Web Applications, A.Y. 2020/2021
Master Degree in Computer Engineering
Master Degree in ICT for Internet and Multimedia

Homework 1 – Server-side Design and Development

Submission date: 23 April 2021

Last Name	First Name	Student Number
Alberton	Stefano	1237336
Alecci	Marco	2013384
Ezeobi	Francisca Chidubem	1167692
Martinelli	Luca	2005837
Ziroldo	Elia	2019297

Objectives

The objective of the project, in continuation with the one developed during the database course, is to develop a web application called HyperU. This app lets users share project ideas and create teams based on required skills to realize the project. The users can customize their profile, select their skills and can post ideas on the main page of the application. Each post can be liked and commented by other users that can ask to work on the project. The author of the post can create a team, and each one of them will have its own page to discuss the project and share contents and links.

Main functionalities

The website is substantially divided into three main areas:

- **Home page:** available to all the registered users. This page contains all the ideas post created by the users and here the users can do all the main operations like:
  - *Create a new idea post* including the title of the idea, an image, a small description, the list of the skills required to build the project and the topics related to the idea.
  - Like and comment ideas created by others
  - Ask to join a project if they are interested in it
  - See all the join requests that have been sent to the user that is logged in that moment and accepting/declining them.

    - See all the team of which the logged user is part.
    - *Search for ideas* by title, topic, or skills required
    - *Search user* by username or name

- **Profile Page:** contains all the personal information about a user, the skill he owns, the topics he's following and also the ideas he has shared. In this page the user can:
    - *Edit all his personal information* like: name, surname, username, gender, profile picture, biography and a profile picture.
    - *Edit the list of the owned skills*, adding them from a list and selecting a level between 1 and 5.
    - *Edit the list of the followed topics*, adding them from a list. Doing this the application will show in the main page only the ideas that each user would like

- **Team Chat Page:** where the members of a team can discuss the project and share links, media and other types of file. When an idea is created, also a team is created with only the creator as a member.

There are three roles for the user that use this application:

- **Admin:** are the most powerful users in fact they have the same privileges of moderators but they can also choose the moderators or downgrade them to common users if they aren't doing their job properly.

- **Moderator:** guarantee the correct functioning of the application.
    - *They can ban common users* that have posted inappropriate comments or an inappropriate idea. When a user is banned his comments and his ideas are hidden.
    - *They can add new skills and topics* to the lists of selectable ones.

- **Common user:** don't have any privileges. If a user wants to add on his personal page a skill that isn't on the list, they can request the moderators to add it to the list. Same for topics.

Both administrators and moderators have special web pages to do their job. Roles can be sorted according to the privileges they grant. In particular, admin > moderator > common users. This means that a user can access all the areas of the web-site granted by their and lower roles.
The site has an accessory area which allows every user to either register or login to the application.
They have to sign up to the application giving name, surname, birthdate and email. During the registration process they have to choose a unique username and a password that they will use to log in.

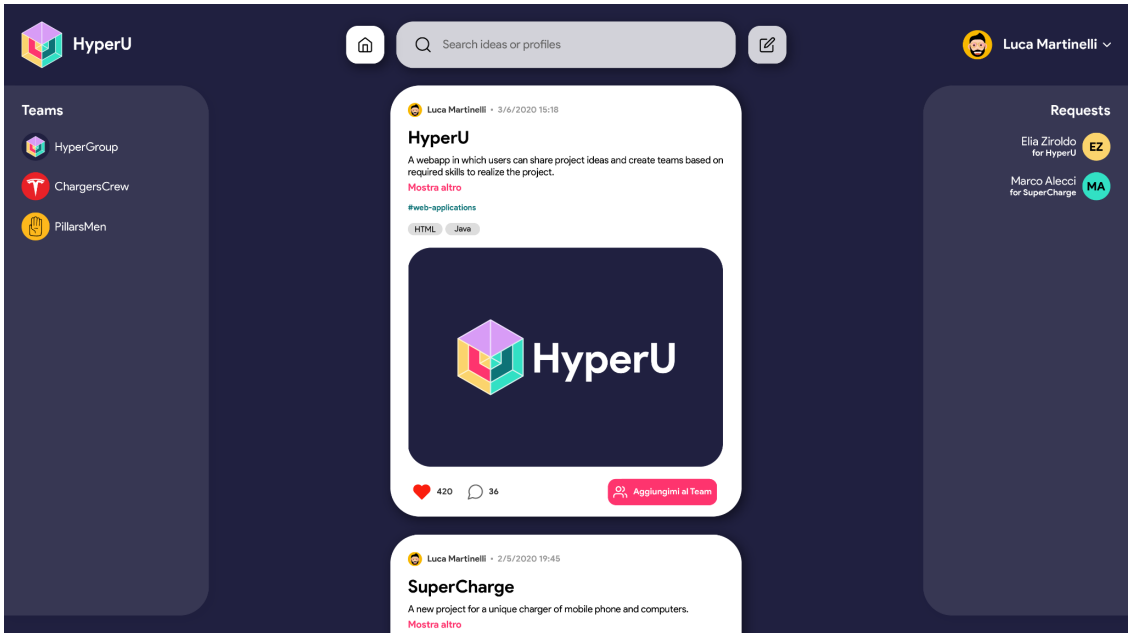
Presentation Logic Layer

The project is thought to be divided into the following pages and they are all written in written in HTML with the support of javascript:

- *Homepage:* contains all the ideas created by the users and lets the users do all the essential tasks.

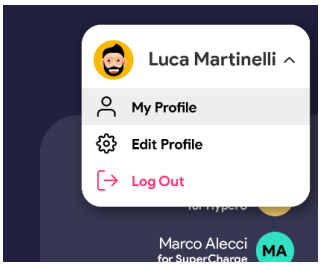
- *Search page:* display all the results obtained by a search done in the homepage
- *Topic page:* display the name and the description of a specific topic and all the ideas related to that topic.
- *Skill page:* display the name and the description of a specific skill and all the ideas related to that require that skill.
- *Login page:* allows the users login to the web application
- *Registration page:* allows the users to register to the web application
- *Profile page:* display all the information about a user and can be used to edit this information.
- *Team Chat:* display some information about a team and let users chat between each other.
- *Admin page:* allows the admins to manage the moderators.
- *Moderator page*: allows the moderators to ban users and to add new skills/topics.

Home Page (Interface Mockup)

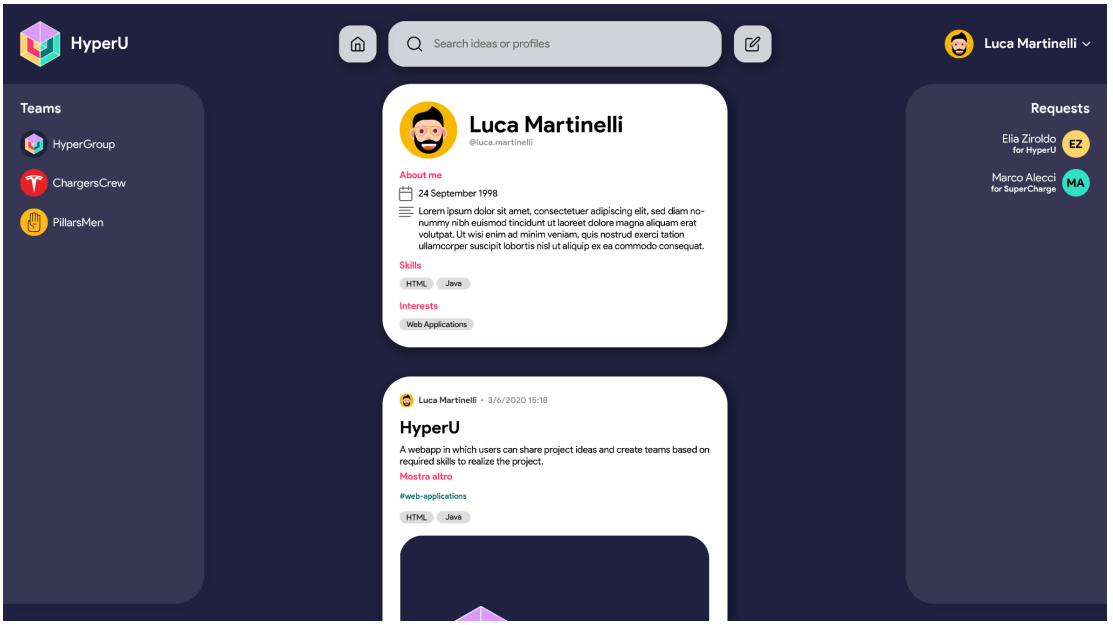


The homepage can be divided in 3 parts. On the left, apart from the logo of the web application, there's the section that lists the teams that the user is currently part of. On the right, there's a section dedicated to the received requests sent by other users that would like to join a specific team created by the logged user. Actually these two sections are always present wherever the user navigates through the web application. In the main area of the homepage the user can scroll past posts about topics he follows, if there are any. Otherwise, the posts aren't filtered by topics and they are listed in chronological order. The user can obviously like and comment posts. These actions can be undone. Each post is about an idea of a project. Each post includes the title of the idea, an image (like a small presentation or a schema of the project), a small description and a list of required skills. The hashtags are the related topics upon which the post has been classified on. The user in order to take part in the project can send a join request by simply clicking on the button at the bottom right of the post. If the

request is accepted, instead of the join-request button, it will appear a writing that lets the user know that he's part of the project's group. On top of the page there's a homepage button, a search bar and a write-post button. After typing into the search bar, it will open a search page where search's result are shown. These results can be either other users' profiles or posts filtered by skills or topics. To notice also that clicking on the username in the top right, it opens a dropdown menu that lets the user access its own profile page, edit it or logout from the web application.

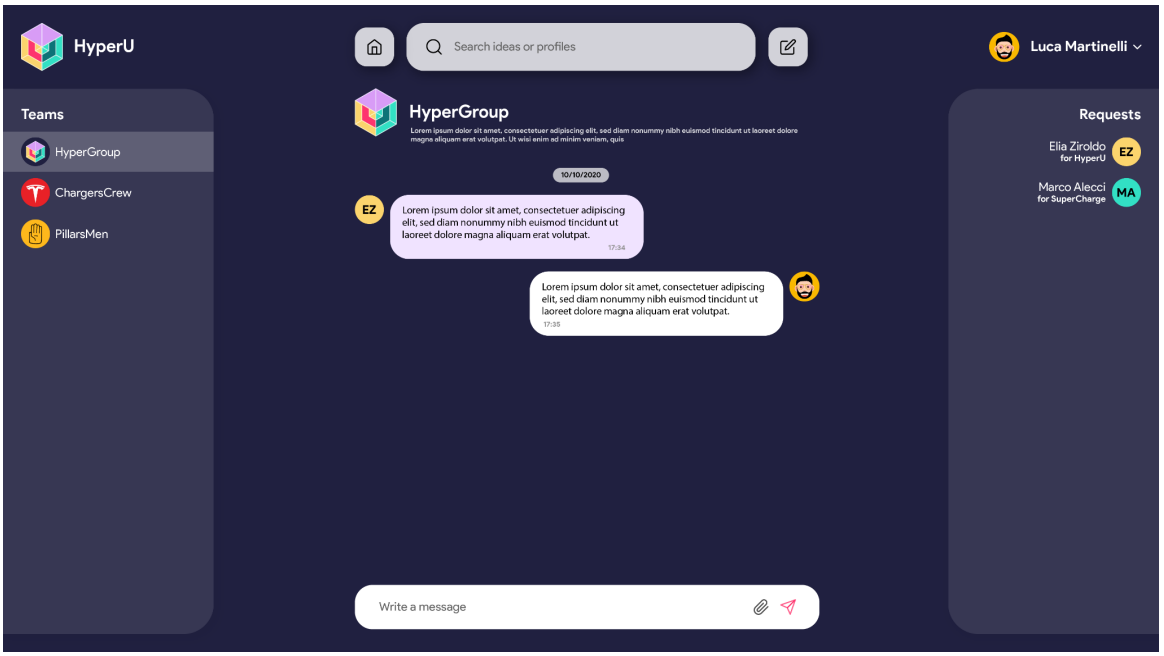


Profile Page (Interface Mockup)



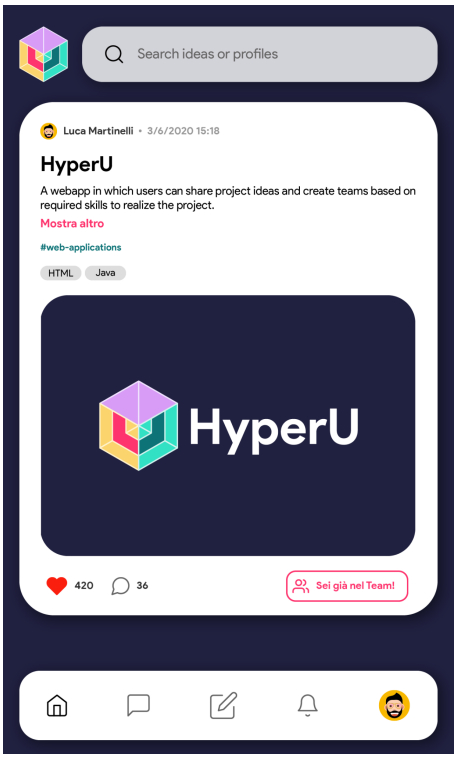
On the profile page, it will be shown the name, surname, username, gender, profile picture, a short biography, a list of skills the user possesses and topics (also called as interests) the user is interested in. The profile photo, gender and biography are optional. The biography supports markdown language which means it's possible for the user to add links, to set text in bold, underline or italic, etc... . In this page, there's also the possibility to see all the posts made by the user himself. The posts can be further edited anytime.

Team Chat Page (Interface Mockup)



A click on a team's name in the section on the left will lead to the selected team's chat page. In this page the user can chat with other group members in order to discuss the project and also share links, media and other types of files. The exchanged messages will appear on the main area of the page. More specifically, messages sent by other team members are aligned to the left meanwhile the ones sent by the logged user are aligned to the right. Alongside every message there's the profile picture of the person who sent it. A text field, in which the user writes the message, is placed at the bottom. Next to it, there are two other buttons: the clip button for attaching any type of files and the paper plane shaped button for sending messages.

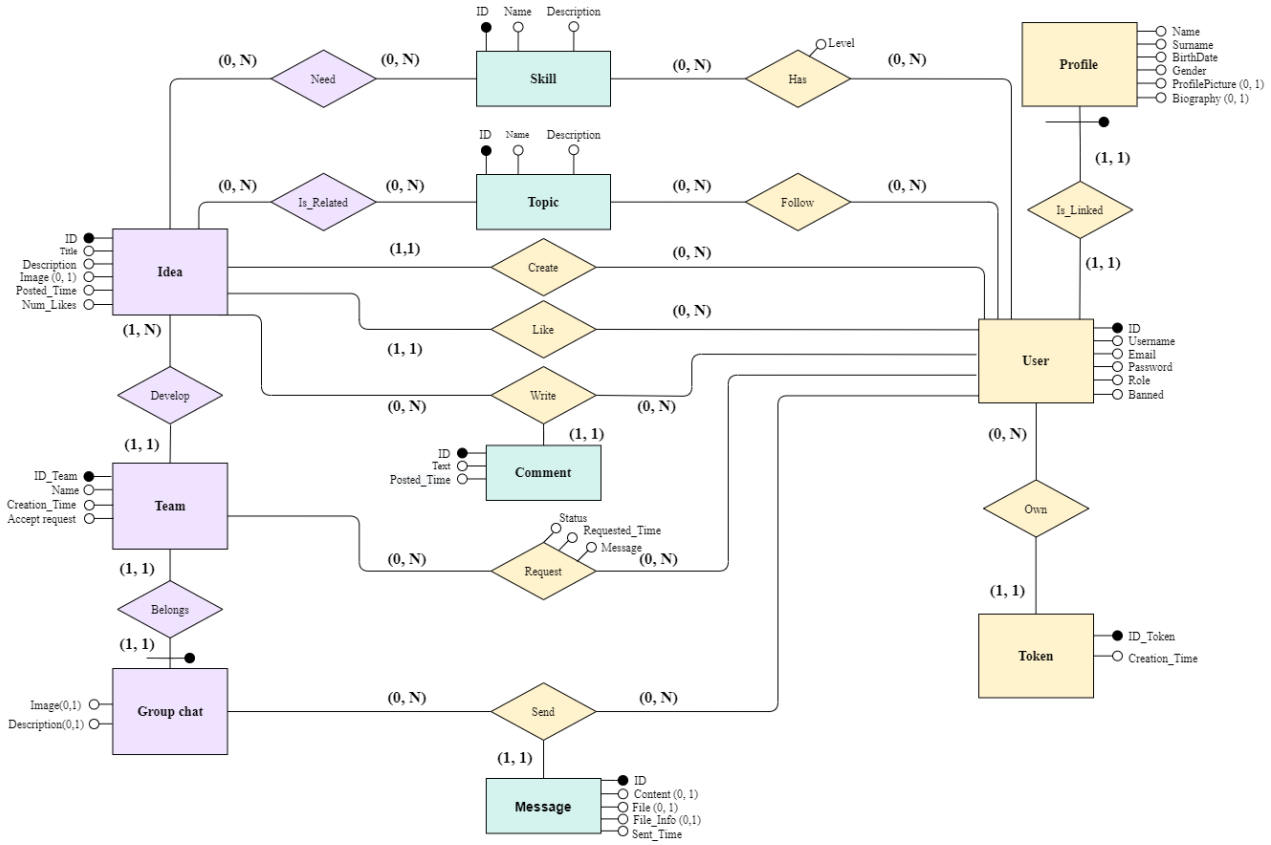
Mobile Home Page (Interface Mockup)



A supposed mobile homepage is obviously configured differently. On top, alongside the web application's logo there's the search bar. While on bottom there's a navigation bar containing 5 buttons: one leads to the homepage, one for the groups' chat pages, one for writing a post, one that leads to a page that notifies the user about the received join requests and finally another one for accessing the profile page.

Data Logic Layer

Entity-Relationship Schema



The entity-relationship contains the following *entities*:

- ***Idea:*** this entity represents the main concept of the app. It's the core of the HyperU web application. An idea is identified by an integer ID and there are five additional attributes: the title, the description, the image (saved as an array of byte), the posted time (saved in timestamp format) and the number of likes to the post. Each idea is linked to the user who has created it by the relationship Create, since the relationship is 1-N this information is recorded directly in the Idea with a foreign key.

- ***User:*** each user that uses HyperU is identified by an ID and he also has to indicate an username and an email. The attribute called role of the user (custom enumeration) determines if a user is a simple user, a moderator or an administrator, meanwhile a boolean variable called banned indicates if the user has been banned or not. To store the user's password in a secure way a salt is added to it and then it's hashed through MD5 before being stored in the database.

- ***Profile:*** to store less frequently used information about a user (like personal ones) we save them in the Profile entity. Since Profile is linked with User with a 1-1 relationship, this entity contains directly the id of the associated user. The userID is also used as the primary key for Profile. There are also 4 mandatory attributes: Name, Surname, BirthDate and gender(custom enumeration) and 2 optional attributes: the profile pictures and the biography.

● ***Skill:*** this entity represents a specific competence and it is uniquely defined by an ID. To better define it there are two attributes: the name of the skill and a small description.

● ***Topic:*** since the idea is often a complex project, it can regard multiple arguments, they are called Topic. This entity is uniquely defined by an ID and contains also the name of the topic and a small description of it.

● ***Comment:*** this entity represents a text message to put under the post of the idea, where for example different users can ask some details about the project to decide if they participate or not. It is uniquely defined with an ID, while the other attributes are the text of the comment and the time at which the comment was posted. Since Comment has a 1-N ternary relationship with Idea and User the IDs of the idea and of the user are saved here as foreign keys.

● ***Team:*** this entity represents the group created for the development of an idea, it can be always modified by adding or removing members. It is uniquely defined by an ID and there is also a name, the time of the creation and a special boolean attribute that defines if other users can send requests to join the team. Each team is linked to the idea with a 1-N relationship so the ID of the Idea is saved as foreign key.

● ***Group Chat:*** this entity is used to represent the private conversation of the group. Since this is a weak entity, it's defined by the ID of the associated team. (In fact each team can have only one group chat).The other attributes are: a group image and a description to be shown in the chat page.

● ***Message:*** this entity represents a message sent in the Group Chat. It is identified with an ID and also the time of when the message was sent is saved. The message can be a simple text message and in this case the attribute content is used. If the message is a file the attributes File and File_Info are used. Since Message has a 1-N ternary relationship with Group Chat and User the IDs of the team and of the user are saved here as foreign keys.

● ***Token***: this entity is used to save UUID associated to users in order to maintain the session of the user with an automatically re-login using cookies.

There are also some *N-N relationship*, that need a table to be represented:

● **Has**: this table links the user (with ID_User) with the skill (with ID_Skill). In fact a user can have different abilities. The level attribute goes from 1 to 5.

● **Follow**: this table has the foreign key to User (ID_User) and to Topic (ID_Topic). Here is registered every topic that a user has followed in order to show ideas that are related with his interests.

● **Request**: this relation represents the request that a user makes to join a team. It links the ID_Idea, with ID_User and the status of the request is represented by an attribute with a custom enumeration. The other attributes are the text message attached to the request, and the time of when the join request was sent. Only the owner of the team can accept or reject the requests.

● **Like**: this relation stores which ideas the users has liked. There are two foreign keys that are ID_Idea and ID_User

• **Need**: this relation stores which skills are needed for realizing an Idea. There are two forein keys: ID_Idea and ID_Skill. It has been decided not to put also the level of skill to avoid confusion, so the owner of the group will evaluate the skill level by looking at the user's profile.

• **Is_Related**: this relation links an idea with the topics to which it is concerned, so a user that follows specific topics can find more interesting ideas. It has two foreign keys: ID_Idea and ID_Topic.

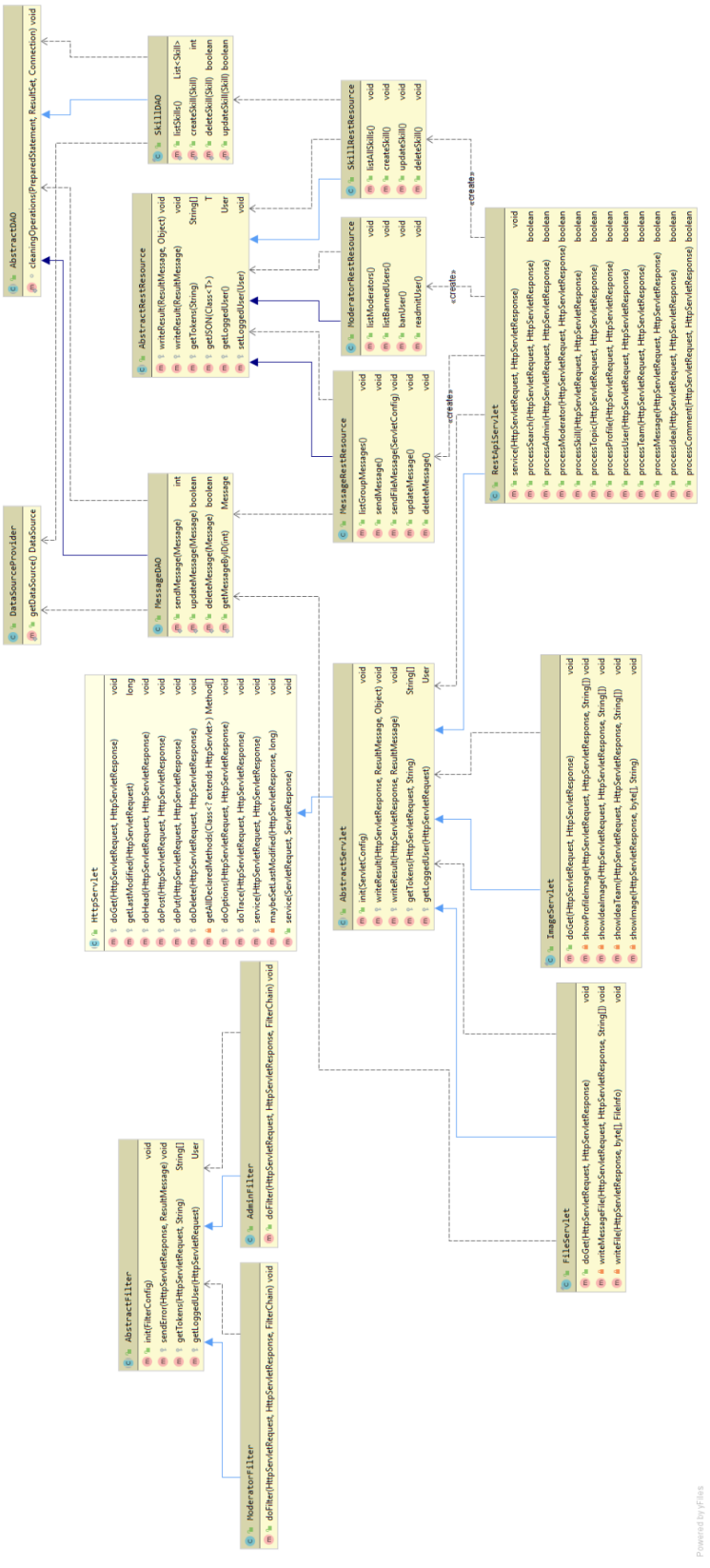
Other Information

There are 3 custom enumerations:

- **Gender*:*** attribute of Profile, which can be male, female or not declared
- **Role**: attribute of User, that represents the powers of the user in the group. It can be an administrator, a moderator or a common user.
- **Status*:*** attribute of the join request sent from a user to enter in a team. The request can be pending or accepted. If a request is denied it's removed from the database.

Business Logic Layer

Class Diagram



The class diagram contains only some of the classes used in the project due to space reasons
It is possible to observe that we have a main servlet, based on the REST paradigm, called RestApiServlet, that parses the URI determining the resource that the user wants to interact with. Once the servlet has processed the request, it forwards it to the proper Rest manager class. There are several Rest manager classes such as the ones we can see in the diagram:

- SkillRestResource: implements methods to interact with skills
- ModeratorRestResource: implements methods which are accessible only to users with the "moderator" authorization level.
- MessageRestResource: implements methods to send and list messages in the team chat

They are all sub-classes of AbstracRestResource that contains some useful methods to write JSON or to tokenize the URI. There are also other Servlets such as FileServlet and ImageServlet to do operations concerning files and media without using the REST paradigm. All the Servlets are sub-classes of AbstractServlet that contain utilities methods like the ones used by AbstractRestResource class.

To prevent unauthorized access to the REST Resources there are differents filter classes such the ones we can see in the diagram:

- ModeratorFilter: to let only moderators to do some operations
- AdminFilter: to let only admins to do some operations

There are many other filters and they are better described in the section "REST Api Summary", but one key filter is the LoginFilter(not shown in diagram) that is applied to almost all the URIs and doesn't allow unregistered users to use HyperU. All the filter classes are sub-classes of AbstractFilter that contain some methods useful to do the filtering operations.

To interact with the database, there is one Data Access Objects (DAO) for each resource in the database. Each DAO implements different methods to do basic operations such as retrieve, insert, modify and delete the different resources, but also methods to do some useful queries to the database. In the diagram we can see for example:

- MessageDAO: to interact with Messages
- SkillDAO: to interact with Skills

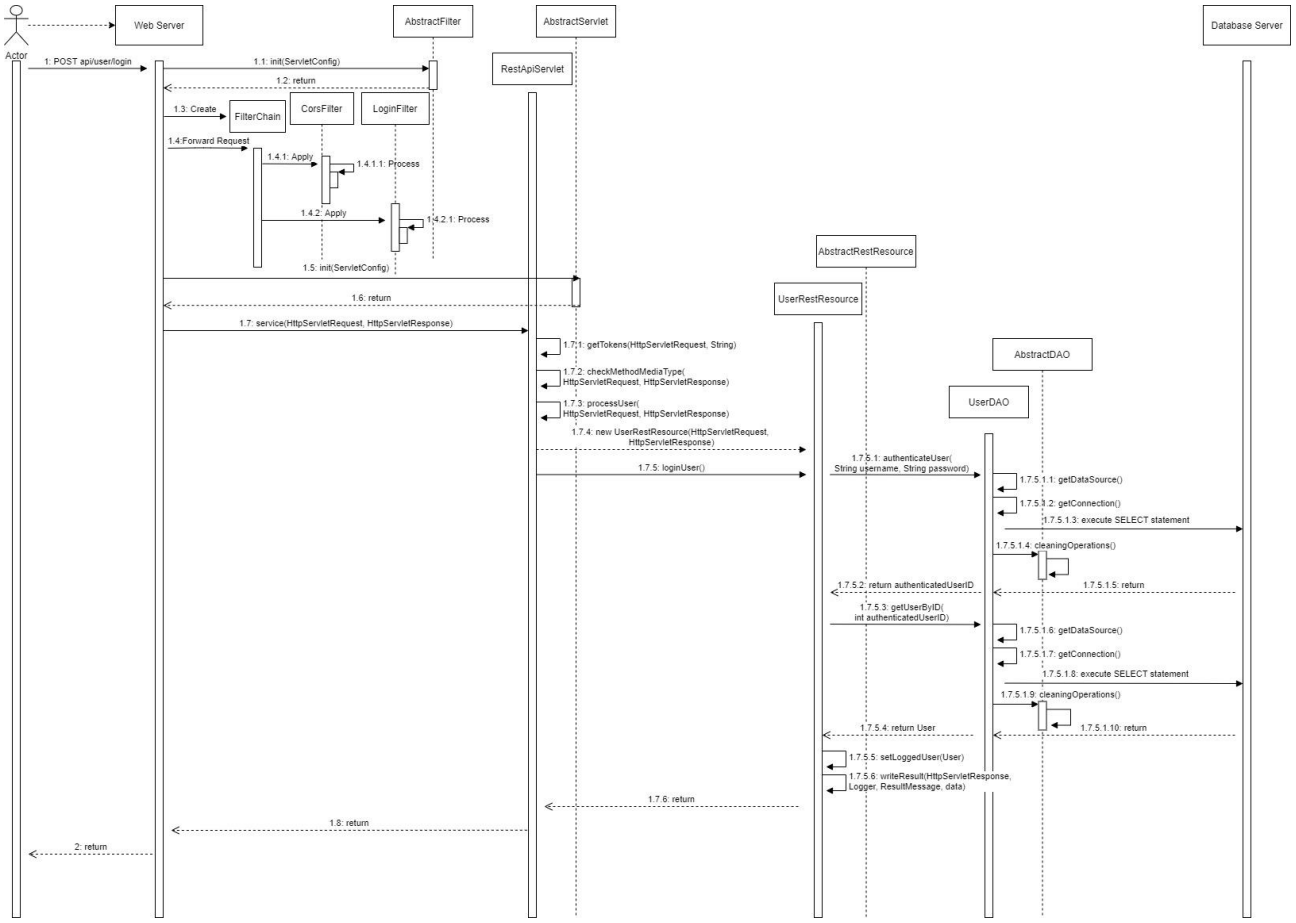
The methods of a DAO that insert some data in the database return the ID of the created row, meanwhile the methods that update/delete rows return a boolean value that is true if the operation was successful. All the DAOs extend the AbstractDAO class and exploit a DataSource object, defined as singleton in the DataSourceProvider class, to obtain the connection to the database. Each resource of the database is mapped to a specific java class but due to space reasons they have been omitted from the diagram. Instances of such classes are instantiated by and passed between Rest manager classes and DAOs.

There are also some util classes. Some of them are for examples:

- PasswordEncryptor: to calculate the MD5 of the passwords adding salt.
- Classes that represent a specific data type such as:
    - StatusType
    - UserType
    - GenderType
- InfoMessage: to manage result messages to display after an operation
- ErrorCode: to manage the different types of errors.

Sequence Diagram

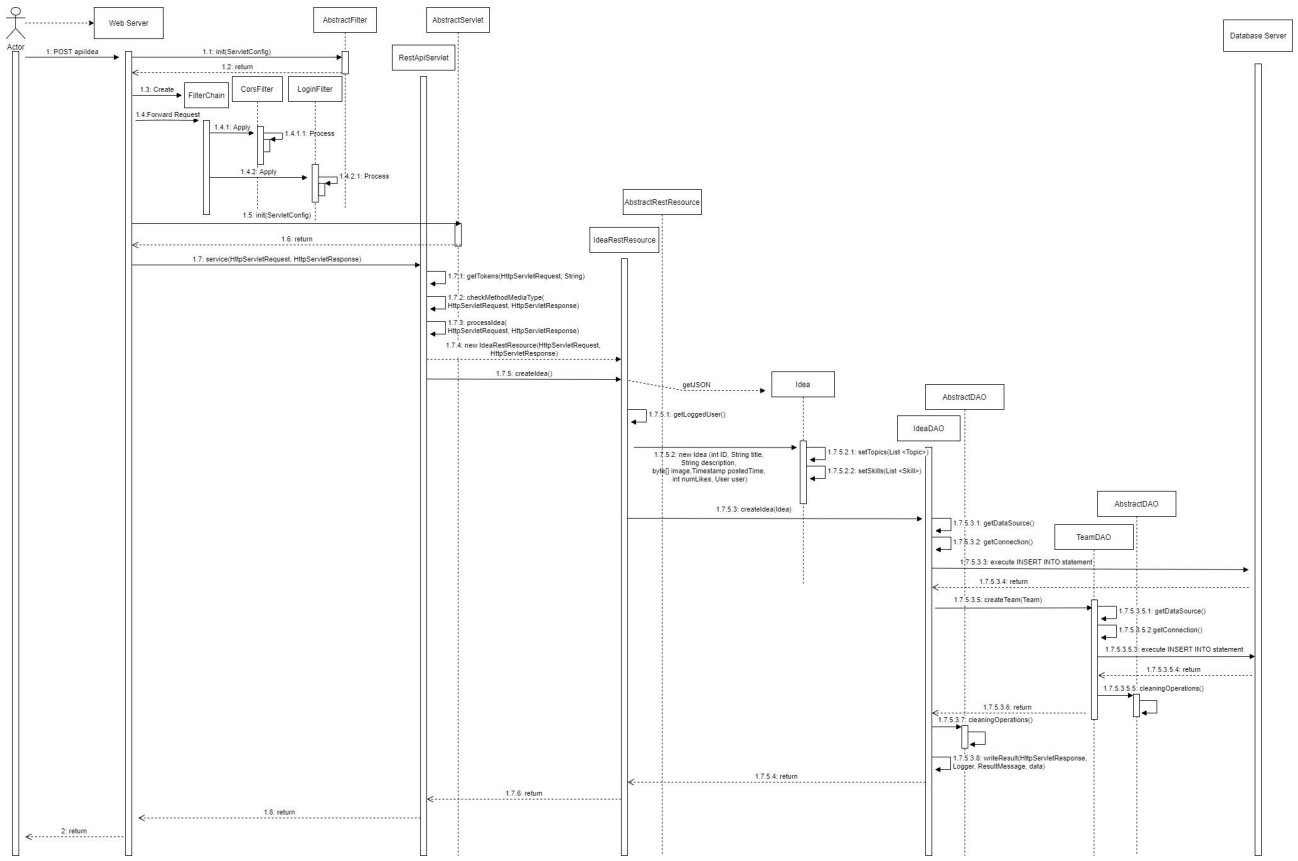
Here we present the sequence diagram of **api/user/login POST** request:



Here reported the sequence diagram for the login operations. The user executes a POST request to the web server, specifying the URI api/user/login/. The credentials of the user are passed as parameters of the POST request. The `LoginFilter` is applied to all the URI of the Rest API, and so it is called. Since the URI corresponds to the one for the login operation the `LoginFilter` calls the `doFilter()` method and lets the workflow continue. The web server instantiates the `RestApiServlet` and calls its `service()` method, passing the `HttpServletRequest` and the HttpServlet response. The `RestApiServlet` analyzes the request and recognizes that it is a

login operation, thus it istanties the class `UserRestResource` and calls its method `loginUser()`. This method controls whether the username and the password have been correctly provided, if not, returns an error. Else, a new `User` object is initialized with the provided data. Such an object is passed as an argument to the `authenticateUser()` method of the class `UserDAO`. Here, a connection is requested to the `DataSourceProvider`, and using this one, the `UserDAO` contacts the Database server and checks whether the user passed as an argument is present in the database . After that, the control is returned to the `RestApiServlet`, which obtains the session from the `HttpServletRequest` and sets the generated `User` object as parameter. It also creates a new Java Web Token with the user information with an expiring time of 3 hours. Finally, a JSON object containing the user information, the java web token and a success message is printed on the page.

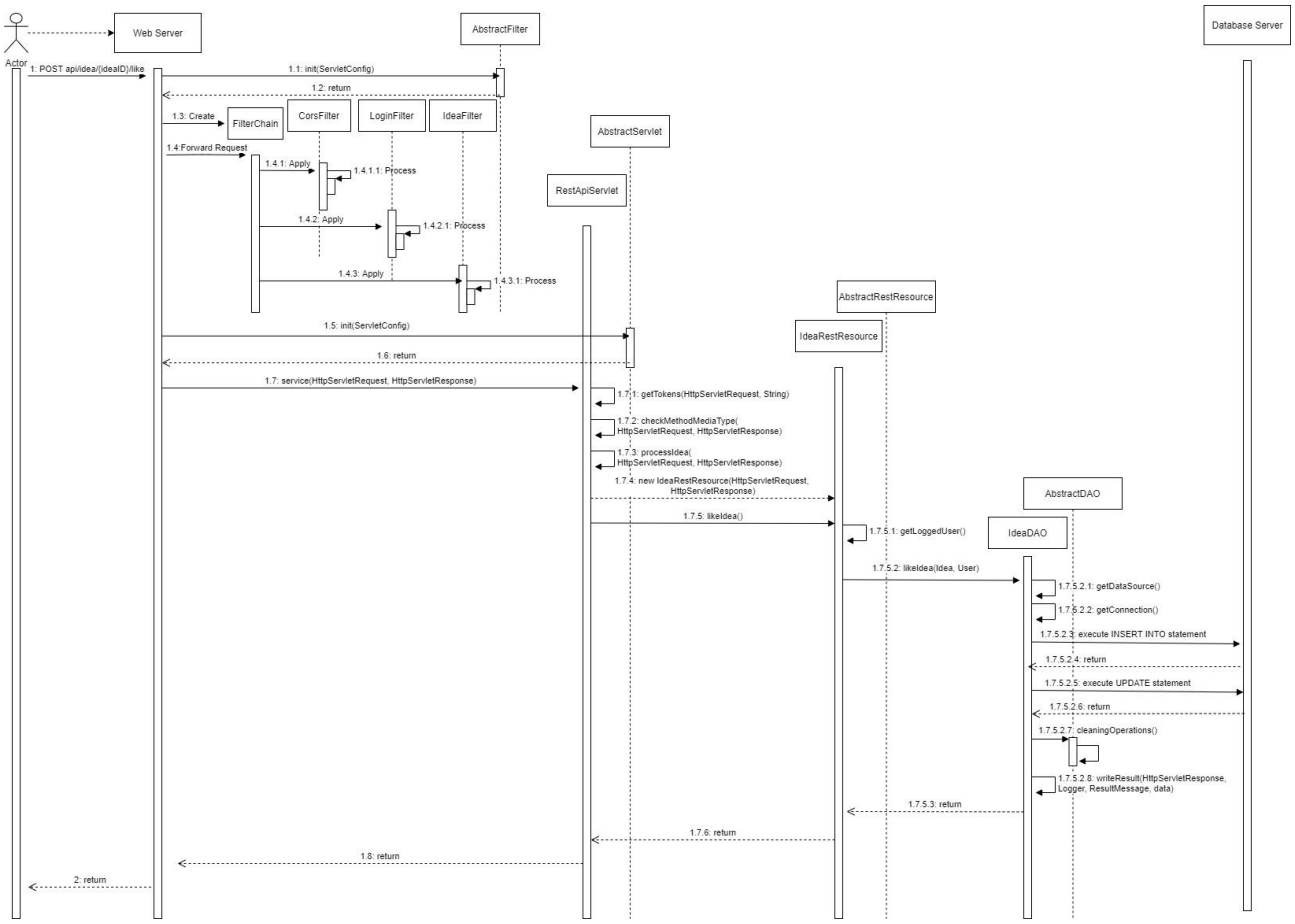
Here instead we have the sequence diagram of **api/idea POST** request:



Here reported the sequence diagram for creating a new idea. When the user clicks on the button to create the idea it executes a POST request to the web server, specifying the URI api/idea. All the information needed to create the idea are passed to the web server as JSON parameters of the request. The web server checks if the one who is trying to create a new idea is a logged user or not by using the `LoginFilter`. After the check the web server istanties the `RestApiServlet` and calls its `service()` method, passing the `HttpServletRequest` and the HttpServlet response. The `RestApiServlet` analyzes the request and recognizes that it is an operation concerning the creation of an Idea, so it istanties the class `IdeaRestResource` and calls its method

`createIdea()`. This method checks that all the fields necessary for the idea creation are not null and not empty, otherwise it returns an error. After the check, a new Idea object is initialized with the provided data and using the `User` object retrieved with the `getLoggedUser()` method. The `Idea` object is then passed as an argument to the `createIdea()` method of the class `IdeaDAO`. Here, a connection is requested to the `DataSourceProvider`, and using this one, the `IdeaDAO` contacts the database server and inserts a new row into the database. After the insertion the method calls the `createTeam()` method of the `TeamDAO` class and with this method the DAO manages the creation of a new team into the database. Finally a result message is returned with attached the id of the just created Idea.

Finally the sequence diagram of the **api/idea/{ideaID}/like** request:



Here reported the sequence diagram to like an idea. When the user clicks on the like button it executes a POST request to the web server, specifying the URI api/idea/{ideaID}/like, with the ideaID directly retrieved from the idea that has been liked. The web server checks if the one who is trying to like an idea is a logged user or not by using the `LoginFilter`. Since the URI starts with "idea", due to the filter mapping also the `IdeaFilter` is applied. Here the filter recognizes a like operation and so it calls the `doFilter()` method. After the filters the web server istanties the `RestApiServlet` and calls its `service()` method, passing the `HttpServletRequest` and the HttpServlet response. The `RestApiServlet` analyzes the request and recognizes that it is a like operation, so it istanties the class `IdeaRestResource` and calls its method `likeIdea()`. This method

instanties a new `Idea` object using the ideaID and retrieves a `User` object representing the logged `User`. These objects are then passed as arguments to the `likeIdea()` method of the class `IdeaDAO`. Here, a connection is requested to the `DataSourceProvider`, and using this one, the `IdeaDAO` contacts the database server and:

- inserts a new row into the database in the Like table.
- update the Num_Likes value in the Idea table

Finally a result message is returned.

REST API Summary

Almost all the endpoints in our web application are developed according to the REST paradigm, with the entirety of the information needed to satisfy the request directly in the URI. Some endpoints need additional information that are passed through URL or as parameters of the request. The parameters are passed in JSON format. Part of the URIs are filtered through different filters.
The full documentation is available at HyperU Documentation.

Filters

- **Login** (**Lgd**): The user is logged in. This filter is applied to all REST URIs, except for login, token and register
- **Admin** (**Adm**): The user is an administrator.
- **Moderator** (**Mdr**): The user is a moderator.
- **Team** (**Tem**): The user is part of the team.
- **Team Creator** (**TmC**): The user is the creator of the idea associated with the team.
- **Message** (**Msg**): The user sent the message. For file download, the user is in the same group where the file was sent.
- **Team opened** (**TeO**): The team accepts new join requests.
- **Idea** (**Ide**): The user is the creator of the idea.
- **Comment** (**Cmt**): The user sent the comment.

URI	Method	Description	Filter
api/user/register	POST	Allow to register a new user to HyperU	
api/user/login	POST	Login operation with user's credential	
api/user/token	POST	Login operation with userID and token (to use cookies for stay logged)	
api/user/logout	GET	Allow the user to logout from HyperU	
api/user/me	GET	Get the profile of the logged user, with his skills, topics, number of ideas created, number of likes got	
api/user/me	PUT	Update the profile of logged user	
api/user/me/password	POST	Update the password of the logged user. The method used is post because PUT doesn't support parameters passed through application/x-www-form-urlencoded,	

		but only query parameters, and for password this could be a security problem	
api/user/me/image	PUT	Allow to update the profile picture of the logged user	
api/user/me/image	DELETE	Allow to remove the profile picture of the logged user	
api/user/me/ideas	GET	Get a list of all the ideas created by the logged user	
api/user/me/requests	GET	Get a list of the join requests that has been sent to the logged user	
api/user/me/teams	GET	Get a list of the teams (and groups) of which the logged user is a member	
api/user/me/feed	GET	Get a list of idea personalized for logged user (based on the topics that he's following)	
api/user/me/skill/{skillID}	POST	Allow a user to add a new skill into his personal page	
api/user/me/skill/{skillID}	PUT	Allow a user to update the level of one of his skill	
api/user/me/skill/{skillID}	DELETE	Allow a user to delete a skill from his personal page	
api/user/me/topic/{topicID}	POST	Allow a user to follow a new topic	
api/user/me/topic/{topicID}	DELETE	Allow a user to unfollow a topic	
api/profile/{userID}	GET	Get the profile of the user, with his skills, topics, number of ideas created, number of likes got. (using userID)	
api/profile/{username}	GET	Get the profile of the user, with his skills, topics, number of ideas created, number of likes got. (using username)	
api/profile/{userID}/ideas	GET	Get a list of all the ideas created by the user	
api/admin	GET	Get a list of all the administrators	
api/admin/moderator/{userID}	POST	Allow to promote a user to moderator	Adm
api/admin/moderator/{userID}	DELETE	Allow to downgrade a moderator to user	Adm
api/moderator	GET	Get a list of all the moderators	
api/moderator/ban	GET	Get a list of all the banned users	Mdr
api/moderator/ban/{userID}	PUT	Allow to ban a user	Mdr
api/moderator/unban/{userID}	PUT	Allow to unban a user	Mdr
api/moderator/skill	POST	Allow to insert a new skill to the database	Mdr
api/moderator/skill/{skillID}	PUT	Allow to edit the information of a skill	Mdr
api/moderator/skill/{skillID}	DELETE	Allow to delete a skill from the database	Mdr
api/moderator/topic	POST	Allow to Insert a new topic to the database	Mdr
api/moderator/topic/{topicID}	PUT	Allow to edit the information of a topic	Mdr
api/moderator/topic/{topicID}	DELETE	Allow to delete a topic from the database	Mdr
api/skill	GET	Get a list of all the skills that are in the database	
api/skill/{skillID}	GET	Get a list of all the ideas related to a skill	
api/topic	GET	Get a list of all the topics that are in the database	
api/topic/{topicID}	GET	Get a list of all the ideas related to a topic	
api/idea	POST	Allow to post a new idea	
api/idea/{ideaID}	GET	Get an idea with skills, topics and teams related to it	
api/idea/{ideaID}	PUT	Allow to update an idea	Ide
api/idea/{ideaID}	DELETE	Allow to delete an idea	Ide

api/idea/{ideaID}/image	PUT	Allow to update the image associated to an idea	Ide
api/idea/{ideaID}/image	DELETE	Allow to delete the image associated to an idea	Ide
api/idea/{ideaID}/team	POST	Allow to create a team for the idea (the first team is created automatically)	Ide
api/idea/{ideaID}/skill/{skillID}	POST	Allow to insert a new skill to an idea	Ide
api/idea/{ideaID/skill/{skillID}	DELETE	Allow to delete a skill from an idea	Ide
api/idea/{ideaID/topic/{topicID}	POST	Allow to insert a new topic to an idea	Ide
api/idea/{ideaID}/topic/{topicID}	DELETE	Allow to delete a topic from an idea	Ide
api/idea/{ideaID}/like	POST	Allow to like an idea	
api/idea/{ideaID}/like	DELETE	Allow to dislike an idea	
api/idea/{ideaID}/comment	GET	Get a list of the comment of an idea	
api/idea/{ideaID}/comment	POST	Allow to comment an idea	
api/comment/{commentID}	DELETE	Allow to delete a comment	Cmt
api/comment/{commentID}	PUT	Allow to edit a comment	Cmt
api/team/{teamID}	GET	Get the info of the Team and the Group	
api/team/{teamID}	PUT	Allow to edit the team and group page	TmC
api/team/{teamID}	DELETE	Allow to delete a team and group	TmC
api/team/{teamID}/image	PUT	Allow to update the image of the team	TmC
api/team/{teamID}/image	DELETE	Allow to delete the image of the team	TmC
api/team/{teamID}/member	GET	Get a list of the members of a group	TeM
api/team/{teamID}/member/{userID}	DELETE	Allow to remove a user from the group	TmC
api/team/{teamID}/request	GET	Get a list of all the join requests sent to a team	TmC
api/team/{teamID}/request	POST	Allow to send a join request to a team	TeO
api/team/{teamID}/request/{userID}	PUT	Allow to accept a join request	TmC
api/team/{teamID}/request/{userID}	DELETE	Allow to decline a join request	TmC
api/team/{teamID}/message	GET	Get a list of the messages of a group	TeM
api/team/{teamID}/message	POST	Allow to send a message into the group chat	TeM
api/message/{messageID}	PUT	Allow to edit a message (not for files)	Msg
api/message/{messageID}	DELETE	Allow to delete a message from the group chat	Msg
api/search/users	GET	Allow to search for users	
api/search/ideas	GET	Allow to search for ideas	
api/image/profile/{userID}	GET	Get the profile image of user	
api/image/idea/{ideaID}	GET	Get the image associated to an idea	
api/image/team/{teamID}	GET	Get the image of a team	
api/file/message/{messageID}	GET	Get the file of a message	Msg

REST API Error Codes

Here the list of errors defined in the application. Application specific errors have the application error which follows a progressive numeration starting from -100.

- **-100** for general errors
- **-200** for errors related to users
- **-300** for query missing
- **-400** for errors related to skills
- **-500** for errors related to topics
- **-600** for errors related to ideas
- **-700** for errors related to comments
- **-800** for errors related to teams
- **-900** for errors related to messages
- **-1000** for errors related to join requests

Error Code	Error Name	HTTP Status Code	Description
-100	CANNOT_ACCESS_DATABASE	INTERNAL_SERVER_ERROR	Cannot access to the database, problems with SQL code or constraints violations
-101	GENERAL_ERROR	INTERNAL_SERVER_ERROR	A general error has occurred. This error at the moment is never used
-102	OPERATION_UNKWOWN	NOT_FOUND	The user is requesting an operation not specified in the rest api summary
-103	BAD_INPUT	BAD_REQUEST	The data passed in input (in the url path or in the body) has missing values or are not in the expected format. Possibile examples are a not number id or a malformed JSON
-104	NO_ELEMENT_ADDED	NOT_MODIFIED	No elements has been inserted in the database
-105	NO_CHANGES	NOT_MODIFIED	No changes has been done in the database
-106	FILE_TOO_LARGE	BAD_REQUEST	File is too large (> 200MB)
-107	IMAGE_TOO_LARGE	BAD_REQUEST	Image is too large (> 5MB)
-108	NOT_IMAGE	UNSUPPORTED_MEDIA_TYPE	Uploaded file is not an image (only png and jpeg accepted)
-109	CANNOT_UPLOAD_FILE	INTERNAL_SERVER_ERROR	Problems while uploading the file
-110	USER_NOT_LOGGED	METHOD_NOT_ALLOWED	An unregistered user is trying to use the web application
-111	USER_NOT_AUTHORIZED	METHOD_NOT_ALLOWED	The user has not the privileges to access that specific page. Examples could be a non admin that tries to access to the admin page or a user is trying to manage an idea for which he isn't the creator
-200	USERNAME_MISSING	BAD_REQUEST	Username is missing during login/registration/update phase

-201	PASSWORD_MISSING	BAD_REQUEST	Password is missing during login/registration/update phase
-202	WRONG_CREDENTIALS	BAD_REQUEST	User credentials are wrong
-203	EMAIL_MISSING	BAD_REQUEST	Email address is missing during registration/update phase
-204	PASSWORD_CHECK_MISSING	BAD_REQUEST	Password checker is missing during registration/update phase
-205	DIFFERENT_PASSWORDS	CONFLICT	Different passwords when repeating the password in the registration/update phase.
-206	NAME_MISSING	BAD_REQUEST	Name is missing during registration/update phase
-207	SURNAME_MISSING	BAD_REQUEST	Surname is missing during registration/update phase
-208	BIRTHDATE_MISSING	BAD_REQUEST	Birthdate is missing during registration/update phase
-209	AGE_NOT_ACCEPTABLE	NOT_ACCEPTABLE	The user must be at least 16 years old while registering or updating his profile
-210	GENDER_MISSING	BAD_REQUEST	Gender is missing during registration/update phase
-211	DUPLICATE_USER	NOT_ACCEPTABLE	Username is already taken when trying to register od updating the profile
-212	OLD_PASSWORD_MISSING	BAD_REQUEST	Old password is missing during password change
-213	OLD_PASSWORD_WRONG	BAD_REQUEST	Old Password is wrong during password change
-214	DUPLICATE_SKILL_USER	NOT_ACCEPTABLE	User is trying to add a skill that he already owns
-215	DUPLICATE_TOPIC_USER	NOT_ACCEPTABLE	User is trying to follow a topic that he already follows
-216	NO_USER_FOUND	NOT_FOUND	User with specific userID doesn't exist
-300	QUERY_MISSING	BAD_REQUEST	The query used to search for idea or users is missing or empty
-401	SKILL_NAME_MISSING	BAD_REQUEST	Skill Name is missing while adding/updating a skill in the database
-402	SKILL_DESCRIPTION_MISSING	BAD_REQUEST	Skill Description is missing while adding/updating a skill in the database
-500	TOPIC_NAME_MISSING	BAD_REQUEST	Topic Name is missing while adding/updating a topic in the database
-501	TOPIC_DESCRIPTION_MISSING	BAD_REQUEST	Topic Description is missing while adding/updating a topic in the database
-600	IDEA_TITLE_MISSING	BAD_REQUEST	Idea Title is missing while adding/updating an idea in the database
-601	IDEA_DESCRIPTION_MISSING	BAD_REQUEST	Idea Description is missing while adding/updating an idea in the database
-602	DUPLICATE_SKILL_IDEA	NOT_ACCEPTABLE	User is trying to add to an idea a skill that is already linked to it.

-603	DUPLICATE_TOPIC_IDEA	NOT_ACCEPTABLE	User is trying to add to an idea a topic that is already linked to it.
-604	NO_IDEA_FOUND	NOT_FOUND	Idea with specific ideaID doesn't exist
-605	IDEA_ALREADY_LIKED	NOT_ACCEPTABLE	User is trying to like an idea that he has already liked
-700	COMMENT_TEXT_MISSING	BAD_REQUEST	Text is missing while adding/updating a comment in the database
-701	NO_COMMENT_FOUND	NOT_FOUND	Comment with specific commentID doesn't exist
-800	TEAM_NAME_MISSING	BAD_REQUEST	Name is missing while adding/updating a team in the database
-801	NO_TEAM_FOUND	NOT_FOUND	Team with specific teamID doesn't exist
-900	MESSAGE_CONTENT_MISSING	BAD_REQUEST	Text is missing while adding/updating a message in the database
-901	NO_MESSAGE_FOUND	NOT_FOUND	Message with specific messageID doesn't exist
-902	MESSAGE_NOT_FILE	NOT_FOUND	The message the user is trying to download doesn't have a file
-1000	REQUEST_ALREADY_EXISTS	NOT_ACCEPTABLE	User is trying to send a join request to a team for which a request has already been sent
-1001	USER_IS_MEMBER	NOT_ACCEPTABLE	User is trying to send a join request to a team he's already member or is its creator
-1002	TEAM_CLOSED	NOT_ACCEPTABLE	Team doesn't accept new join requests

REST API Details

We report here 3 different resource types that our web application handles during its functioning.
The full documentation is available at HyperU Documentation.

1) *Ideas created by the logged user*

With this request we receive the list of all the ideas created by the logged user.

- **URL**
  api/user/me/ideas
- **Method**
  GET
- **URL Params**
  No parameters are required in the url
- **Data Params**
  No data is passed when requiring this method
- **Success Response**
  The RestApiServlet returns a JSON object with a message field containing a response string and a data field containing the requested information.

  Code: 200
  Content:


```
{
   "message": {
      "message": "Ideas listed.",
      "isError": false
   },
   "data": [
      {
         "id": 1,
         "title": "HyperU",
         "description": "An application to share ideas",
         "imageUrl": "image/idea/1",
         "postedTime": "2020-10-05T13:00:00+0200",
         "numLikes": 6,
         "user": {
            "id": 1,
            "username": "flamingomark",
            "email": "marcoalecci98@gmail.com",
            "role": "ADMINISTRATOR",
            "banned": false,
            "profilePictureUrl": "image/profile/1"
         },
```

```
        "topics": [
          {
            "id": 14,
            "name": "Databases",
                "description": "organized collection of data, stored and accessed  electronically from a computer system"
          },
          {
            "id": 16,
            "name": "Web Application",
                "description": "Computer programs that utilizes web browsers and web technology to perform tasks over the Internet"
          }
        ],
        "skills": [
          {
            "id": 21,
            "name": "Adobe Premiere",
            "description": "Ability to use Adobe Premiere"
          },
          {
            "id": 7,
            "name": "CSS",
            "description": "Knowledge of the CSS language"
          },
          {
            "id": 6,
            "name": "HTML",
            "description": "Knowledge of the HTML language"
          },
          {
            "id": 14,
            "name": "Problem Solving",
            "description": "Analytic and rational approach to problems"
          }
        ],
        "teams": [
          {
            "id": 1,
            "name": "HyperGroup",
            "creationTime": "2020-10-06T17:00:00+0200",
            "acceptRequests": true,
            "imageUrl": "image/team/1",
            "description": "The chat of the HyperGroup!"
          }
        ]
    }
  ]
```

}

- **Error Response**

  USER_NOT_LOGGED

  Code: 405 METHOD_NOT_ALLOWED
  Content:

```
{
   "message": {
      "message": "You must be logged in.",
      "isError": true,
      "errorCode": -110
   }
}
```

  When: An unregistered user try to use HyperU

  CANNOT_ACCESS_DATABASE

  Code: 500 INTERNAL_SERVER_ERROR
  Content:

```
{
   "message": {
      "message": "Cannot access to the database.",
      "isError": true,
      "errorCode": -100
   }
}
```

  When: Database server is unreachable, Errors related to SQL code

2) Insert a comment under an idea

With this request we insert a new comment into the database

- **URL**
  api/idea/{ideaID}/comment

- **Method**
  POST

- **URL Params**
  - ideaID = {string}
    The id of the Idea that the user wants to comment.

- **Data Params**
  - text = {string}
    The content of the comment

- **Success Response**

  The RestApiServlet returns a JSON object with a message field containing a response string and a data field containing the id of the inserted comment.

  Code: 200
  Content:

```
{
   "message": {
      "message": "Comment sent.",
      "isError": false
   },
   "data": 17
}
```

- **Error Response**

  USER_NOT_LOGGED

  Code: 405 METHOD_NOT_ALLOWED
  Content:

```
    {
       "message": {
          "message": "You must be logged in.",
          "isError": true,
          "errorCode": -110
       }
    }
```

  When: An unregistered user try to use HyperU

  COMMENT_TEXT_MISSING

  Code: 400 BAD_REQUEST
  Content:

```
{
   "message": {
      "message": "Text is missing.",
      "errorCode": -700,
      "isError": true
   }
}
```

  When: The user is trying to send an empty comment

NO_IDEA_FOUND

Code: 404 NOT_FOUND
Content:

```
{
   "message": {
      "message": "Idea doesn't exists",
      "errorCode": -604,
      "isError": true
   }
}
```

When: It doesn't exist an idea associated with that specific ideaID

NO_ELEMENT_ADDED

Code: 304 NOT_MODIFIED
Content:

```
{
   "message": {
      "message": "No elements added.",
      "isError": true,
      "errorCode": -104
   }
}
```

When: The insertion in the database has gone wrong

CANNOT_ACCESS_DATABASE

Code: 500 INTERNAL_SERVER_ERROR
Content:

```
{
   "message": {
      "message": "Cannot access to the database.",
      "isError": true,
      "errorCode": -100
   }
}
```

When: Database server is unreachable, Errors related to SQL code

BAD_INPUT

Code: 400 BAD_REQUEST
Content:

```
{
   "message": {
      "message": "Input not allowed.",
      "isError": true,
```

```
    "errorCode": -103
  }
}
```


When: The data passed in input (in the url path or in the body) has missing values or are not in the expected format. Possibile examples are a not number id or a malformed JSON

3) Ban a user

With this request we modify the status “banned” of a user from *False* to *True*.

- **URL**
  api/moderator/ban/{userID}

- **Method**
  PUT

- **URL Params**
  - userID = {string}
    The id of the User that has to be banned

- **Data Params**
  No data is passed when requiring this method

- **Success Response**
  The RestApiServlet returns a JSON object with a message field containing a response string.

  Code: 200
  Content:


```
{
  "message": {
    "message": "User banned.",
    "isError": false
  }
}
```


- **Error Response**

  USER_NOT_LOGGED

  Code: 405 METHOD_NOT_ALLOWED
  Content:


```
  {
    "message": {
      "message": "You must be logged in.",
      "isError": true,
      "errorCode": -110
    }
  }
```

When: An unregistered user try to use HyperU

USER_NOT_AUTHORIZED

Code: 405 METHOD_NOT_ALLOWED
Content:

```
{
   "message": {
      "message": "You cannot access this page",
      "isError": true,
      "errorCode": -111
   }
}
```

When: The user is not a moderator

NO_USER_FOUND

Code: 404 NOT_FOUND
Content:

```
{
   "message": {
      "message": "User doesn't exists",
      "errorCode": -126,
      "isError": true
   }
}
```

When: It doesn't exist a User associated with that specific userID

NO_CHANGES

Code: 304 NOT_MODIFIED
Content:

```
{
   "message": {
      "message": "No changes",
      "isError": true,
      "errorCode": -105
   }
}
```

When: The update of the "banned" status in the database has gone wrong or the user was already banned.

CANNOT_ACCESS_DATABASE

Code: 500 INTERNAL_SERVER_ERROR

Content:

```
{
   "message": {
      "message": "Cannot access to the database.",
      "isError": true,
      "errorCode": -100
   }
}
```

When: Database server is unreachable, Errors related to SQL code

BAD_INPUT

Code: 400 BAD_REQUEST
Content:

```
{
   "message": {
      "message": "Input not allowed.",
      "isError": true,
      "errorCode": -103
   }
}
```

When: The data passed in input (in the url path or in the body) has missing values or are not in the expected format. Possibile examples are a not number id or a malformed JSON