, *customConfig*])

Arguments

- **id** (string) – FileEntity UUID
- **customConfig** (object) – contains extra configuration

Return Promise The raw content

Function: getUserAccess

Get current user access right to the provided entity.

The returned promise will be resolved with an object literal containing three boolean flags corresponding the user access:

- canRead
- canWrite
- canManage

getUserAccess(*entity*)

:param module:clb-storage.EntityDescriptor entity: The entity to retrieve user access for
:return object: Contains {boolean} canRead, {boolean} canWrite, {boolean} canManage

Function: `getChildren`

Retrieve children entities of a 'parent' entity according to the options and add them to the children list. The returned promise will be resolved with the list of fetched children and a flag indicating if more results are available in the queried direction.

`getChildren`(*parent[, options][, options.accept][, options.acceptLink][, options.sort][, options.filter][, options.until][, options.from][, options.pageSize]*)

:param module:clb-storage.EntityDescriptor parent: The parent entity :param object options: Options to make the query :param array/string options.accept: Array of accepted _entityType :param boolean|array/string options.acceptLink: `true` or an array of accepted linked _entityType :param string options.sort: Property to sort on :param string options.filter: The result based on Acls. Values: `read` (default), `write` :param UUID options.until: Fetch results until the given id (exclusive with from) :param UUID options.from: Fetch results from the given id (exclusive with until) :param int options.pageSize: The number of results per page. Default is provided by the service. Set to 0 to fetch all the records. :return Promise: When fulfilled, return a paginated result set. You can also access it immediately using `promise.instance`

Function: `upload`

Create file entity and upload the content of the given file.

options should contain a *parent* key containing the parent entity.

Possible error causes:

- FileTooBig
- UploadError - generic error for content upload requests
- EntityCreationError - generic error for entity creation requests
- FileAlreadyExistError

`upload`(*file*, *options*)

Arguments

- **`file`** (`File`) – The file descriptor to upload
- **`options`** (`Object`) – The list of options

Return Promise a Promise that notify about progress and resolve with the new entity object.

Function: `downloadUrl`

Asynchronously ask for a short-lived (a few seconds), presigned URL that can be used to access and download a file without authentication headers.

`downloadUrl`(*entity*)

:param module:clb-storage.EntityDescriptor entity: The file to download :return Promise: Return a string containing the URL once the Promise

is fulfilled.

Member: `setContextMetadata`: the function links the contextId with the doc browser entity in input by setting a specific metadata on the entity.

Entity object in input must contain the following properties: - _entityType - _uuid

In case of error, the promise is rejected with a *HbpError* instance. Member: `deleteContextMetadata`: the function unlink the contextId from the entity in input by deleting the context metadata.

Entity object in input must contain the following properties: - _entityType - _uuid

In case of error, the promise is rejected with a *HbpError* instance. Member: `updateContextMetadata`: the function delete the contextId from the *oldEntity* metadata and add it as *newEntity* metadata.

Entity objects in input must contain the following properties: - _entityType - _uuid

In case of error, the promise is rejected with a *HbpError* instance.

Description

The `clb-storage` module contains tools needed to access and work with the HBP Document Service. It is targeted to integrate easily with the HBP Collaboratory, even if the service is more generic.

Typedef: `EntityDescriptor`

Describe an arbitrary entity in the storage stytem. The principal types are

- *file*: the entity is a file whose content can be retrieved
- *folder*: the entity is a folder and can be the parent of other entities
- *project*: First level folder. It behave like a folder but also defines the ACL for all the children

Properties

- `UUID _uuid`: The entity UUID
- `string _entityType`: The entity type (e.g.: `file`, `folder`, `project`)

Module: `clb-stream`

Local Navigation

Children

Namespace: clbStream

Member Of *Module: clb-stream*

Local Navigation

Children

Description

clbStream service is used to retrieve feed of activities given a user, a collab or a specific context.

Function: buildURLOptions

Builds the URL options such as the from and to date as well as the page_size

buildURLOptions(*url*, *options*)

Arguments

- **url** (string) – original url
- **options** (object) – pageSize:15, date:'2016-07-20'

Return string Built URL

Function: getStream

Get a feed of activities regarding an item type and id.

getStream(*type*, *id*, *options*)

Arguments

- **type** (string) – The type of object to get the feed for
- **id** (string|int) – The id of the object to get the feed for
- **options** (object) – Parameters to pass to the query

Return Promise resolve to the feed of activities

Member: activityListFactoryFunc: Return activities

Description

The *clb-stream* module contains a service and a few directives to retrieve and display the HBP Collaboratory stream provided by the various applications.

Function: `registerUrlHandler`

Add a function that can generate URL for some types of object reference.

The function should return a string representing the URL. Any other response means that the handler is not able to generate a proper URL for this type of object.

The function signature is `function(objectReference) { return 'url' // or nothing}`

`registerUrlHandler`(*handler*)

Arguments

- **`handler`** (`function`) – a function that can generate URL string for some objects

Return provider The provider, for chaining.

Member: `clbResourceLocator`: resourceLocator service

Module: `clb-ui-dialog`

Local Navigation

- *Children*
- *Description*

Children

Namespace: `clbDialog`

Local Navigation

- *Children*
- *Description*

Children

Description

Service to trigger modal dialog.

Description

Module: clb-ui-error

Local Navigation

- *Children*
- *Description*

Children

Namespace: clbErrorMessage

Local Navigation

- *Children*
- *Description*
- *Examples*

Children

Description

The clb-error-message directive displays an error.

clb-error is a HbpError instance, built by the HbpErrorService

Examples

Listing 3.38: Retrieve the current context object

```
<div ng-controller='SomeController'>
  Validation error:
  <clb-error-message clb-error='error'></clb-error-message>
  Permission denied error:
  <clb-error-message clb-error='errorPermissions'></clb-error-message>
</div>
```

Description

Member: clbError: The factory clbUiError instantiates modal error dialogs. Notify the user about the given error.

Module: `clb-ui-form`

Local Navigation

Children

Namespace: `clbFormControlFocus`

Local Navigation

Children

Description

The `clbFormControlFocus` Directive mark a form element as the one that should receive the focus first.

Examples

Listing 3.39: Give the focus to the search field

```
angular.module('exampleApp', ['clb-ui-form']);

// HTML snippet:
// <form ng-app="exampleApp"><input type="search" clb-ui-form-control-focus></form>
```

Namespace: `clbFormGroupState`

Local Navigation

Children

Description

`clbFormGroupState` directive flag the current form group with the class has-error or has-success depending on its form field current state.

Examples

Listing 3.40: Track a field validity at the `.form-group` level

```
angular.module('exampleApp', ['hbpCollaboratory']);
```

Description

clb-ui-form provides directive to ease creation of forms.

Module: `clb-ui-identity`

Local Navigation

Children

Namespace: `clbUserAvatar`

Member Of *Module: clb-ui-identity*

Local Navigation

Children

Description

Display an square icon for a user.

Attributes

Name	Description
clb-user	The ClbUser instance to display

Examples

Listing 3.41: Display user avatar

```
<clb-user-avatar clb-user="vm.currentUser"></clb-user-avatar>
```

Namespace: `clbUsercardPopoverDirective`

Member Of *Module: clb-ui-identity*

Local Navigation

- *Children*
- *Description*

Children

Description

Display the user summary in a popover element.

Only one of those can be open at any time.

Name Description	
{string|HBPUser} clb-usercard-popover	The ClbUser instance to display or its ID

Namespace: `clbUserCardPopoverService`

Member Of *Module: clb-ui-identity*

Local Navigation

Children

Description

A singleton to manage clb-usercard-popover instances

Description

Provides UI widgets around user and groups.

Function: `clbUsercard`

Display general user informations.

Attributes

Name	Description
clb-user	The ClbUser instance to display
clb-template	URL of a custom template to use

`clbUsercard`(*lodash*)

Arguments

- **`lodash`** (`object`) – Angular DI

Return object Directive Descriptor

Module: `clb-ui-loading`

Local Navigation

Children

Namespace: `clbPerformAction`

Local Navigation

Children

Description

clbPerformAction directive run an action when the given control is clicked. it can be added as an attribute. While the action is running, the control is disabled.

Examples

Listing 3.42: use perform action to disable a button while code is running

```
<div ng-controller="myController">
 <input class="btn btn-primary" type="submit" clb-perform-action="doSomething()">
</div>
```

Description

Provides a simple loading directive.

Function: `clbLoading`

The directive clbLoading displays a simple loading message. If a promise is given, the loading indicator will disappear once it is resolved.

Attributes

Name	Description
{Promise} [clb-promise]	Hide the loading message upon fulfilment.
{string} [clb-message]	Displayed loading string (default="'loading..."')

`clbLoading`()

Return object Angular directive descriptor

Module: `clb-ui-storage`

Local Navigation

Children

Namespace: `clbFileBrowser`

Local Navigation

Children

Namespace: `clbFileBrowserFolder`

Local Navigation

Children

Description

clbFileBrowserFolder directive is a child directive of clbFileBrowser that render a folder item within the file browser view.

Available attributes:

- clb-ui-storage-folder: the folder entity
- [clb-ui-storage-folder-icon]: a class name to display an icon
- [clb-ui-storage-folder-label]: a label name (default to folder._name)

Examples

```
<!-- minimal -->
<div clb-ui-storage-folder="folderEntity"></div>
<!-- all wings out -->
<div clb-ui-storage-folder="folderEntity"
     clb-ui-storage-folder-icon="fa fa-level-up"
     clb-ui-storage-label="up"></div>
```

Namespace: `clbFileBrowserPath`

Local Navigation

Children

Description

clbFileBrowserPath directive is a child of clbFileBrowser directive that renders the breadcrumb according to the file browser setup.

Examples

```
<clb-file-browser-path></clb-file-browser-path>
```

Namespace: `FileBrowserViewModel`

Member Of *Namespace: clbFileBrowser*

Local Navigation

- *Function:* `handleFocus`
- *Function:* `handleNavigation`
- *Function:* `loadMoreFiles`
- *Function:* `loadMoreFolders`

Children

Description

ViewModel of the clbFileBrowser directive. This instance is accessible by all direct children of the file browser.

It is responsible to handle all the interactions between the user and the services. It does not update the views directly but sends the relevant events when necessary.

Function: `handleFocus`

When the user focus on a browser item, emit a 'clbFileBrowser:focusChanged' event.

The event signature is (event, newEntity, previousEntity).

handleFocus(*entity*)

Arguments

- **entity** (`Object`) – selected entity

Function: `handleNavigation`

When the current context change, trigger a navigation update.

This will render the view for the new current entity. All navigations are chained to ensure that the future view will end in a consistant state. As multiple requests are needed to render a view, request result would sometimes finish after a new navigation event already occured.

handleNavigation(*entity*)

Arguments

- **entity** (`Object`) – the new current entity

Return promise resolve when the navigation is done.

Function: `loadMoreFiles`

Load the next page of file entities for the current entity.

loadMoreFiles()

Return Promise resolve when the files are loaded

Function: `loadMoreFolders`

Load the next page of folder entities for the current entity.

`loadMoreFolders`()

Return Promise resolve when the folders are loaded

Description

clbFileBrowser Directive

This directive renders a file browser. It handles creation of folder, mutliple file uploads and selection of entity. Focus selection change can be detected by listening to the event `clbFileBrowser:focusChanged`.

Attributes

Parameter	Description
`{EntityDescriptor} [clb-root]`	A project or a folder that will be the root of the tree.
`{EntityDescriptor} [clb-entity]`	The selected entity.

Events

clbFileBrowser:focusChanged	Emitted when the user focus a new file or folder
clbFileBrowser:startCreateFolder	Emitted when the user start to create a new folder

Examples

Listing 3.43: Simple directive usage

```
<clb-file-browser clb-root="someProjectEntity"
                  clb-entity="someSubFolderEntity">
</clb-file-browser>
```

Namespace: `clbFileChooser`

Local Navigation

- *Children*
- *Description*

Children

Description

The `clbFileChooser` directive let you browse the storage to pick a file.

Name	Description
[clb-root]	Cannot go beyond this ancestor in the browser
[ng-model]	The ngModel to bind to the chosen value
[clb-validate]	a string, array of string, regex or function (can be async)

The directive emit the following events:

Name	Description
clbFileChooser:fileSelected	The second parameter is the EntityDescriptor
clbFileChooser:cancel	The second parameter is the initial EntityDescriptor

Namespace: `clbFileUpload`

Local Navigation

Children

Description

clbFileUpload directive.

Provide an upload widget where user can stack files that should be uploaded at some point. The directive doesn't proceed to upload by itself but rather triggers the onDrop callback.

The directive accepts the following attributes:

- on-drop: a function to call when one or more files are dropped or selected the callback will receive an array of File instance.
- on-error: a function to call when an error occurs. It receives an HbpError instance in parameter.

Examples

```
<clb-file-upload on-drop="handleFileUpload(files)"
                      on-error="handleError(error)">
</clb-file-upload>
```

Description

The `clb-ui-storage` module provides Angular directive to work with the HBP Collaboratory storage.

Featured Component

- The directive clb-file-browser provides an easy to use browser which let the user upload new files, create folder and act as file selector.

Module: `clb-ui-stream`

Local Navigation

Children

Description

Member: `clbActivity`: `clb-activity` directive is displays an activity retrieved by the HBP Stream service in a common way.

It try to look up for a detailled description of the event and fallback to the summary if he cannot. Member: `clbFeed`: `clb-feed` directive displays a feed of activity retrieved by the HBP Stream service. It handles scrolling and loading of activities. Each activity is rendered using the `clb-activity` directive.

Namespace: `angular`

Local Navigation

Children

Description

Function: `clbBootstrap`

Bootstrap AngularJS application with the HBP environment loaded.

It is very important to load the HBP environement *before* starting the application. This method let you do that synchronously or asynchronously. Whichever method you choose, the values in your environment should look very similar to the one in https://collab.humanbrainproject.eu/config.json, customized with your own values.

At least `auth.clientId` should be edited in the config.json file.

`clbBootstrap`(*module*, *options*, *options.env*)

Arguments

- **`module`** (`string`) – the name of the Angular application module to load.
- **`options`** (`object`) – pass those options to deferredBootstrap
- **`options.env`** (`object`) – HBP environment JSON (https://collab.humanbrainproject.eu/config.json)

Return Promise return once the environment has been bootstrapped

Member: `clbBootstrap`:

README

---

angular-hbp-collaboratory is a collection of AngularJS module to develop applications for the HBP Collaboratory.

Install Using Bower

```
bower install angular-hbp-collaboratory
```

- *angular-hbp-collaboratory.js* provides all the needed AngularJS module
- *angular-hbp-collaboratory.css* provides styles for the visual components

Alternatively you can rely on *src/angular-hbp-collaboratory/main.scss* if you plan to use Sass in your project.

Contributing

Dependencies:

This project use NodeJS and NPM to test, lint and generating the final library.

Bower, Gulp and ESLint should be installed globally:

```
npm install -g bower gulp eslint
```

Install:

```
git clone git@github.com:HumanBrainProject/angular-hbp-collaboratory.git
cd angular-hbp-collaboratory
npm install
bower install
```

Install this pre-commit hook to ensure your code is green before a committing:

```
cp .git-pre-commit .git/hooks/pre-commit
```

Running tests on code change:

```
gulp watch
```

Migration from angular-hbp-common

Here is a quick (and incomplete) checklist of refactoring to operate to your project if you want to migrate from angular-hbp-common to this cleaner library:

Service Refactoring

First, rely on the services from angular-hbp-collaboratory. For this, you need to depends on the `hbpCollaboratoryCore` module.

If you use bower to install, it will ask you to resolve a conflict about the angular-bootstrap version. Stick to the angular-hbp-common declaration at this point. At this point, your code should still work, that will let you progressively refactor to use the new library instead of the old one:

```
Add dependency 'hbpCollaboratory'
hbpErrorService -> clbError                    (from clb-error module)
hbpUtil.ferr -> clbError.rejectHttpError       (from clb-error module)
hbpUtil.paginatedResultSet -> clbResultSet.get (from clb-rest module)
hbpIdentityUserDirectory -> clbUser            (from clb-identity module)
hbpCollabStore -> clbCollab                    (from clb-collab module)
hbpCollabStore.context -> clbContext           (from clb-collab module)
hbpCollaboratoryNavStore -> clbCollabNav       (from clb-collab module)
hbpCollaboratoryAppStore -> clbCollabApp       (from clb-collab module)
hbpEntityStore -> clbStorage                   (from clb-storage module)
hbpFileStore -> clbStorage                     (from clb-storage module)
hbpProjectStore -> clbStorage                  (from clb-storage module)
hbpConfigStore -> Manually refactor to clbCtxData (from clb-ctx-data)
  The service now use JSON as data format and the method signature
  changed from method(config) to method(ctx, data)
```

In fact, `hbpCollaboratoryCore` is a shell module that will require many sub-modules as an easy way to migrate and import everything. It would be even better if your application require only the needed sub-modules as indicated by the refactoring list above.

Once the refactoring of module is done, there is the refactoring of methods:

```
clbStorage.getEntityByContext(ctx) -> clbStorage.getEntity({ctx: ctx})
clbStorage.get( -> clbStorage.getEntity(
clbStorage.getChildren now return a ResultSet like other services
```

clbUser.isHbpMember is no more because the accreditation multiply. You should instead make a call like:

clbUser.isHbpMember() -> clbUser.isGroupMember(['hbp-accred-sga1']);

At this point, your javascript code should rely only on `angular-hbp-collaboratory`, with the exception of the UI. Your application should work as previously. If you were not using any directive from the beforementioned module, you are done and you can remove the old module import, as well as their reference in `bower.json`

```
// If there is no UI components in use

// before
angular.module('myModule', [
  'hbpCollaboratoryCore',
  'hbpCommon',
  'bbpDocumentClient'
]);

// after
angular.module('myModule', [ // some of the following:
  'clb-app',
  'clb-automator',
  'clb-collab',
  'clb-env',
  'clb-error',
  'clb-identity',
  'clb-rest',
  'clb-storage',
  'clb-stream'
]);
```

If your code is using some of the directive from angular-hbp-common or angular-hbp-document-client, you need to refactor them as well before being able to cut the old dependencies.

UI Refactoring

UI Bootstrap has been upgraded to the next major version and the components are now prefixed. This means you cannot use the UI part of angular-hbp-common with angular-hbp-collaboratory. At this point, you should entirely remove angular-hbp-common from your dependencies and require the UI package from angular-hbp-collaboratory.

```
// before
angular.module('myModule', [
  'hbpCollaboratoryCore',
  'hbpCommon',
  'bbpDocumentClient'
]);

// after
angular.module('myModule', [
  'hbpCollaboratoryCore',
  'hbpCollaboratoryUI',
]);
```

You now need to run `bower update` and resolve the conflict on `angular-bootsrap` by choosing the version in `angular-hbp-collaboratory`.

If your code is using directives from this library, please refer to the angular-bootstrap Migration Guide <https://github.com/angular-ui/bootstrap/wiki/Migration-guide-for-prefixes>. To find if and where your code is using such directives, you can run the following command in your source code folder:

```
grep -ro '<accordion\|<accordion-group\|<accordion-heading\|<accordionConfig\|<alert\|
 ↪<btn-checkbox\|<btn-radio\|<buttonConfig\|<carousel\|<slide\|<collapse\|<dateParser\|
 ↪<datepicker\|<datepicker-popup\|<daypicker\|<monthpicker\|<yearpicker\|
 ↪<datepickerConfig\|<datepickerPopupConfig\|dropdown=\|dropdown-toggle=\|dropdown-
 ↪menu=\|<keyboard-nav\|<dropdownService\|<$modal\|<$modalInstance\|<$modalStack\|<modal-
 ↪transclude\|<pagination\|<pager\|<pagerConfig\|<paginationConfig\|popover=\|popover-
 ↪template=\|popover-html=\|$position\|<progressbar\|<bar\|<progress\|<progressConfig\|
 ↪<rating\|<ratingConfig\|<tabset\|<tab\|<tab-
 ↪heading\|timepicker\|timepickerConfig\|tooltip=\|tooltip-template=\|tooltip-html=\|
 ↪$tooltip\|typeahead\|typeahead-match\|typeaheadHighlightFilter\|typeaheadParser' .
```

You can also use the directives provided by this package. Please be sure to check the change in the directive attributes prefix as well.:

```
hbp-file-browser -> clb-ui-storage (root -> clb-root, entity -> clb-entity)
hbp-error-message -> clb-error-message (hbp-promise -> clb-promise, hbp-message -> clb-
 ↪message)
hbp-usercard -> clb-usercard (hbp-user -> clb-user, hbp-template -> clb-template)
hbp-loading -> clb-loading (hbp-promise -> clb-promise, hbp-message -> clb-message)
hbp-perform-action -> clb-perform-action
```

If you wrote a usercard custom template (using `hbp-template` attribute), you should update the following css classes and probably update the template to conform to the new html structure:

```
hbp-usercard -> clb-usercard
hbp-usercard-pix -> clb-usercard-pix
hbp-user-avatar -> clb-user-avatar
hbp-usercard-header -> clb-usercard-header
hbp-usercard-institution -> clb-usercard-institution
hbp-usercard-contact -> clb-usercard-contact
hbp-usercard-contact-item -> clb-usercard-contact-item
```

`hbpDialogFactory` has been removed, with the exception of `hbpDialogFactory.error` and `hbpDialogFactory.confirm` which are now respectively `clbErrorDialog.open` (module `clb-ui-error`) and `clbConfirm.open` (module `clb-ui-dialog`). These two refactoring will have you covered:

```
hbpDialogFactory -> clbErrorDialog and/or clbConfirm
hbpDialogFactory.error -> clbErrorDialog.open
hbpDialogFactory.confirm -> clbConfirm.open
```

If you were using other methods from hbpDialogFactory (e.g.: `.alert()`), you need to rewrite them using angular-bootstrap `$uibModal` (read the documentation <https://angular-ui.github.io/bootstrap/#/modal>)

Since usage of `hbp-generated-icon` has been deprecated for anything but users without avatars, it has been replaced by a new directive called `clb-user-avatar` available in the module clb-ui-identity. It displays either a generated icon or the user profile picture. This new component is

also easier to customize using pure css.

At the end of the process, your application should only load `angular-hbp-collaboratory`

```
angular.module('myModule', [
// some of the following:
  'clb-app',
  'clb-automator',
  'clb-collab',
  'clb-env',
  'clb-error',
  'clb-identity',
  'clb-rest',
  'clb-storage',
  'clb-stream',
  'clb-ui-error',
  'clb-ui-storage',
  'clb-ui-form',
  'clb-ui-loading',
  'clb-stream'
]);
```

LICENSE

MIT

Read the project LICENSE file.

HBP Storage

Contents:

Rest APIs

HBP Storage provides project APIs. It is available at the following endpoint: https://services.humanbrainproject.eu/storage/v1/api/.

Entity endpoint

Get entity details

GET https://services.humanbrainproject.eu/storage/v1/api/entity/(uuid: *entity_id*)**/**

Get generic entity by UUID.

Example request:

```
GET /storage/v1/api/entity/e7c582ce-cb64-43ba-a08a-f1e361df71fa/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
```

```
Content-Type: application/json
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "created_by": "226241",
    "created_on": "2017-02-02T14:24:22.785969Z",
    "entity_type": "folder",
    "modified_by": "226241",
    "modified_on": "2017-02-02T14:24:22.786374Z",
    "name": "folder_1",
    "parent": "68d1971d-7293-422e-81aa-06b9de13a461",
    "uuid": "e7c582ce-cb64-43ba-a08a-f1e361df71fa"
}
```

Parameters

- **entity_id** (uuid) – folder id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – folder successfully retrieved
- 400 Bad Request – invalid request. More details in the response. Possible causes are: name already in use, invalid parent type, missing field.
- 403 Forbidden – invalid Oauth2 token provided

Get entity path

GET https://services.humanbrainproject.eu/storage/v1/api/entity/(uuid: *entity_id*)**/path/**

Retrieve entity path.

Example request:

```
GET /storage/v1/api/entity/e7c582ce-cb64-43ba-a08a-f1e361df71fa/path/ HTTP/
↪1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

```
    "path": "/12345/folder_1",
}
```

Parameters

- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – entities successfully retrieved
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Get entity Collab ID

GET https://services.humanbrainproject.eu/storage/v1/api/entity/(uuid: *entity_id*)**/collab/**

Retrieve entity Collab ID.

Example request:

```
GET /storage/v1/api/entity/e7c582ce-cb64-43ba-a08a-f1e361df71fa/collab/␣
 ↪HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "collab_id": 12345,
}
```

Parameters

- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – entities successfully retrieved
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Get entity by query param

GET https://services.humanbrainproject.eu/storage/v1/api/entity/

Retrieve entity by query param which can be either uuid/path/metadata.

Example request:

```
GET /storage/v1/api/entity/?path=/12345/file_1 HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "content_type": "application/json",
    "created_by": "226241",
    "created_on": "2017-02-02T14:27:35.621795Z",
    "entity_type": "file",
    "modified_on": "2017-02-02T14:27:35.621840Z",
    "modified_by": "226241",
    "name": "file_1",
    "parent": "e7c582ce-cb64-43ba-a08a-f1e361df71fa",
    "uuid": "77fb486d-21d5-4a24-a48c-af118b1a23e2"
}
```

Query Parameters

- **uuid** – (optional) uuid=UUID
- **path** – (optional) path=/COLLAB/PATH/TO/FILE
- **any_matadata_key** – (optional) metadata_key=metadata_value. Only one key can be provided; others will be ignored.

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – entity successfully retrieved
- 400 Bad Request – more than one entity matches the query (metadata only)
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity was not found

Project endpoint

Projects are tightly linked to the corresponding Collabs. And Collab permissions propagate to the project permissions.

It is available at the following endpoint: https://services.humanbrainproject.eu/storage/v1/api/project/.

Create a project

Please use Collab Service to create the Collab and allocate HBP Storage project.

List projects

GET https://services.humanbrainproject.eu/storage/v1/api/project/

List all the projects the user have access to.

Example request:

```
GET /storage/v1/api/project/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "collab_id": 1,
            "created_by": "253829",
            "created_on": "2017-01-31T16:04:23.290458Z",
            "entity_type": "project",
            "modified_on": "2017-01-31T16:04:23.290499Z",
            "modified_by": "253829",
            "name": "test1",
            "uuid": "68d1971d-7293-422e-81aa-06b9de13a461"
        }
    ]
}
```

Query Parameters

- **hpc** – (optional) If 'true', the result will contain only the HPC projects (Unicore projects)
- **access** – (optional) if provided, the result will contain only project where the user has Admitted values: ['read', 'write']
- **page_size** – (optional) number of elements per page (default: 100)
- **page** – (optional) number of the page

- **ordering** – (optional) indicate on which fields to sort the result. Prepend '-' to invert order. Multiple values can be provided. Example: 'ordering=name,created_on'. Ordering is supported on: 'name', 'created_on', 'modified_on'
- **name** – (optional) filter on project name
- **collab_id** – (optional) filter on collab_id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – projects successfully retrieved
- 403 Forbidden – invalid Oauth2 token provided

Get project details

GET https://services.humanbrainproject.eu/storage/v1/api/project/(uuid: *project_id*)**/**

Get project info.

Example request:

```
GET /storage/v1/api/project/68d1971d-7293-422e-81aa-06b9de13a461/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "collab_id": 1,
    "created_by": "253829",
    "created_on": "2017-01-31T16:04:23.290458Z",
    "entity_type": "project",
    "modified_on": "2017-01-31T16:04:23.290499Z",
    "modified_by": "253829",
    "name": "test1",
    "uuid": "68d1971d-7293-422e-81aa-06b9de13a461"
}
```

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – projects successfully retrieved
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – project id not found

List project content

GET https://services.humanbrainproject.eu/storage/v1/api/project/(uuid: *project_id*)**/children/**

List all files and folders (not recursively) contained in the project.

Example request:

```
GET /storage/v1/api/project/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/children/␣
↪HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "content_type": "application/json",
            "created_by": "226241",
            "created_on": "2017-02-08T15:31:30.421449Z",
            "entity_type": "file",
            "modified_by": "226241",
            "modified_on": "2017-02-08T15:31:30.421488Z",
            "name": "file2",
            "parent": "17d74c4d-a253-4d6b-b196-a98f03accf04",
            "uuid": "00ae94e2-f2bf-445a-8a5f-c65b5b422d7d"
        },
        {
            "content_type": "text/plain",
            "created_by": "226241",
            "created_on": "2017-02-08T15:31:01.940271Z",
            "entity_type": "file",
            "modified_by": "226241",
            "modified_on": "2017-02-08T15:31:01.940344Z",
            "name": "file3",
            "parent": "17d74c4d-a253-4d6b-b196-a98f03accf04",
            "uuid": "9cfc012a-8cd6-41f9-a303-f77a45fb1e41"
        }
    ]
}
```

Parameters

- **project_id** (uuid) – project id

Query Parameters

- **page_size** – (optional) number of elements per page (default: 100)

- **page** – (optional) number of the page
- **ordering** – (optional) indicate on which fields to sort the result. Prepend '-' to invert order. Multiple values can be provided. Example: 'ordering=name,created_on'. Ordering is supported on: 'name', 'created_on', 'modified_on', 'content_type'
- **name** – (optional) filter on entity name
- **entity_type** – (optional) filter on entity type. Admitted values: 'file', 'folder'
- **content_type** – (optional) filter on entity content type (only files are returned)

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – entities successfully retrieved
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – project id was not found

Set metadata

POST https://services.humanbrainproject.eu/storage/v1/api/(**string:** *entity_type*)**/uuid:** *entity_id***/metadata/**

Set metadata. Warning: it will replace all existing metadata. Post an empty body to remove all the metadata.

Example request:

```
POST /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "foo": "1000",
    "bar": "2000"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "foo": "1000",
    "bar": "2000"
}
```

JSON Parameters

- **key** (string) – key value pairs

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 201 Created – metadata successfully set
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Get metadata

GET https://services.humanbrainproject.eu/storage/v1/api/(**string:** *entity_type*)**/uuid:** *entity_id***/metadata/**

List metadata.

Example request:

```
GET /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "foo": "1000",
    "bar": "2001",
    "baz": "3000"
}
```

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – metadata successfully retrieved

- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Update metadata

PUT https://services.humanbrainproject.eu/storage/v1/api/(**string:** *entity_type*)**/**
uuid: *entity_id***/metadata/**

Update metadata. Existing metadata will not be affected.

Example request:

```
PUT /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "bar": "2001",
    "baz": "3000"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "foo": "1000",
    "bar": "2001",
    "baz": "3000"
}
```

JSON Parameters

- **key** (string) – key value pairs

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – metadata successfully updated
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Delete metadata

DELETE https://services.humanbrainproject.eu/storage/v1/api/(string: *entity_type*)/uuid: *entity_id*/metadata/

Delete metadata by key. To delete all the metadata see 'Set metadata'.

Example request:

```
DELETE /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "keys": [
        "foo"
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "bar": "2001",
    "baz": "3000"
}
```

JSON Parameters

- **keys** (string) – list of keys to be deleted. Posting an empty key list has no effect.

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – metadata successfully deleted
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Folder endpoint

Create a folder

POST https://services.humanbrainproject.eu/storage/v1/api/folder/

Create a new folder.

Example request:

```
POST /storage/v1/api/folder/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "name": "folder_1",
    "parent": "68d1971d-7293-422e-81aa-06b9de13a461"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "created_by": "226241",
    "created_on": "2017-02-02T14:24:22.785969Z",
    "entity_type": "folder",
    "modified_by": "226241",
    "modified_on": "2017-02-02T14:24:22.786374Z",
    "name": "folder_1",
    "parent": "68d1971d-7293-422e-81aa-06b9de13a461",
    "uuid": "e7c582ce-cb64-43ba-a08a-f1e361df71fa"
}
```

JSON Parameters

- **name** (string) – the name of the folder
- **parent** (uuid) – the uuid of the entity where the folder will be created. The parent entity type must be *project* or *folder*.

Request Headers

- Authorization – OAuth2 token

Status Codes

- 201 Created – folder successfully created
- 400 Bad Request – invalid request. More details in the response. Possible causes are: name already in use, invalid parent invalid type parent
- 403 Forbidden – invalid Oauth2 token provided

Get folder details

GET https://services.humanbrainproject.eu/storage/v1/api/folder/(uuid: *folder_id*)**/**

Get folder info.

Example request:

```
GET /storage/v1/api/folder/e7c582ce-cb64-43ba-a08a-f1e361df71fa/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "created_by": "226241",
    "created_on": "2017-02-02T14:24:22.785969Z",
    "entity_type": "folder",
    "modified_by": "226241",
    "modified_on": "2017-02-02T14:24:22.786374Z",
    "name": "folder_1",
    "parent": "68d1971d-7293-422e-81aa-06b9de13a461",
    "uuid": "e7c582ce-cb64-43ba-a08a-f1e361df71fa"
}
```

Parameters

- **folder_id** (uuid) – folder id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – folder successfully retrieved
- 400 Bad Request – invalid request. More details in the response. Possible causes are: name already in use, invalid parent type, missing field.
- 403 Forbidden – invalid Oauth2 token provided

List folder content

GET https://services.humanbrainproject.eu/storage/v1/api/folder/(uuid: *folder_id*)**/children/**

List all files and folders (not recursively) contained in the folder.

Example request:

```
GET /storage/v1/api/folder/e7c582ce-cb64-43ba-a08a-f1e361df71fa/children/␣
 ↪HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "created_by": "226241",
            "created_on": "2017-02-02T14:26:45.132948Z",
            "entity_type": "folder",
            "modified_by": "226241",
            "modified_on": "2017-02-02T14:26:45.132996Z",
            "name": "folder_1",
            "parent": "e7c582ce-cb64-43ba-a08a-f1e361df71fa",
            "uuid": "098271f4-a073-4125-a738-3dd01ba7e89b"
        },
        {
            "content_type": "application/json",
            "created_by": "226241",
            "created_on": "2017-02-02T14:27:35.621795Z",
            "entity_type": "file",
            "modified_by": "226241",
            "modified_on": "2017-02-02T14:27:35.621840Z",
            "name": "file_1",
            "parent": "e7c582ce-cb64-43ba-a08a-f1e361df71fa",
            "uuid": "77fb486d-21d5-4a24-a48c-af118b1a23e2"
        }
    ]
}
```

Parameters

- **folder_id** (uuid) – folder id

Query Parameters

- **page_size** – (optional) number of elements per page (default: 100)
- **page** – (optional) number of the page
- **ordering** – (optional) indicate on which fields to sort the result. Prepend '-' to invert order. Multiple values can be provided. Example: 'ordering=name,created_on'. Ordering is supported on: 'name', 'created_on', 'modifi
- **name** – (optional) filter on entity name

- **entity_type** – (optional) filter on entity type. Admitted values: 'file', 'folder'
- **content_type** – (optional) filter on entity content type (only files are returned)

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – entities successfully retrieved
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – folder id was not found

Delete a folder

DELETE https://services.humanbrainproject.eu/storage/v1/api/folder/(**uuid:** *folder_id*)**/**

Delete a folder. It will recursively delete all the content.

Example request:

```
DELETE /storage/v1/api/folder/e7c582ce-cb64-43ba-a08a-f1e361df71fa/ HTTP/1.
↪1
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 204 OK
```

Parameters

- **folder_id** (uuid) – folder id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 204 No Content – folder successfully deleted
- 400 Bad Request – invalid request. More details in the response.
- 403 Forbidden – invalid Oauth2 token provided

Set metadata

POST https://services.humanbrainproject.eu/storage/v1/api/(**string:** *entity_type*)**/**
uuid: *entity_id***/metadata/**

Set metadata. Warning: it will replace all existing metadata. Post an empty body to remove all the metadata.

Example request:

```
POST /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "foo": "1000",
    "bar": "2000"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "foo": "1000",
    "bar": "2000"
}
```

JSON Parameters

- **key** (string) – key value pairs

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 201 Created – metadata successfully set
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Get metadata

GET https://services.humanbrainproject.eu/storage/v1/api/(**string:** *entity_type*)**/**
uuid: *entity_id***/metadata/**

List metadata.

Example request:

```
GET /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "foo": "1000",
    "bar": "2001",
    "baz": "3000"
}
```

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – metadata successfully retrieved
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Update metadata

PUT https://services.humanbrainproject.eu/storage/v1/api/(**string:** *entity_type*)**/**
uuid: *entity_id***/metadata/**

Update metadata. Existing metadata will not be affected.

Example request:

```
PUT /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "bar": "2001",
    "baz": "3000"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "foo": "1000",
    "bar": "2001",
    "baz": "3000"
}
```

JSON Parameters

- **key** (string) – key value pairs

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – metadata successfully updated
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Delete metadata

DELETE https://services.humanbrainproject.eu/storage/v1/api/(**string:** *entity_type*)**/**
uuid: *entity_id***/metadata/**

Delete metadata by key. To delete all the metadata see 'Set metadata'.

Example request:

```
DELETE /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "keys": [
        "foo"
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{
    "bar": "2001",
    "baz": "3000"
}
```

JSON Parameters

- **keys** (string) – list of keys to be deleted. Posting an empty key list has no effect.

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – metadata successfully deleted
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

File endpoint

Create a file

POST https://services.humanbrainproject.eu/storage/v1/api/file/

Create a new file.

Example request:

```
POST /storage/v1/api/file/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "name": "file_1",
    "content_type": "plain/text",
    "parent": "e7c582ce-cb64-43ba-a08a-f1e361df71fa"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "content_type": "application/json",
```

```
    "created_by": "226241",
    "created_on": "2017-02-02T14:27:35.621795Z",
    "entity_type": "file",
    "modified_on": "2017-02-02T14:27:35.621840Z",
    "modified_by": "226241",
    "name": "file_1",
    "parent": "e7c582ce-cb64-43ba-a08a-f1e361df71fa",
    "uuid": "77fb486d-21d5-4a24-a48c-af118b1a23e2"
}
```

JSON Parameters

- **name** (string) – the name of the file
- **content_type** (string) – the file content type
- **parent** (uuid) – the uuid of the entity where the file will be created. The parent entity type must be *project* or *folder*.

Request Headers

- Authorization – OAuth2 token

Status Codes

- 201 Created – file successfully created
- 400 Bad Request – invalid request. More details in the response. Possible causes are: name already in use, invalid parent type, missing field.
- 403 Forbidden – invalid Oauth2 token provided

Get file details

GET https://services.humanbrainproject.eu/storage/v1/api/file/(**uuid:** *file_id*)**/**

Get file info.

Example request:

```
GET /storage/v1/api/file/77fb486d-21d5-4a24-a48c-af118b1a23e2/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "content_type": "application/json",
    "created_by": "226241",
    "created_on": "2017-02-02T14:27:35.621795Z",
    "entity_type": "file",
    "modified_on": "2017-02-02T14:27:35.621840Z",
    "modified_by": "226241",
```

```
    "name": "file_1",
    "parent": "e7c582ce-cb64-43ba-a08a-f1e361df71fa",
    "uuid": "77fb486d-21d5-4a24-a48c-af118b1a23e2"
}
```

Parameters

- **file_id** (uuid) – file id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – file successfully retrieved
- 400 Bad Request – invalid request. More details in the response. Possible causes are: name already in use, invalid parent type, missing field.
- 403 Forbidden – invalid Oauth2 token provided

Upload file content

POST https://services.humanbrainproject.eu/storage/v1/api/file/(**uuid:** *file_id*)**/content/upload/**

Upload a file content. File entity must exist (see POST https://services.humanbrainproject.eu/storage/v1/api/file/)

If *ETag* is provided in the *If-Match* header the content of the file on the server is verified against the *ETag* provided. If it does not match *412 Precondition Failed* is returned. This means client needs to update it's knowledge of the resource before attempting to update again. This can be used for optimistic concurrency control.

Example request:

```
POST /storage/v1/api/file/77fb486d-21d5-4a24-a48c-af118b1a23e2/content/
 ↪upload/ HTTP/1.1
Accept: application/json
If-Match: 0ade138937c4b9cb36a28e2edb6485fc
Authorization: Bearer TOKEN
Content-Length: 151
Content-Type: multipart/form-data;␣
 ↪boundary=98c4e0a7bde143acaeafc3e6cd2acd7c
Host: services.humanbrainproject.eu

--98c4e0a7bde143acaeafc3e6cd2acd7c
Content-Disposition: form-data; name="file"; filename="test.txt"

my file content

--98c4e0a7bde143acaeafc3e6cd2acd7c--
```

Example response:

```
HTTP/1.1 201 OK
ETag: "71e1ed9ee52e565a56aec66bc648a32c"
```

Parameters

- **file_id** (uuid) – file id

Request Headers

- Authorization – OAuth2 token

Response Headers

- ETag – file content hash to be used for caching purposes.

Status Codes

- 201 Created – file successfully uploaded
- 400 Bad Request – invalid request. More details in the response.
- 403 Forbidden – invalid Oauth2 token provided
- 412 Precondition Failed – content on the server is defferent from the one expected by client provided by ETag in If-Match.

Copy file content

PUT https://services.humanbrainproject.eu/storage/v1/api/file/(**uuid:** *file_id*)**/content/**

Copy file content from file specified by UUID in *X-Copy-From* header.

File entity must exist (see `POST https://services.humanbrainproject.eu/storage/v1/api/file/`)

Example request:

```
PUT /storage/v1/api/file/77fb486d-21d5-4a24-a48c-af118b1a23e2/content/␣
↪HTTP/1.1
Accept: application/json
X-Copy-From: 8961d83a-cca7-42d2-8cad-d1705d465b38
Authorization: Bearer TOKEN
Content-Length: 0
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 204 OK
```

Parameters

- **file_id** (uuid) – file id

Request Headers

- *X-Copy-From* – UUID of the source file
- Content-Length – to perform copy operation must be set to 0
- Authorization – OAuth2 token

Status Codes

- 204 No Content – file content successfully copied from source
- 400 Bad Request – invalid request. More details in the response.
- 403 Forbidden – invalid Oauth2 token provided

Download file content

GET https://services.humanbrainproject.eu/storage/v1/api/file/(**uuid:** *file_id*)**/content/**

Download a file content.

If *If-None-Match* header with ETag value is provided then in case contend did not change *304 Not Modified* is returned. If contend changed the actual content will be returned along with new *ETag*

Example request:

```
GET /storage/v1/api/file/77fb486d-21d5-4a24-a48c-af118b1a23e2/content/␣
 ↪HTTP/1.1
Accept: */*
If-None-Match: 0ade138937c4b9cb36a28e2edb6485fc
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Disposition: attachment; filename=file_1
Content-Length: 151
Content-Type: plain/text
ETag: "71e1ed9ee52e565a56aec66bc648a32c"
Last-Modified: Thu, 02 Feb 2017 14:57:31 GMT

--98c4e0a7bde143acaeafc3e6cd2acd7c
Content-Disposition: form-data; name="file"; filename="test.txt"

my file content

--98c4e0a7bde143acaeafc3e6cd2acd7c--
```

Parameters

- **file_id** (uuid) – file id

Request Headers

- If-None-Match – ETag
- Authorization – OAuth2 token

Response Headers

- ETag – file content hash to be used for caching purposes.

Status Codes

- 200 OK – file successfully downloaded
- 304 Not Modified – file content matched If-None-Match Etag and no content was returned
- 400 Bad Request – invalid request. More details in the response.
- 403 Forbidden – invalid Oauth2 token provided

Get signed URL

GET https://services.humanbrainproject.eu/storage/v1/api/file/(**uuid:** *file_id*)**/content/secure_link/**

Get a signed unauthenticated URL to download the file content without the need for a token. The signed URL expires after 5 seconds.

Example request:

```
GET /storage/v1/api/file/77fb486d-21d5-4a24-a48c-af118b1a23e2/content/
 ↪secure_link/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "signed_url": "/file/17d74c4d-a253-4d6b-b196-a98f03accf04/9cfc012a-
 ↪8cd6-41f9-a303-f77a45fb1e41/content/?expires=1486&hash=eGwevqxqw&
 ↪name=file3&content_type=text/plain"
}
```

Parameters

- **file_id** (uuid) – file id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – signed url successfully generated
- 403 Forbidden – invalid Oauth2 token provided

Delete a file

DELETE https://services.humanbrainproject.eu/storage/v1/api/file/(**uuid:** *file_id*)**/**

Delete a file.

Example request:

```
DELETE /storage/v1/api/file/77fb486d-21d5-4a24-a48c-af118b1a23e2/ HTTP/1.1
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 204 OK
```

Parameters

- **file_id** (uuid) – file id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 204 No Content – file successfully deleted
- 400 Bad Request – invalid request. More details in the response.
- 403 Forbidden – invalid Oauth2 token provided

Set metadata

POST **https://services.humanbrainproject.eu/storage/v1/api/**(**string:** *entity_type*)**/**
uuid: *entity_id***/metadata/**

Set metadata. Warning: it will replace all existing metadata. Post an empty body to remove all the metadata.

Example request:

```
POST /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "foo": "1000",
    "bar": "2000"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "foo": "1000",
    "bar": "2000"
}
```

JSON Parameters

- **key** (string) – key value pairs

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 201 Created – metadata successfully set
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Get metadata

GET https://services.humanbrainproject.eu/storage/v1/api/(**string:** *entity_type*)**/uuid:** *entity_id***/metadata/**

List metadata.

Example request:

```
GET /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Host: services.humanbrainproject.eu
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "foo": "1000",
    "bar": "2001",
    "baz": "3000"
}
```

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – metadata successfully retrieved
- 403 Forbidden – invalid Oauth2 token provided

- 404 Not Found – entity id was not found

Update metadata

PUT https://services.humanbrainproject.eu/storage/v1/api/(**string:** *entity_type*)**/**
uuid: *entity_id***/metadata/**

Update metadata. Existing metadata will not be affected.

Example request:

```
PUT /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "bar": "2001",
    "baz": "3000"
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "foo": "1000",
    "bar": "2001",
    "baz": "3000"
}
```

JSON Parameters

- **key** (string) – key value pairs

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – metadata successfully updated
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Delete metadata

DELETE https://services.humanbrainproject.eu/storage/v1/api/(string: *entity_type*)/uuid: *entity_id*/metadata/

Delete metadata by key. To delete all the metadata see 'Set metadata'.

Example request:

```
DELETE /storage/v1/api/$ENTITY_TYPE$/8609a4e9-baf5-4cf5-b3c6-7a98b21ceee3/
 ↪metadata/ HTTP/1.1
Accept: application/json
Authorization: Bearer TOKEN
Content-Type: application/json
Host: services.humanbrainproject.eu

{
    "keys": [
        "foo"
    ]
}
```

Example response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "bar": "2001",
    "baz": "3000"
}
```

JSON Parameters

- **keys** (string) – list of keys to be deleted. Posting an empty key list has no effect.

Parameters

- **entity_type** (string) – possible values: *project*, *folder*, *file*
- **entity_id** (uuid) – entity id

Request Headers

- Authorization – OAuth2 token

Status Codes

- 200 OK – metadata successfully deleted
- 403 Forbidden – invalid Oauth2 token provided
- 404 Not Found – entity id was not found

Migration from V0

This goal of the migration guide is to help the user to migrate a client using document-service v0 apis to use the v1 of the api described in the previous section.

The main change is the renaming of the properties of the resources. See *Properties: V0 > V1 mapping* for more details.

Models: v0 > v1 properties mapping

Project

v0	v1
_uuid	uuid
_name	name
_createdBy	created_by
_createdOn	created_on
_description	description
_entityType	entity_type
_modifiedOn	modified_on
_parent	-
_contentType	-
-	collab_id
-	modified_by

Folder

v0	v1
_uuid	uuid
_name	name
_createdBy	created_by
_createdOn	created_on
_entityType	entity_type
_modifiedOn	modified_on
_parent	parent
-	collab_id
-	modified_by

File

v0	v1
_uuid	uuid
_parent	parent
_name	name
_createdBy	created_by
_createdOn	created_on
_description	description
_entityType	entity_type
_modifiedOn	modified_on
_contentType	content_type
_fileSize	-
_contentUri	-
-	modified_by

Project list & Project/Folder content

Endpoints affected:

- /document/v0/api/project/
- /document/v0/api/project/$UUID/children
- /document/v0/api/folder/$UUID/children

v0	v1	notes
result	results	-
hasMore	-	replaced by *next* and *previous*
-	next	url of the next page (can be null)
-	previous	url of the previous page (can be null)
-	count	total number or elements

Endpoints: v0 > v1 mapping

As a general rule, the context path of the API changed from */document/v0/api/* to */storage/v1/api/entity*. If an endpoint is not listed in the next section, it means that the nothing changed, except for the context path.

Entity

method	v0	v1	notes
GET	/document/v0/api/entity/$UUID	/storage/v1/api/entity/$UUID/ or /storage/v1/api/entity/?uuid=$UUID	
GET	/document/v0/api/entity/?managed_by_collab=$COLLAB_ID	/storage/v1/api/project/?collab_id=$COLLAB_ID	Note the endpoint change (entity > project)
GET	/document/v0/api/entity_path/$UUID	/storage/v1/api/entity/$UUID/path	returns a json with a single field named *path*

Project

method	v0	v1	notes
POST	/document/v0/api/project/	not supported	In v1 projects exist only if linked to a collab; so the creation is delegate to the collab service.
DELETE	/document/v0/api/project/$UUID/	not supported	In v1 projects exist only if linked to a collab; so deletion is delegate to the collab service.
GET	/document/v0/api/project/?filter=$FIELD:$VALUE	/storage/v1/api/project/?$FIELD=$VALUE	see *Project* api doc for the list of supported fields
GET	/document/v0/api/project/?sort=$FIELD	/storage/v1/api/project/?ordering=$FIELD	see *Project* api doc for the list of supported fields
GET	/document/v0/api/project/?from=$UUID	not supported	Replaced with page size based pagination. See *Project* api doc for more details
GET	/document/v0/api/project/?until=$UUID	not supported	Replaced with page size based pagination. See *Project* api doc for more details
GET	/document/v0/api/project/?limit=$PAGE_SIZE	/storage/v1/api/project/?page_size=$PAGE_SIZE&page=$PAGE_NUMBER	see *Project* api doc for the list of supported fields
GET	/document/v0/api/project/$UUID/children/?filter=$FIELD:$VALUE	/storage/v1/api/project/$UUID/children/?$FIELD=$VALUE	see *Project* api doc for the list of supported fields
GET	/document/v0/api/project/$UUID/children/?sort=$FIELD	/storage/v1/api/project/$UUID/children/?ordering=$FIELD	Replaced with page size based pagination. See *Project* api doc for more details
GET	/document/v0/api/project/$UUID/children/?from=$UUID	not supported	Replaced with page size based pagination. See *Project* api doc for more details
GET	/document/v0/api/project/$UUID/children/?until=$UUID	not supported	
GET	/document/v0/api/project/$UUID/children/?limit=$PAGE_SIZE	/storage/v1/api/project/$UUID/children/?page_size=$PAGE_SIZE&page=$PAGE_N	

Folder

method	v0	v1	notes
GET	/document/v0/api/folder/$UUID/?folder=$FIELD:$VALUE	/storage/v1/api/folder/$UUID/?folder=$FIELD:$VALUE	see *Folder* api doc for the list of supported fields
GET	/document/v0/api/folder/$UUID/children/?order=$FIELD	/storage/v1/api/folder/$UUID/children/?order=$FIELD	see *Folder* api doc for the list of supported fields
GET	/document/v0/api/folder/$UUID/children/?from=$UUID	not supported	Replaced with page size based pagination. See *Folder* api doc for more details
GET	/document/v0/api/folder/$UUID/children/?until=$UUID	not supported	Replaced with page size based pagination. See *Folder* api doc for more details
GET	/document/v0/api/folder/$UUID/children/?page=$PAGE_NUM&size=$PAGE_SIZE	/storage/v1/api/folder/$UUID/children/?page_size=$PAGE_SIZE&page=$PAGE_N	

File

method	v0	v1	notes
POST	/document/v0/api/file/	/storage/v1/api/file/	*content_type* field is now required.
POST	/document/v0/api/file/$UUID/content/	/storage/v1/api/file/$UUID/content/	

Miscellaneous

- In v1 projects exist only attached to a Collab and each collab can be linked to one project only. For this reason, the creation and deletion of collabs from the API is not supported anymore.
- All the v1 endpoints require a trailing slash ('/'). If not provided, a permanent redirect (301) is returned.

Introduction

HBP storage provides Collab storage capabilities and access to the HPC Unicore enabled sites.

Related Resources

- HBP Collab Rest Service API Documentation

CHAPTER

FOUR

CONTACT & SUPPORT

Contact

Who is in charge of Collaboratory ?

how can I contact them ?

Support

If I have an issue with Collaboratory, who should I contact ?

Is there a bug tracking system ?

Is part of the code open sourced, if yes where can I find it ?

CHAPTER

FIVE

LICENSE

What the project license ?

CHAPTER

SIX

FREQUENTLY ASKED QUESTIONS

What are the most common asked questions ?

[CIT001] Stream Framework documentation, http://feedly.readthedocs.org/en/latest/notification_systems.html (2015/12/1)

/https:

T

U