BREAKING EDUCATIONAL BARRIERS WITH CONTEXTUALISED PERVASIVE AND GAMEFUL LEARNING

Grant Agreement No. 687676

Innovation Action

ICT-20-2015

D4.1 Integrated BEACONING Ecosystem

Due date	M18
Actual date	M18
Deliverable author(s)	Antoniu Ștefan, Ioana Andreea Ștefan, Bogdan Drăgulin
Partner(s)	ATS
Contributors	Ancuța Florentina Gheorghe
Version	V7
Status	Final
Dissemination level	PU

Project Coordinator

Coventry University

Sylvester Arnab

Priory Street, Coventry CV1 5FB, UK

E-mail: s.arnab@coventry.ac.uk

Project website: http://www.beaconing.eu

This project has received funding under the Horizon 2020 Framework Programme of the European Union – Grant Agreement No. 687676

Version control				
Version	**Date**	**Author**	**Institution**	**Change and where applicable reason for change**
V1	27.01.2017	Ioana Andreea Ștefan	ATS	Initial draft of the deliverable
V2	08.05.2017	Antoniu Ștefan	ATS	Initial description of the core services, of the personal data store, and of the cloud-based approach
V5	09.06.2017	Antoniu Ștefan	ATS	Implemented proposed changes
V6	16.06.2017	Antoniu Ștefan, Bogdan Drăgulin, Ioana Andreea Ștefan	ATS	Added additional API descriptions Refinement of chapters 2, 3 and 4

Quality control				
QA Version	**Date**	**QA Responsible**	**Institution**	**Change and where applicable reason for change**
V3	19.05.2017	Adrien Tiévant	Playsoft	Internal review, see track changes
V4	22.05.2017	Baltasar Fernandez Manjon	UCM	Internal review, see track changes
V6	28.06.2017	Jannicke Baalsrud Hauge	BIBA	QM
V7	29.06.2017	Jayne Beaufoy	COVUNI	Language check and formatting

Release approval				
Version	**Date**	**Name**	**Institution**	**Role**
V7	29.06.2017	Jannicke Baalsrud Hauge	BIBA	QM

Statement of originality

This deliverable contains original unpublished work except where clearly indicated otherwise. Acknowledgement of previously published material and of the work of others has been made through appropriate citation, quotation or both.

Contributors

Contributors				
Version of document contributed to	Chapter	Contribution description	Institution	Name of contributor
V5	Chapter 2	Describing user authentication flow	ATS	Ancuța Florentina Gheorghe

TABLE OF CONTENTS

EXECUTIVE SUMMARY **6**

1 INTRODUCTION **7**

1.1 BACKGROUND 7
1.2 ROLE OF THIS DELIVERABLE IN THE PROJECT 7
1.3 APPROACH 7
1.4 STRUCTURE OF THE DOCUMENT 8

2 CORE SERVICES OF THE BEACONING PLATFORM **9**

2.1 SINGLE SIGN-ON 9
2.1.1 User authentication flow *11*
2.1.2 Refreshing the access token *12*
2.2 USER MANAGEMENT 13
2.3 BEACONING ASSET MANAGEMENT 14
2.4 MINIGAME MANAGEMENT 15
2.5 GAME PLOT MANAGEMENT 18

3 PERSONAL DATA STORE **19**

4 CLOUD-BASED PLATFORM **21**

5 CONCLUSION **22**

5.1 RESULTS 22
5.2 IMPACT 22

6 REFERENCES **23**

LIST OF FIGURES

Figure 1. Insecure scenario where user credentials are passed between services 10
Figure 2. The secure authentication workflow implemented in BEACONING using OAuth 2.0 11
Figure 3. Using the interactive documentation to retrieve the list of minigames 16
Figure 4. Retrieving details for one minigame 17
Figure 5. Registering a new minigame 17
Figure 6. Updating a minigame 18
Figure 7. Deleting a minigame 18
Figure 8. Interaction workflow for obtaining real identities of the students 19

LIST OF TABLES

Table 1. Managing users and groups 13
Table 2. Security prerequisites and requirements 14
Table 3. Asset Library Functions 15
Table 4. Minigame management 16

ACRONYMS

AES	Advanced Encryption Standard
API	Application Programming Interface
HTTP	Hypertext Transfer Protocol
GPE	Game Plot Editor
JSON	JavaScript Object Notation
PDS	Personal Data Store
SSO	Single Sign On
UX	User Experience
URL	Uniform Resource Locator

EXECUTIVE SUMMARY

Deliverable 4.1 Integrated BEACONING Ecosystem implements part of Deliverable 3.6 System architecture and presents the core services of the cloud-based learning platform, describing the key functions that carry out the asset management, the minigames management, and the game plots management. The services provide a full-featured built-in user management that can easily authenticate users across platform components.

The deliverable introduces the Personal Data Store component that concerns privacy protections of student data and sensitive data management during small- and large-scale piloting, in accordance with the Data Management Plans defined in D1.7 and D1.8. It details the interaction workflow for obtaining and managing the real identities of the students involved in BEACONING piloting.

The system is based on an open architecture that allows new components to be integrated using a set of documented APIs, and maintains an equal level of privileges between build-in components and third-party components from the ecosystem.

To facilitate increased user adoption, the services are provided through a cloud-based approach but can also be deployed locally if needed.

An updated version of this deliverable will be provided in M28.

1 INTRODUCTION

1.1 BACKGROUND

Deliverable 4.1 Integrated BEACONING Ecosystem presents the core services developed for the BEACONING Platform and Ecosystem. These services streamline the user experience at platform/ ecosystem level and can be called by the key BEACONING components: the context-aware system; the Game Plot Editor; the Gamified lesson path runtime: the Minigame runtime; the Authoring System; the Procedural Content Generation; the Game Plot Editor; and the Learning analytics services.

By grouping these core services into a specialised component with open API access, the BEACONING Platform supports an ecosystem approach, where new applications and tools can be easily integrated, without impacts on the performance of existing components. The services can further be enhanced to provide a market place for content authors and consumers.

The BEACONING consortium understands the value of personal data privacy and therefore has implemented an architecture where personal identification data is managed using a specialized service that can be hosted locally by each piloting organisation, in accordance with internal privacy policies and local jurisdiction regulations.

1.2 ROLE OF THIS DELIVERABLE IN THE PROJECT

Building and maintaining a sustainable learning ecosystem implies delivering peak performance on all levels. Deliverable 4.1 Integrated BEACONING Ecosystem builds upon the Deliverable 3.6 System architecture and implements the Data Management Plan that have been detailed in D1.7 delivered in M6, as well as the updated version D1.8 delivered in M12. The BEACONING core services will be tested in T5.1 Test of the single components and T5.2 Integration testing.

An updated version of the Integrated BEACONING Ecosystem will be provided in M28. The updated deliverable will take into consideration the output of the testing tasks and of the small and large-scale piloting carried out in WP5 Unit Testing and Small Scale Pilot and WP6 Large Scale Pilot.

1.3 APPROACH

Effective development of learning services and the integration of a learning ecosystem require optimum performance with regard to the back-end components. The BEACONING Ecosystem implements a REST architecture in order to make it sustainable and easily scalable.

The core services of the BEACONING Platform are implemented using the Node.js application platform, which is based on JavaScript and allows better integration and data transfer with front-end web apps. Node.js is tailored for creating distributed REST services with low overheads and quick response times.

These are backend services that do not provide a User Interface, with the exception of the authentication workflows when the services need to ask the user for login credentials.

In order to minimize deployment and maintenance costs, development of the services has been carried out using cross platform open source components that allow deployment in multiple scenarios.

1.4 STRUCTURE OF THE DOCUMENT

Section 1 describes the interconnections between the core services developed in T4.1 and ongoing testing and piloting tasks. It details the development approach.

Section 2 presents the core services of the BEACONING Platform and Ecosystem that can be called by the key components developed within the project. The current version of the core services includes authentication, user profile management, asset management, mini-games, Game Plot Editor, and the Gamified Lesson Paths.

Section 3 describes the personal data management component that prohibits the transfer of personal student information in certain contexts and provides a coherent conception of the privacy approach across the collection and dissemination of student information.

Section 4 presents the conclusions of the development processed for the back-end services of the BEACONING Platform and Ecosystem.

2 CORE SERVICES OF THE BEACONING PLATFORM

This section presents the backend components that support the delivery of integrated learning services via the cloud-based BEACONING Platform and Ecosystem. These components have been developed in accordance with Deliverable 3.6 System architecture and they expose a set of managed resources for the purpose of querying and manipulation.

Among the key benefits that the core services bring, we can list the following: reduces unnecessary code redundancy; optimizes infrastructure usage; and provides centralized master data management.

2.1 SINGLE SIGN-ON

As the proper function of a learning ecosystem is based on the interactions between a wide range of applications and tools, the main challenges for every educational organisation are enabling seamless communication across these applications, merging existing applications implemented in educational organisations with new solutions and enhancing the User Experience (UX) by providing single authentication. The BEACONING solution aims to facilitate the execution of learning services and transform them into a new simplified digital experience for teachers, students, and parents by initiating a paradigm shift in organising and conducting interactions via technology and improving collaborations between stakeholders.

Learning services provided by BEACONING should appear to their end users as compact and unified entities, regardless of the medium of interactions. In this way, educational organisations enable synergic effects for end users, making their experience with technology more efficient. Transactions that are part of this type of interactions can be divided into three groups:

- The first group of transactions corresponds to the student level and includes the interactions with game-based technologies. Collecting feedback can help estimate the effects of the changes and enable the discovery of good practices.
- The second group of transactions corresponds to the teacher level and relates to managing gamified learning experiences and unified learning analytics.
- The third group of transactions corresponds to the teacher level and includes the exchange of information about student progress for various learning activities. It also includes feedback from parents

The BEACONING platform includes single sign-on services that components can use to authenticate and authorize users. These are part of the core service component and are implemented based on the OAuth 2.0 open standard. OAuth 2.0 is a protocol that allows distinct parties to share information and resources in a secure and reliable manner [1]. The use of OAuth 2.0 allows:

- *Federated identity*. A service provider allows authentication of a user using their identity with another service provider. Users do not need to create individual accounts. The user only needs to maintain a single user account, which gives them access to several service providers. For example, a BEACONING student can use the same account to play the minigames in the Gamified Lesson Paths and also access their learning analytics dashboard. They do not need to create individual accounts or maintain two separate passwords. The user's identity across these sites are federated.

- *Delegated authority*. A service or an application has the ability to gain access to a user's resources on their behalf. For example, the BEACONING Platform accessing a user's Facebook photos on their behalf.

The BEACONING core services component provides:

- User authentication that validates whether a user or a system is actually who they say they are.
- User authorization that determines what actions are allowed to be performed once a user has been authenticated.

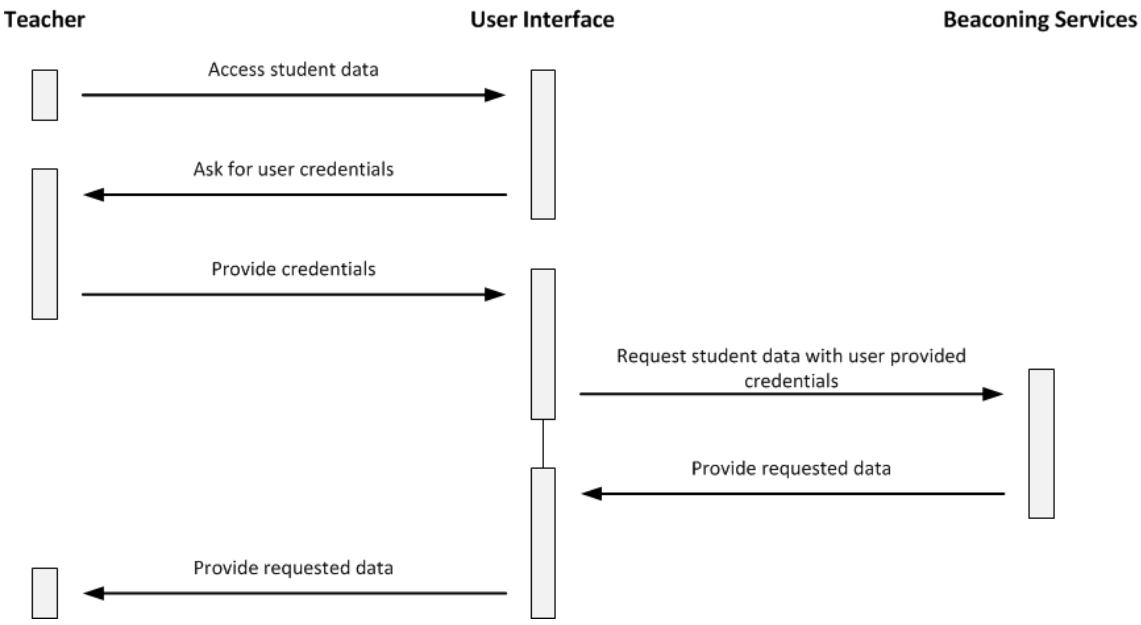

Figure 1. Insecure scenario where user credentials are passed between services

Figure 1 describes a possible scenario for authenticating users when accessing BEACONING platform components and is based on delegating user access credentials to each specific component that is accessed. This is an insecure scenario because it creates multiple, vulnerable points, where credentials can be handled/ stored incorrectly and it also implies that the user trusts all the components that they need to access, including third party components that were developed outside the BEACONING consortium.

Figure 2 describes the approach implemented in BEACONING, using the OAuth 2.0 authentication framework, where services need to ask for specific permissions from the identity provider authority and access is granted using time-limited/ expiring tokens. In this approach, user credentials are handled by only the core services component.

Handling user identity through this centralized component also allows future extensibility to act as a delegated authority and integrate seamless authentication on top of other backend identity services such as LDAP, Google, Facebook or custom implementations for various LMSs.

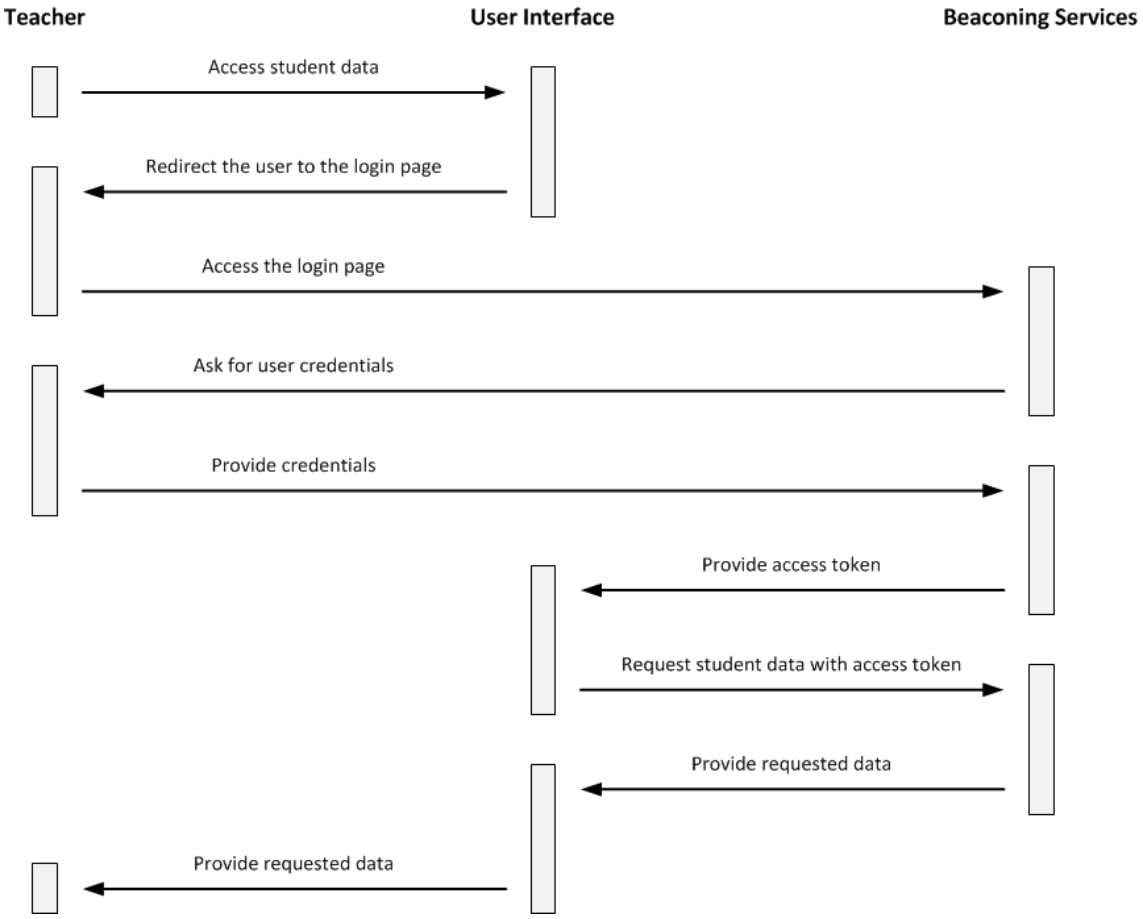

Figure 2. The secure authentication workflow implemented in BEACONING using OAuth 2.0

2.1.1 User authentication flow

To authenticate a user into the BEACONING system, the client application should access the following URL:

https://core.beaconing.eu/auth/auth?response_type=code&client_id=some_client_id_here&redirect_uri=some_callback_url_here

The callback URL has to be https and match the one defined for the client_id in the system configuration.

The user will be presented with a login form where they can type in their username and password. The core services will attempt to validate these credentials and if successful, will use the callback URL to provide an access code in the following format:

some_callback_url_here?code=generated_auth_code_here

The client application backend services responding for the callback should use the temporary authorization code to ask for an access token. The temporary authorization code is valid for 10 minutes.

The client application backend services should call the following endpoint:

https://core.beaconing.eu/auth/token

Using a POST verb with Content-Type: application/json and a request body containing a JSON document using the following structure:

```
{
 "grant_type":"authorization_code",
 "code":" generated_auth_code_here ",
 "client_id":"some_client_id_here ",
 "client_secret":"some_client_secret_here",
 "redirect_uri":"callback_url_here_same_as_initial_request"
}
```

If the code and client credentials are valid, the authentication service will respond with a JSON document with the following structure:

```
{
 "access_token":"generated_access_token_here",
 "refresh_token":"generated_refresh_token_here",
 "expires_in":3600,
 "token_type":"Bearer"
}
```

The access token can then be passed on to the application front-end for making API calls, but the refresh token should be kept private on the server side and reused only when needing to refresh the *access_token*.

The *expires_in* property indicates the number of seconds that the access token is considered valid.

2.1.2 Refreshing the access token

When the access token has expired (based on the moment it was issued and the *expires_in* property), the client application backend must ask for a new one using the refresh token, calling the same endpoint as for the initial authorization code grant:

https://core.beaconing.eu/auth/token

using a POST verb with Content-Type: application/json and a request body containing a JSON document using the following structure:

```
{
 "client_id":"some_client_id_here",
 "client_secret":"some_client_secret_here",
 "refresh_token":"the_refresh_token_received_previously",
 "grant_type":"refresh_token"
}
```

If the refresh token and client credentials are valid, the authentication service will reply with a JSON document with the following structure:

```
{
 "access_token": "generated_access_token_here",
```

"expires_in": 3600,
"token_type": "Bearer"
}

2.2 USER MANAGEMENT

The core services component provides centralized storage, management and auditing for user profile data. This component implements data privacy settings. The main user categories managed at Platform level are teachers, students, and parents. While the privacy policy applied to all categories of users and user data is encrypted at rest and in flight, for students additional protections are implemented to depersonalize the student data.

Table 1 describes the main functionalities of the user management services.

Table 1. Managing users and groups

Functionality	Description
Create a user account	Teachers, students, and parents can create accounts on the BEACONING Platform. The User Interface can call this functionality, when a new user wants to sign up or when teachers are creating new accounts for their students.
Manage user account	Users can update their account information and reset lost passwords.
Create user accounts in batch mode	This functionality can be used when the BEACONING Platform is implemented within a new organisation. It allows bulk management of user accounts.
Create & manage groups and members	These functionalities are called when teachers create groups for their students. Groups can be created at class level, but also for student teams or cross-class collaborations. This approach enables gamified learning, where specific learning tasks can be assigned to student teams.
Delete a user account	Admin users and teachers can use this functionality.
Grant & revoke user permissions	This functionality allows the setting up of access rights for teachers and other staff of an educational organisation using the BEACONING Platform.

In accordance with the guidelines on data privacy (Deliverable 1.7 and 1.8 Data management plan), the student profile and other student related data will be identified using auto-

generated surrogate keys, while the real identities of the persons will be mapped to these keys through the personal data store. This allows system components to collect, manage and analyse student data without knowledge of the student's real identity. The mapping between surrogate identifiers and real identities is carried out using a client side library implemented in JavaScript.

The library can be easily integrated into front-end dashboards, web reports and other UI components and relies on the fact that it can query the PDS even if it is located in the pilot organization's internal network, without communication with the central BEACONING platform.

The BEACONING solution contains security features that ensure that specific users will only have access to information that they are supposed to have. Since the BEACONING Platform consists of base and extended server components, security must also be set up for each server component, otherwise users may not be able to access/ modify data or unauthorised users can access data that should be confidential.

Below is a list of the common security prerequisites and requirements for BEACONING components. It can also be used as a checklist in testing and piloting to confirm that the security requirements for each component have been set up for local instances of BEACONING components.

Table 2. Security prerequisites and requirements

Requirement	Description
TLS server certificate	Each network visible service must be protected using public key certificates to prevent information leakage and main the middle attacks
Data storage encryption	All user related data, including de-personalized data, that is stored on private or public servers must be encrypted at rest.
Firewall setup	Network configuration must prevent access to other service ports than the ones intentionally exposed by the components (most commonly HTTP and HTTPS).

2.3 BEACONING ASSET MANAGEMENT

The BEACONING Asset Library provides core process services, including specific facilities for the management, manipulation, security, movement and distribution, processing, storage, as well as search and retrieval of digital assets.

A BEACONING asset is a reusable resource that can be used by game developers and learning designers during the authoring process (E.g. graphics, audio, etc.) Game assets have specialized requirements for manipulation, long-term storage, defining standards, etc. Game assets are usually managed by specialized teams and their handling, processing, and storage requires specialized workflows to ensure consistency across development processes. Because

game assets are non-textual, they require additional textual information (metadata) that describes them. Thoroughly defined metadata enhances search experiences.

The Platform enables the management of asset metadata, according to the following categories:

- *Implicit*: The asset values are inferred from the file's physical characteristics. E.g. name of the asset, file size, etc.
- *Explicit*: a person or an automated process made some judgement about the assets. E.g. subject, ranking, category, value, etc.

Table 3 describes the main functions of the asset library.

Table 3. Asset Library Functions

Functions	Description
Get	Returns the content and metadata of an asset based on its ID.
Post	Allows creation of a new asset in the Asset Library.
Put	Allows the caller to update an existing asset from the Library.
Delete	Can be used to delete an asset based on its ID. The system will only make the asset unavailable for future use, but the asset can still be retrieved by existing game plots that reference it.
List	Returns a list of all available assets that are stored in the Library.
Search	Can be used to locate an asset stored in the Library.

The asset library stores assets as binaries with associated metadata. It can store multiple file types and allow for the customization of metadata fields. Besides user input metadata, the asset library also stores automatically collected information related to the usage of each asset.

The asset library also includes licensing descriptions in the resource metadata to allow end users to select resources based on their intended usage scenarios.

Assets are tracked together with relationships with other assets to facilitate retrieval and searching of related resources.

Each asset has a permanent URL link associated to facilitate adoption allows assets to be embedded in multiple scenarios.

The asset library allows users to find assets by searching metadata descriptors such as author, license and format.

2.4 MINIGAME MANAGEMENT

Mini-games are hosted independently by each provider, but are registered with this service to allow listing and locating these mini-games.

Table 4. Minigame management

Functions	Description
Get	Retrieve the list of available mini-games and their descriptive metadata, for customization purposes.
Post	Allows creation of a new mini-game resource in the library.
Put	Allows the update of an existing mini-game from the library.
Delete	Can be used to delete a mini-game registration from the library. The mini-game can still be accessed by existing gamified lesson paths, but will not be displayed for selection for new content.
List	Returns a list of mini-games that are available.
Search	Returns a list of mini-games that fit the search criteria.

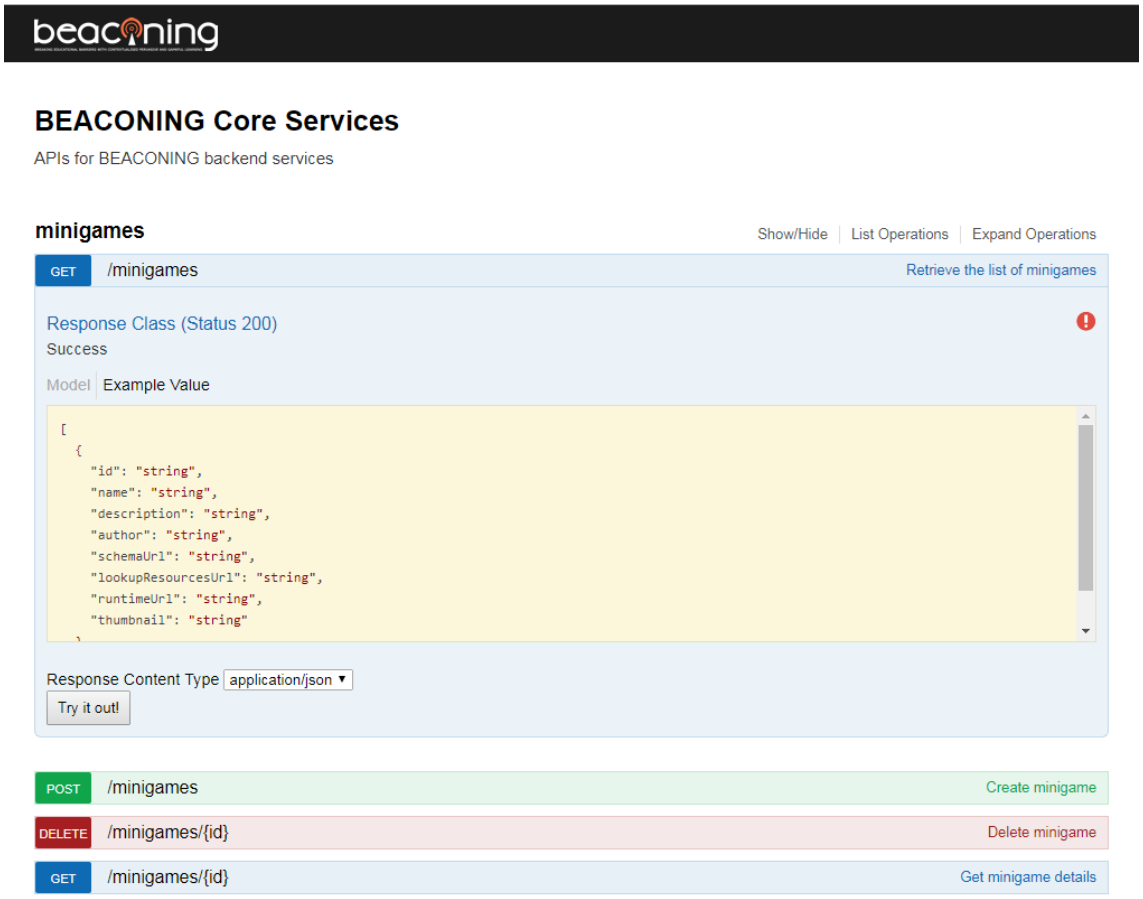

Figure 3. Using the interactive documentation to retrieve the list of minigames

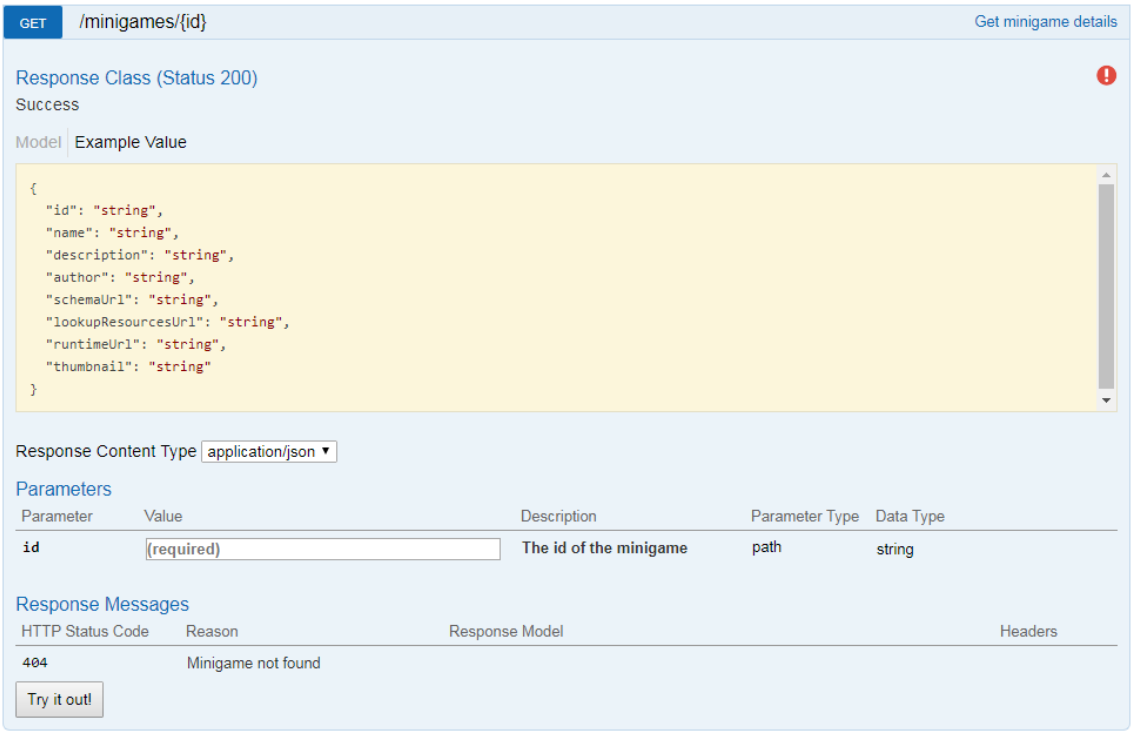

Figure 4. Retrieving details for one minigame

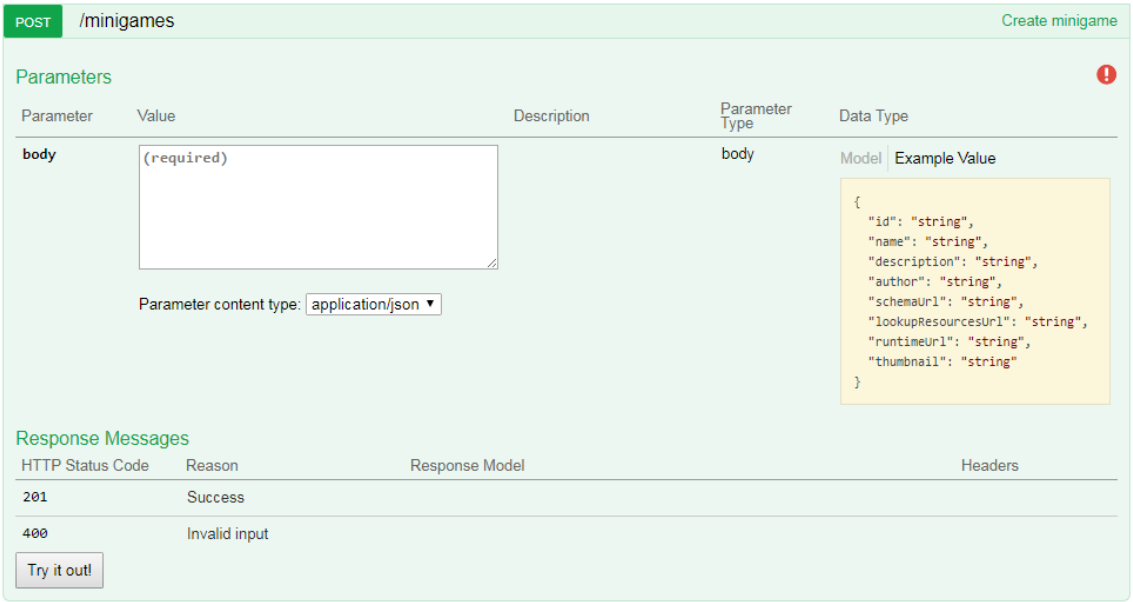

Figure 5. Registering a new minigame

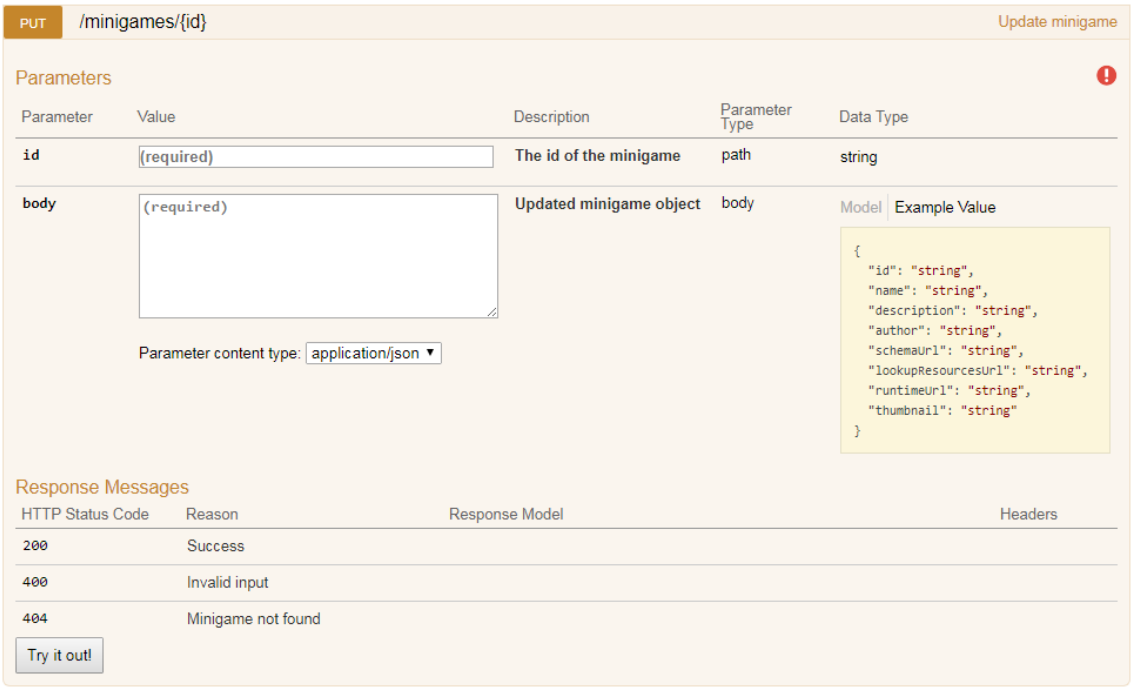

Figure 6. Updating a minigame

Figure 7. Deleting a minigame

2.5 GAME PLOT MANAGEMENT

The BEACONING core services component enables the Game Plot Editor (GPE) to save new game plot scenarios that are created by game designers and edit the existing plots. The GPE can call the core services to retrieve a list of available assets and mini-games that can be included in the game plot.

3 PERSONAL DATA STORE

Digitized data is collected during practically every interaction with the BEACONING Platform. This information is either consciously (e.g. a user provides their e-mail address to register to a learning service) or unconsciously transferred (e.g. a learning service keeps track of the posts a user reads). As a result, online learning services build up extensive, detailed backgrounds on the usage habits of their users, usually correlated with personal information such as age, gender, ethnicity, or other details that the user has either explicitly provided or that can be inferred [2]. While data could be anonymized to prevent the association between the user's identity and the user actions, online communication requires the identification of the user, and therefore the system implements de-personalization as a form of data privacy protection instead of full anonymity. This is in line with the EU General Data Protection Regulation [3] that recommends the use of pseudonymisation for such scenarios.

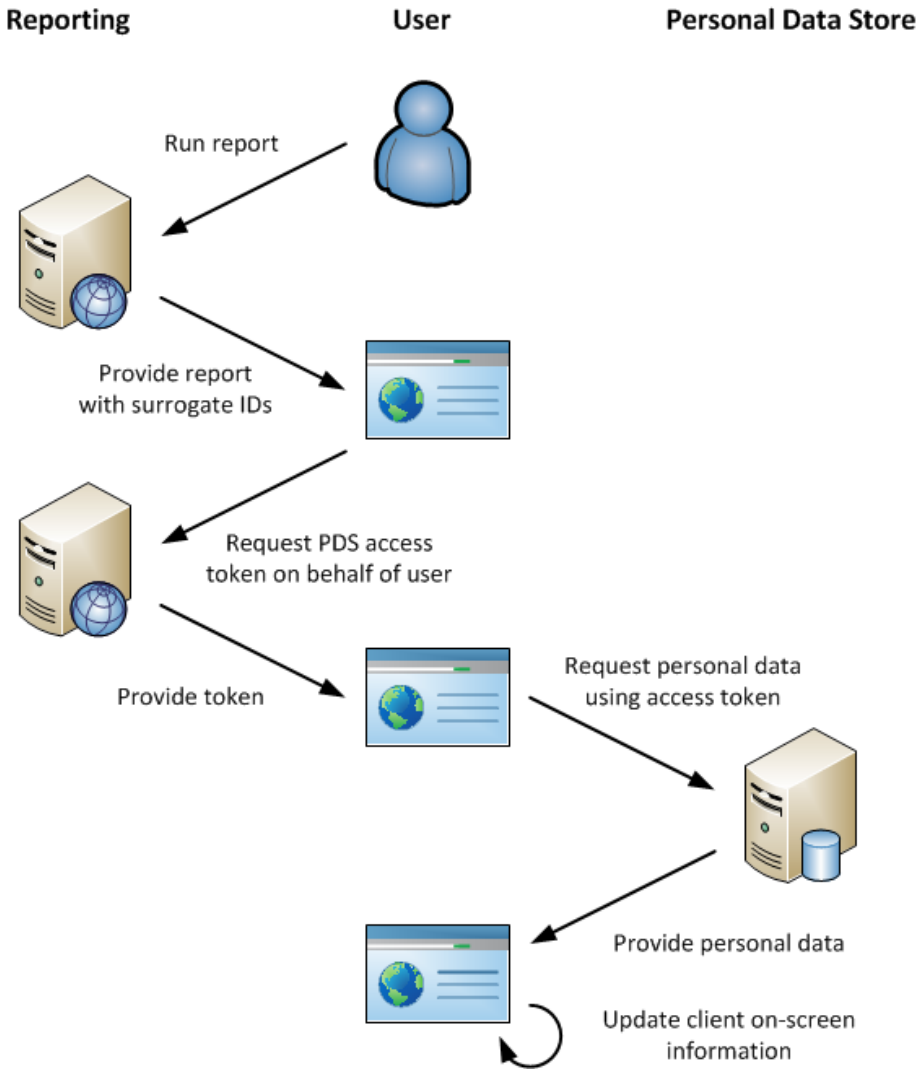

Figure 8. Interaction workflow for obtaining real identities of the students

The Personal Data Store (PDS) is a component that stores real student identities and the associated surrogate ID that was generated by the system (Fig. 4). By querying this component,

other actors in the ecosystem can display the real identities of the students in user interfaces and reports.

This is managed as an independent component that can be hosted under the direct supervision of the piloting organization and in accordance with its data storage policies and local regulation requirements.

Reports that contain sensitive data by default will not contain the real identities of the students, and instead will show internally generated ids (surrogates). Using a JavaScript library, the real identity of each student can be queried from the PDS and the code will then update the information displayed on-screen with the retrieved identities.

Access to the component will be controlled and audited by the core services component, which must generate a special time limited access token for each query to the PDS. This prevents direct queries to the PDS by unauthorized third parties, and mitigates the impact of leaked access tokens. By default, the validity of the access token will be 10 minutes and can be configured by each implementing organization if needed.

The PDS component is currently under development and an updated version will be described in the revised deliverable in M28.

4 CLOUD-BASED PLATFORM

The confluence of the BEACONING ecosystem approach and large-scale piloting associates with vast quantities of data, and therefore requires increased storage capacity and computing power. Managing abundant data collections efficiently enables the extraction of useful knowledge on learning activities and the discovery of strengths and weaknesses concerning both the learning services, and the student performance. However, this ability presents challenges to governance and privacy.

The system is based on an open architecture that allows new components to be integrated using a set of documented APIs, and maintains an equal level of privileges between build-in components and third-party components from the ecosystem.

Wide user adoption is considered to be essential for learning services. To facilitate increased user adoption, the services are provided through a cloud-based approach but can also be deployed locally if needed.

The BEACONING solution is being developed in-line with the following objectives:

- Automate the deployment of the applications in the BEACONING ecosystem.
- Develop a dynamic and robust cloud computing security system for large-scale piloting.
- Intensify and automate security mechanisms in all stages of the system.
- Use an OAuth 2.0 based framework to authenticate users through-out the ecosystem.
- Use strong encryption algorithms to encrypt data at rest.
- Reduce the running costs by out sourcing the services to the Cloud Providers.
- Virtualize the data and equipment.
- Employ public key infrastructure for encryption of data communications between system components and communications with end-users.

5 CONCLUSION

This document describes the core services that support the implementation and piloting of the BEACONING Platform and Ecosystem. The core services streamline the interactions across the BEACONING components, providing a unified experience at user level.

5.1 RESULTS

The work presented herein describes the main functionalities that support the operation of the BEACONING Platform and Ecosystem. It features an easy to manage user administration facility, and it describes private data management via the PDS.

5.2 IMPACT

The services described in this deliverable provide the backend functionalities of the BEACONING Platform and Ecosystem. The core services provide the user authentication and management. They can be called by the main components of the platform (GPE, Authoring System, gamified lesson paths) to perform the administration of BEACONING assets, minigames, and plots.

A revised version of this deliverable will be produced in M28 after the beta version of the platform has been delivered based on the outcomes of testing and piloting tasks.

6 REFERENCES

1. Bihis, C. (2015). Mastering OAuth 2.0. Packt Publishing
2. Psygkas, A., Akrivopoulou, C. 2010. Personal Data Privacy and Protection in a Surveillance Era, IGI Global.
3. REGULATION (EU) 2016/679 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC

BEACONING Core Services

APIs for BEACONING backend services
Version: 1.0.0
BasePath:/api

OAuth Access

1. Authorization URL: https://core.beaconing.eu/auth/auth
2. Token URL: https://core.beaconing.eu/auth/token

Methods

[Jump to Models]

Table of Contents

Assets

- POST /assets
- DELETE /assets/{id}
- GET /assets/{id}
- GET /assets
- PUT /assets/{id}

Gameplots

- POST /gameplots
- DELETE /gameplots/{id}
- GET /gameplots/{id}
- GET /gameplots
- PUT /gameplots/{id}

Gamifiedlessonpaths

- POST /gamifiedlessonpaths
- DELETE /gamifiedlessonpaths/{id}
- GET /gamifiedlessonpaths/{id}
- GET /gamifiedlessonpaths
- PUT /gamifiedlessonpaths/{id}

Minigames

- POST /minigames
- DELETE /minigames/{id}
- GET /minigames/{id}
- GET /minigames
- PUT /minigames/{id}

Studentgroups

- POST /studentgroups
- DELETE /studentgroups/{id}
- GET /studentgroups/{id}
- GET /studentgroups
- PUT /studentgroups/{id}

Students

- POST /students
- DELETE /students/{id}
- GET /students/{id}
- GET /students
- PUT /students/{id}

Assets

POST /assets

Create asset (**createAsset**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body Asset (required)
Body Parameter —

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
201
Success
400
Invalid input

DELETE /assets/{id}

Delete asset (**deleteAsset**)

Path parameters

id (required)
Path Parameter — The id of the asset

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success
404

Asset not found

GET /assets/{id}

Get asset details (**getAsset**)

Path parameters

id (required)
Path Parameter — The id of the asset

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type
Asset

Example data
Content-Type: application/json

```
{
  "author" : "aeiou",
  "name" : "aeiou",
  "description" : "aeiou",
  "id" : "aeiou",
  "contentType" : "aeiou",
  "content" : "aeiou"
}
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success Asset
404
Asset not found

GET /assets

Retrieve the list of assets (**listAssets**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type
array[Asset]

Example data
Content-Type: application/json

```
[ {
  "author" : "aeiou",
  "name" : "aeiou",
  "description" : "aeiou",
  "id" : "aeiou",
  "contentType" : "aeiou",
  "content" : "aeiou"
} ]
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success

PUT /assets/{id}

Update asset (**updateAsset**)

Path parameters

id (required)
Path Parameter — The id of the asset

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body Asset (required)
Body Parameter — Updated asset object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success
400
Invalid input
404
Asset not found

Gameplots

POST /gameplots

Create game plot (**createGamePlot**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body [GamePlot](#) **(required)**
Body Parameter —

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
201
Success
400
Invalid input

DELETE /gameplots/{id}

Delete game plot (**deleteGamePlot**)

Path parameters

id (required)
Path Parameter — The id of the game plot

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success
404
Game plot not found

GET /gameplots/{id}

Get game plot details (**getGamePlot**)

Path parameters

id (required)
Path Parameter — The id of the game plot

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type
GamePlot

Example data
Content-Type: application/json

```
{
  "author" : "aeiou",
  "name" : "aeiou",
  "description" : "aeiou",
  "id" : "aeiou",
  "content" : "aeiou"
}
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success GamePlot
404
Game plot not found

GET /gameplots

Retrieve the list of game plots (**listGamePlots**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type
array[GamePlot]

Example data
Content-Type: application/json

```
[ {
  "author" : "aeiou",
  "name" : "aeiou",
  "description" : "aeiou",
  "id" : "aeiou",
  "content" : "aeiou"
} ]
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200

Success

PUT /gameplots/{id}

Update game plot (**updateGamePlot**)

Path parameters

id (required)
Path Parameter — The id of the game plot

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body GamePlot (required)
Body Parameter — Updated game plot object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success
400
Invalid input
404
Game plot not found

Gamifiedlessonpaths

POST /gamifiedlessonpaths

Create gamified lesson path (**createGamifiedLessonPath**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body GamifiedLessonPath (required)
Body Parameter —

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

201
Success
400
Invalid input

DELETE /gamifiedlessonpaths/{id}

Delete gamified lesson path (**deleteGamifiedLessonPath**)

Path parameters

id (required)
Path Parameter — The id of the gamified lesson path

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200
Success
404
Gamified lesson path not found

GET /gamifiedlessonpaths/{id}

Get gamified lesson path details (**getGamifiedLessonPath**)

Path parameters

id (required)
Path Parameter — The id of the gamified lesson path

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type

GamifiedLessonPath

Example data

Content-Type: application/json

```
{
  "author" : "aeiou",
  "name" : "aeiou",
  "description" : "aeiou",
  "id" : "aeiou",
  "content" : "aeiou"
}
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success GamifiedLessonPath
404
Gamified lesson path not found

GET /gamifiedlessonpaths

Retrieve the list of gamified lesson paths (**listGamifiedLessonPaths**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type
array[GamifiedLessonPath]

Example data
Content-Type: application/json

```
[ {
  "author" : "aeiou",
  "name" : "aeiou",
  "description" : "aeiou",
  "id" : "aeiou",
  "content" : "aeiou"
} ]
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success

PUT /gamifiedlessonpaths/{id}

Update gamified lesson path (**updateGamifiedLessonPath**)

Path parameters

id (required)
Path Parameter — The id of the gamified lesson path

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body GamifiedLessonPath **(required)**
Body Parameter — Updated gamified lesson path object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success
400
Invalid input
404
Gamified lesson path not found

Minigames

POST /minigames

Create minigame (**createMinigame**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body Minigame **(required)**
Body Parameter —

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
201
Success
400
Invalid input

DELETE /minigames/{id}

Delete minigame (**deleteMinigame**)

Path parameters

id (required)
Path Parameter — The id of the minigame

Consumes
This API call consumes the following media types via the Content-Type request header:

- application/json

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses
200
Success
404
Minigame not found

GET /minigames/{id}
Get minigame details (**getMinigame**)

Path parameters

id (required)
Path Parameter — The id of the minigame

Consumes
This API call consumes the following media types via the Content-Type request header:

- application/json

Return type
Minigame

Example data
Content-Type: application/json

```
{
  "thumbnail" : {
    "contentType" : "aeiou",
    "content" : "aeiou"
  },
  "lookupResourcesUrl" : "aeiou",
  "author" : "aeiou",
  "name" : "aeiou",
  "description" : "aeiou",
  "id" : "aeiou",
  "runtimeUrl" : "aeiou",
  "schemaUrl" : "aeiou"
}
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses
200
Success Minigame
404
Minigame not found

GET /minigames

Retrieve the list of minigames (**listMinigames**)

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type

array[Minigame]

Example data

Content-Type: application/json


```
[ {
  "thumbnail" : {
    "contentType" : "aeiou",
    "content" : "aeiou"
  },
  "lookupResourcesUrl" : "aeiou",
  "author" : "aeiou",
  "name" : "aeiou",
  "description" : "aeiou",
  "id" : "aeiou",
  "runtimeUrl" : "aeiou",
  "schemaUrl" : "aeiou"
} ]
```


Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses

200

Success

PUT /minigames/{id}

Update minigame (**updateMinigame**)

Path parameters

id (required)
Path Parameter — The id of the minigame

Consumes

This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body Minigame (required)
Body Parameter — Updated minigame object

Produces

This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success
400
Invalid input
404
Minigame not found

Studentgroups

POST /studentgroups

Create student group (**createStudentGroup**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body StudentGroup **(required)**
Body Parameter —

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
201
Success
400
Invalid input

DELETE /studentgroups/{id}

Delete student group (**deleteStudentGroup**)

Path parameters

id (required)
Path Parameter — The id of the student group

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses
200
Success
404
Student group not found

GET /studentgroups/{id}

Get student group details (**getStudentGroup**)

Path parameters

id (required)
Path Parameter — The id of the student group

Consumes
This API call consumes the following media types via the Content-Type request header:

- application/json

Return type
StudentGroup

Example data
Content-Type: application/json

```
{
  "name" : "aeiou",
  "id" : "aeiou"
}
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- application/json

Responses
200
Success StudentGroup
404
Student group not found

GET /studentgroups

Retrieve the list of student groups (**listStudentGroups**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- application/json

Return type
array[StudentGroup]

Example data
Content-Type: application/json

```
[ {
  "name" : "aeiou",
  "id" : "aeiou"
} ]
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success

PUT /studentgroups/{id}

Update student group (**updateStudentGroup**)

Path parameters

id (required)
Path Parameter — The id of the student group

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body StudentGroup **(required)**
Body Parameter — Updated student group object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success
400
Invalid input
404
Student group not found

Students

POST /students

Create student (**createStudent**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body Student **(required)**
Body Parameter —

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
201
Success
400
Invalid input

DELETE /students/{id}

Delete student (**deleteStudent**)

Path parameters

id (required)
Path Parameter — The id of the student

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success
404
Student not found

GET /students/{id}

Get student details (**getStudent**)

Path parameters

id (required)
Path Parameter — The id of the student

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type

Student

Example data
Content-Type: application/json

```
{
  "profile" : "{}",
  "studentGroupId" : "aeiou",
  "id" : "aeiou",
  "username" : "aeiou"
}
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success Student
404
Student not found

GET /students

Retrieve the list of students (**listStudents**)

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Return type
array[Student]

Example data
Content-Type: application/json

```
[ {
  "profile" : "{}",
  "studentGroupId" : "aeiou",
  "id" : "aeiou",
  "username" : "aeiou"
} ]
```

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success

PUT /students/{id}

Update student (**updateStudent**)

Path parameters

id (required)
Path Parameter — The id of the student

Consumes
This API call consumes the following media types via the Content-Type request header:

- `application/json`

Request body

body Student (required)
Body Parameter — Updated student object

Produces
This API call produces the following media types according to the Accept request header; the media type will be conveyed by the Content-Type response header.

- `application/json`

Responses
200
Success
400
Invalid input
404
Student not found

Models

[Jump to Methods]

Table of Contents

1. Asset
2. GamePlot
3. GamifiedLessonPath
4. Minigame
5. Minigame_thumbnail
6. Student
7. StudentGroup

Asset

id
String

name
String

description (optional)
String

author (optional)
String

content (optional)
String format: base64

contentType (optional)
String

GamePlot

id
String

name
String

description (optional)
String

author (optional)
String

content (optional)
String format: base64

GamifiedLessonPath

id
String

name
String

description (optional)
String

author (optional)
String

content (optional)
String format: base64

Minigame

id
String

name
String

description (optional)
String

author (optional)
String

schemaUrl
String

lookupResourcesUrl
String

runtimeUrl
String

thumbnail (optional)
Minigame_thumbnail

Minigame_thumbnail

content (optional)
String format: base64

contentType (optional)
String

Student

id
String

username
String

studentGroupId (optional)
String

profile (optional)
Object

StudentGroup

id
String

name
String