
WEB DEVELOPMENT

SECOND PRACTICAL TASK – TYPESCRIPT AND ANGULAR

The objective of the second practice is for the student to expand their knowledge in the design and implementation of the connection part with the server using asynchronous technologies at the same time that they can use a specific development environment such as Angular.

In this practice, other processes associated with other options of the main menu shown in the first practice must be solved. The system must be a "single page" application and must be developed using the **Angular** framework.

WARNING:

As in the first practice, a REST API will be used that provides the "back-end" service to the application (management of users and scores). This REST API is available to students so that they can test and validate the generated code before it is submitted for evaluation. The basic features of the REST API are:

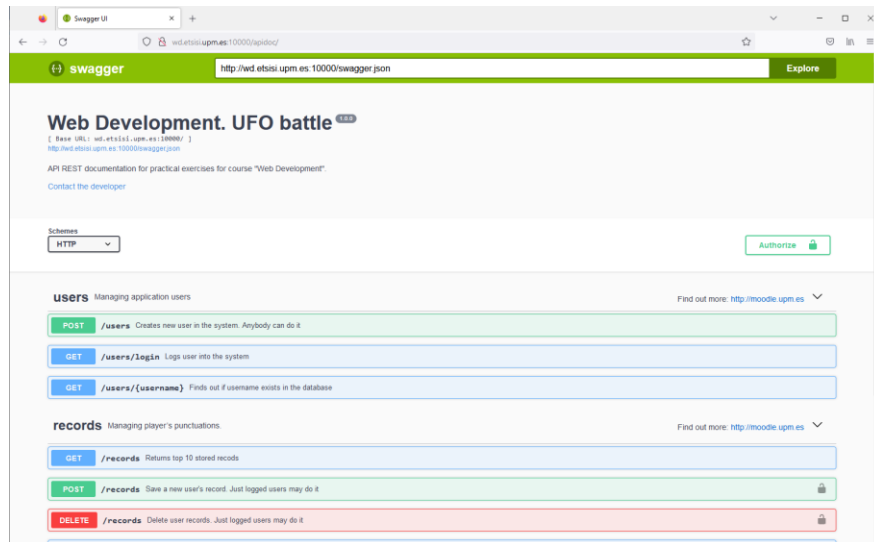
Base URL: <http://wd.etsisi.upm.es:10000> (Note the use of port 10000)

The API is sufficiently documented through the "swagger" tool at the url <http://wd.etsisi.upm.es:10000/apidoc>

This specification page allows, on the one hand, to check the accessible routes and methods; on the other, check the parameters necessary for its execution as well as the results that it provides as a response; and finally, test that its operation is understood through the execution of the methods available on the same page.

It is **important** to note that, in registration requests using the **POST** method, the parameters passed to the API, **in the body**, can be passed as independent parameters (**urlencoded**), (**username=newname&email=newemail&password=newpasswd**) or as a json object whose properties are the names of the quoted parameters (**{username:newname, email:newemail, password:newpassword}**). However, **all responses received from the REST API will be in json format** (even if it's just a simple string or number).

All requests to the routes that concern the registration or deletion of scores require, in a mandatory way, the word "**Bearer**" and the **authorization token** to be passed as content of the "**Authorization**" header obtained in the login operation (separated by a blank space). Since the token is valid for 10 minutes, the REST API, each time it performs an operation, updates and provides a new token with another 10 minutes of validity, in such a way that it expires only if the user does not request any operation during that time.



IMPORTANT NOTE: The student will redo the first practice in such a way that **all** the code, including the "Play" option, is developed with the Angular framework.



Functionality of the "Registration" option

This option will allow the user to proceed with the necessary registration to be able to connect to the system and it **cannot be executed if the user is already connected** to it (he or she has already successfully completed the login operation and therefore has a valid connection token). Once the option is chosen, the system must display the form designed for this purpose in the first practical task. The user must fill in the form, all fields being **mandatory**.

The system **must verify the uniqueness** of the username **before** the data is sent by the user. To do this, when the user moves from one field to another (the element in question loses focus), an asynchronous call to the available REST API must be executed, passing the value of the username identifier as part of the URL:

Method: **get** url: `http://wd.etsisi.upm.es:10000/users/{username}`

The different answers to this request are available in the indicated documentation

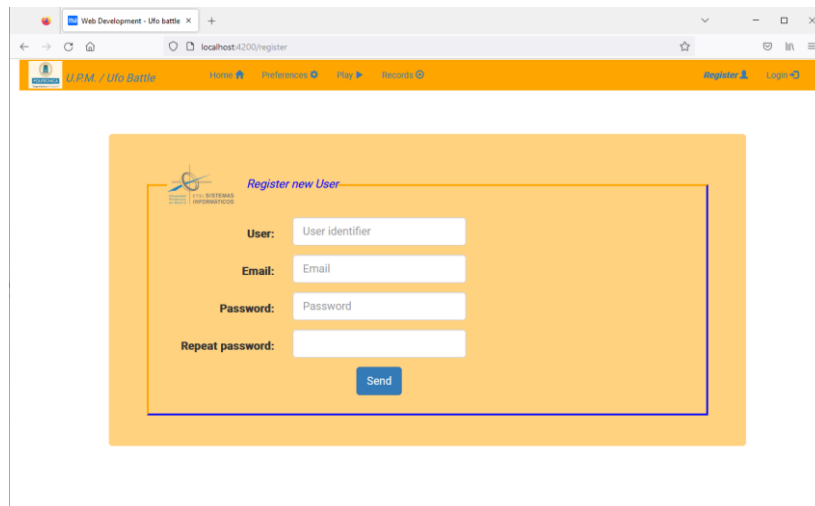
Once these checks have been passed (which, if they do not produce errors, go unnoticed by the user), the user must be able to send (by pressing the “send” button) all the data: **“username”**, **“email”**, **“password”** (all character strings).

This sending will be done, again, to the REST API

Method: **post** url: `http://wd.etsisi.upm.es:10000/users`

The different answers to this request are available in the indicated documentation.

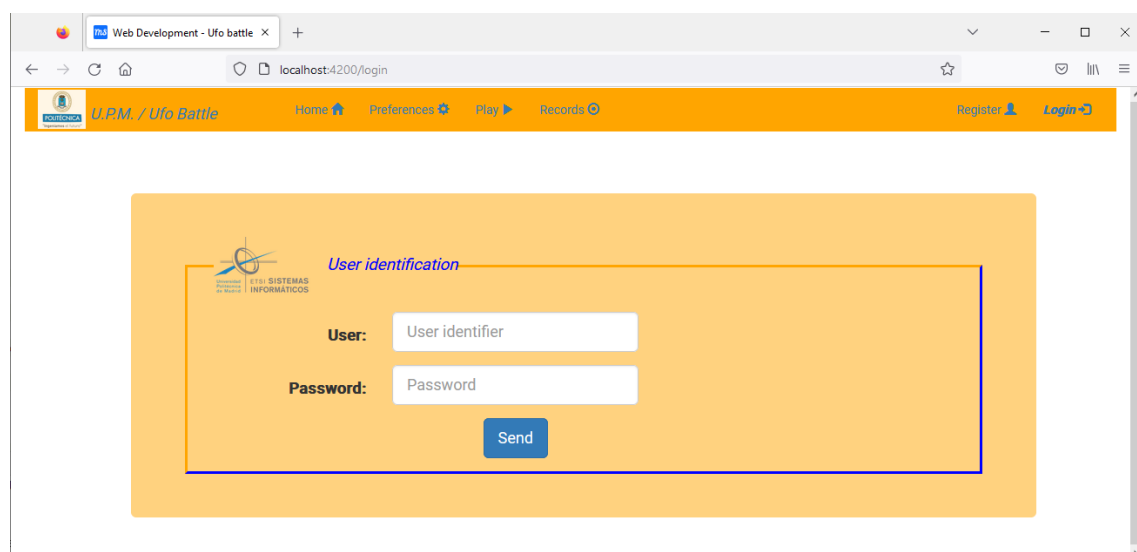
The system must inform the user of the result of the final process by means of a small message.



The screenshot shows a web browser window with the URL `localhost:4200/register`. The page has an orange header bar with the text 'U.P.M. / Ufo Battle' and navigation links: 'Home', 'Preferences', 'Play', and 'Records'. On the right side of the header are links for 'Register' and 'Login'. The main content area features a 'Register new User' form with a blue border. The form includes four input fields: 'User identifier', 'Email', 'Password', and 'Repeat password'. A blue 'Send' button is located at the bottom right of the form. The background of the form area is a light orange color.

Functionality of the “Login” option

This option will allow a registered user to access the system to proceed to record their scores. In order to perform the “login”, the user must identify himself by means of a **username** accompanied by the **password** that he also registered:



The screenshot shows a web browser window with the URL `localhost:4200/login`. The page has an orange header bar with the text 'U.P.M. / Ufo Battle' and navigation links: 'Home', 'Preferences', 'Play', and 'Records'. On the right side of the header are links for 'Register' and 'Login'. The main content area features a 'User identification' form with a blue border. The form includes two input fields: 'User identifier' and 'Password'. A blue 'Send' button is located at the bottom right of the form. The background of the form area is a light orange color.

Once the *login* option is pressed, the designed form should appear to the user. The system must force the user, before proceeding to send the data, to fill in *the user identifier* field and the *password*.

Since the server being accessed provides us with a REST API, the system must cancel the regular submission of the form and proceed to the **asynchronous** request. The parameters, as can be seen in the REST API specification, will be **username** and **password**, in the query, in **urlencoded format** (not json). It can also be verified in the specification that the url to which to make the request is: <http://wd.etsisi.upm.es:10000/users/login>

Upon a positive identification, the REST API returns a security *token* (JWT) in the http **header** of the response called **Authorization**. As determined by the standard, the content of the header will be the token preceded by the string "**Bearer**" and a **blank space**. This token will have been generated by the REST API server with a validity of 10 minutes. The student must store it locally during the user session since they will have to send it in the header (to the REST API) every time they want to access any of the options that require identification.

Attention: Normally the system would not return any message in the body (http code 200 –OK). However, in this case and as a check, the REST API provides a copy of the received token in the body of the message. This copy should only be used as verification, if necessary, during the development phase of the practice.

If the user has not been correctly identified, the system **should warn** him through the corresponding message, while if the security token has been received normally, in the system options menu, the "**login**" option should be changed. by "**logout**" that will work, when selected, locally **destroying** the **token**.

ATTENTION: The student will be able to test the operation of his/her code with the users and passwords that are detailed below:

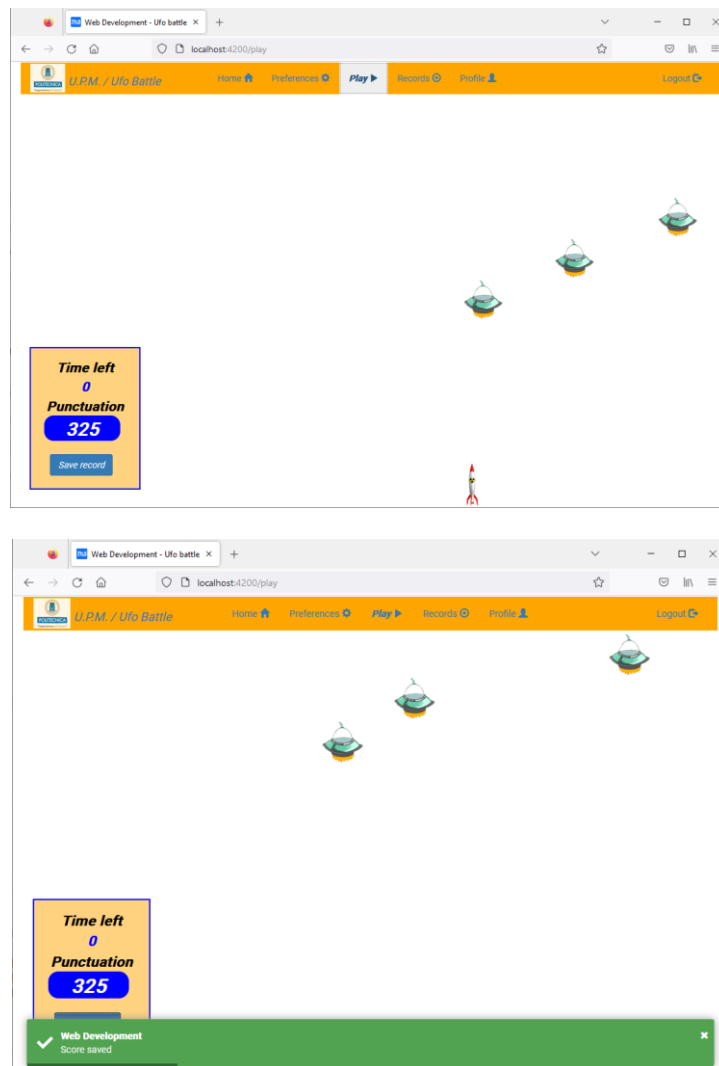
User: **user1** password: **user1**

User: **user2** password: **user2**

Logout: This option, which will not cause any REST API calls, **will destroy the token** on the client side and send the user to the *Home* page, with the option to login again.

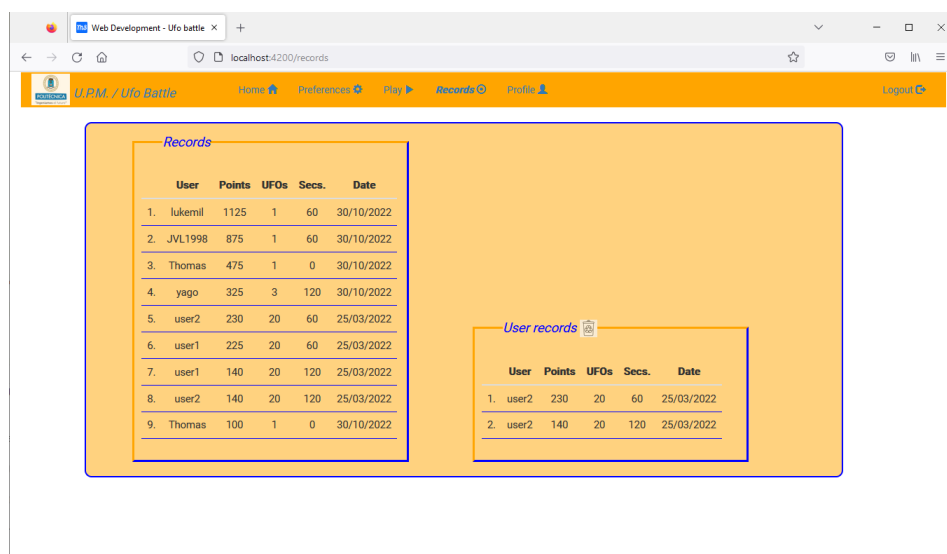
Extension of the functionality of the "Play" option

If the user is playing properly identified (he is connected and **has a valid security token**), at the end of a game and, **only at that moment**, the system will offer him the possibility of "**Recording**" the **score** obtained. In case of using this option, the system will make a **POST** request to the REST API according to the specification shown in the documentation, sending the final score, the number of UFOs with which he/she has played and the time used (eg: {"punctuation" : 200, ufos: 3, disposedTime: 120}) in order to record the score obtained. In order for it to work, the token obtained in the "**login**" operation must be sent in the "**Authorization**" header of the request.



Extension of the functionality of the "Records" option

If the user has made a "login" correctly, when requesting the execution of the "**Records**" option, a **list with his/her ten highest records will also appear**. Of course, for this, it will have to be send the security token.



IT IS REQUESTED:

A zip file must be uploaded to the moodle platform containing all those elements so that, once decompressed, the practice can be viewed and executed without problems after doing the “*npm install*” command. The decompression of the file must give rise to the hierarchy of directories necessary to place each file where it belongs, including those that may belong to the libraries (bootstrap, jquery, etc.) if CDN is not used for their incorporation.

ATTENTION: Folder “*node_modules*” must **NOT** be uploaded as it can be regenerated and it contains a lot of files, as any other that is specific for the developing environment (ex.: . angular)

EVALUATION

- ✓ The evaluation of the practice will be carried out by assessing the aspects indicated in the corresponding rubric published on the platform.