Instituto Superior Técnico

MEEC

Aplicações Distirbuídas pela Internet

IST Big Brother

Trabalho realizado por:
- João Gomes, 96248, joaomgomes2001@tecnico.ulisboa.pt
- Matilde Oliva, 96283, matilde.oliva@tecnico.ulisboa.pt
- Miguel Vicente, 96288, miguel.mendes.vicente@tecnico.ulisboa.pt

2022/2023
1ºSemestre, 1º Período

1 Architecture

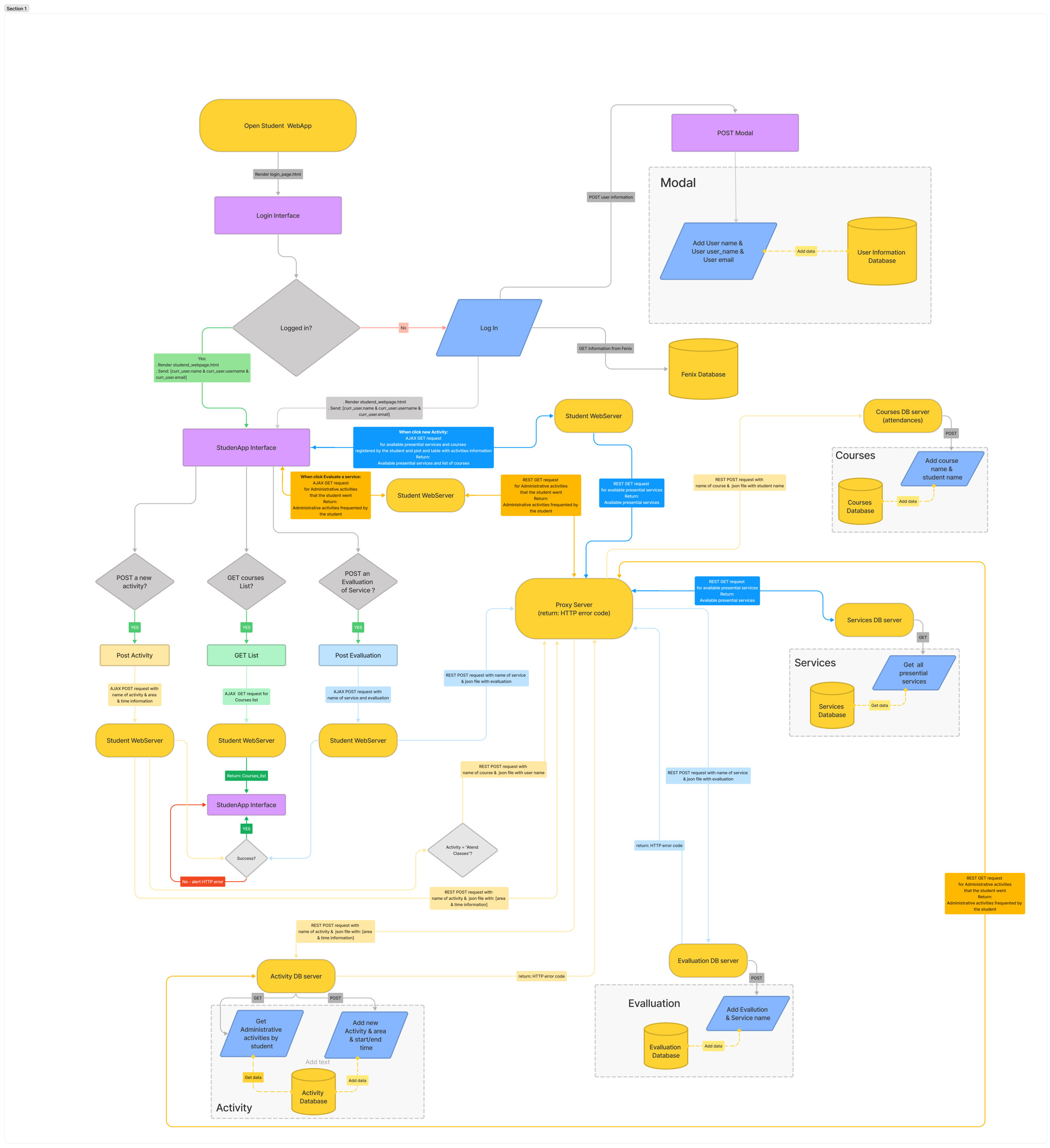
Section 1
Open Student WebApp
Render login_page.html
Login Interface
POST Modal
POST user information
Modal
Add User name & User user_name & User email
Add data
User Information Database
Logged in?
No
Log In
GET information from Fenix
Fenix Database
Yes:
. Render studend_webpage.html
. Send: [curr_user.name & curr_user.username & curr_user.email]
. Render studend_webpage.html
. Send: [curr_user.name & curr_user.username & curr_user.email]
StudenApp Interface
When click new Activity:
AJAX GET request
for available presential services and courses
registered by the student and plot and table with activities information
Return:
Available presential services and list of courses
Student WebServer
When click Evaluate a service:
AJAX GET request
for Administrative activities
that the student went
Return:
Administrative activities frequented by the student
Student WebServer
REST GET request
for Administrative activities
that the student went
Return:
Administrative activities frequented by the student
REST GET request
for available presential services
Return:
Available presential services
REST POST request with
name of course & json file with student name
Courses DB server (attendances)
POST
Courses
Add course name & student name
Add data
Courses Database
POST a new activity?
GET courses List?
POST an Evalluation of Service ?
YES
YES
YES
Post Activity
GET List
Post Evalluation
Proxy Server
(return: HTTP error code)
REST GET request
for available presential services
Return:
Available presential services
Services DB server
GET
Services
Get all presential services
Get data
Services Database
AJAX POST request with name of activity & area & time information
AJAX GET request for Courses list
AJAX POST request with name of service and evalluation
REST POST request with name of service & json file with evalluation
Student WebServer
Student WebServer
Student WebServer
Return: Courses_list
StudenApp Interface
YES
Success?
No - alert HTTP error
REST POST request with
name of course & json file with user name
Activity = "Atend Classes"?
REST POST request with name of service & json file with evalluation
return: HTTP error code
REST GET request
for Administrative activities
that the student went
Return:
Administrative activities frequented by the student
REST POST request with
name of activity & json file with: [area & time information]
REST POST request with
name of activity & json file with: [area & time information]
Activity DB server
return: HTTP error code
GET
POST
Get Administrative activities by student
Add new Activity & area & start/end time
Add text
Get data
Add data
Activity Database
Activity
Evaluation DB server
POST
Evalluation
Add Evallution & Service name
Add data
Evalluation Database

Figura 1: Student WebApp Architecture

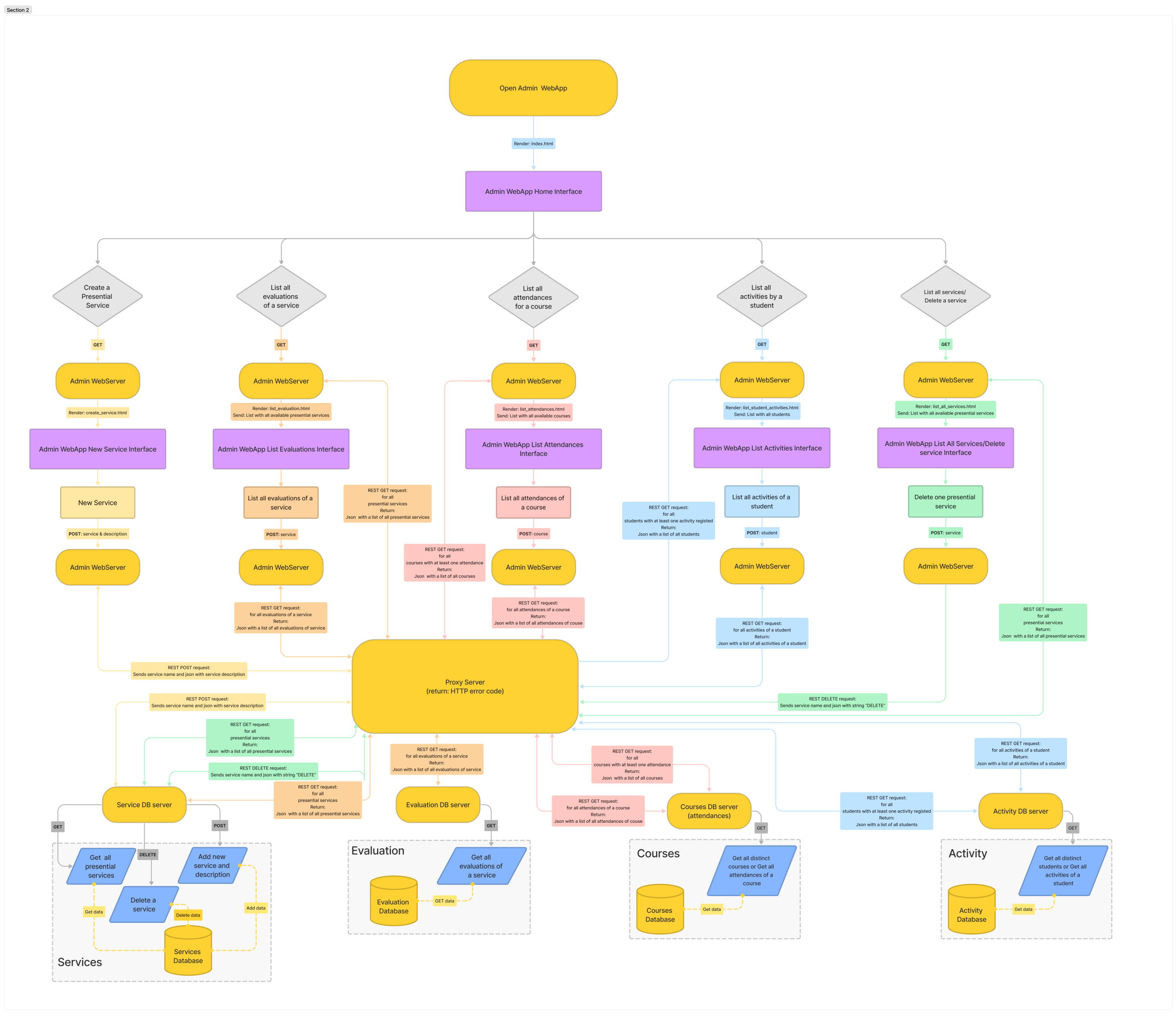
Section 2
Open Admin WebApp
Render: index.html
Admin WebApp Home Interface
Create a Presential Service
List all evaluations of a service
List all attendances for a course
List all activities by a student
List all services/ Delete a service
GET
GET
GET
GET
GET
Admin WebServer
Render: create_service.html
Admin WebApp New Service Interface
New Service
POST: service & description
Admin WebServer
Admin WebServer
Render: list_evaluation.html Send: List with all available presential services
Admin WebApp List Evaluations Interface
List all evaluations of a service
POST: service
Admin WebServer
REST GET request: for all presential services Return: Json with a list of all presential services
REST GET request: for all evaluations of a service Return: Json with a list of all evaluations of service
Admin WebServer
Render: list_attendances.html Send: List with all available courses
Admin WebApp List Attendances Interface
List all attendances of a course
POST: course
Admin WebServer
REST GET request: for all courses with at least one attendance Return: Json with a list of all courses
REST GET request: for all attendances of a course Return: Json with a list of all attendances of couse
Admin WebServer
Render: list_student_activities.html Send: List with all students
Admin WebApp List Activities Interface
List all activities of a student
POST: student
Admin WebServer
REST GET request: for all students with at least one activity registed Return: Json with a list of all students
REST GET request: for all activities of a student Return: Json with a list of all activities of a student
Admin WebServer
Render: list_all_services.html Send: List with all available presential services
Admin WebApp List All Services/Delete service Interface
Delete one presential service
POST: service
Admin WebServer
REST GET request: for all presential services Return: Json with a list of all presential services
REST DELETE request: Sends service name and json with string "DELETE"
REST POST request: Sends service name and json with service description
Proxy Server (return: HTTP error code)
REST POST request: Sends service name and json with service description
REST GET request: for all presential services Return: Json with a list of all presential services
REST DELETE request: Sends service name and json with string "DELETE"
REST GET request: for all presential services Return: Json with a list of all presential services
REST GET request: for all evaluations of a service Return: Json with a list of all evaluations of service
REST GET request: for all courses with at least one attendance Return: Json with a list of all courses
REST GET request: for all attendances of a course Return: Json with a list of all attendances of couse
REST GET request: for all activities of a student Return: Json with a list of all activities of a student
REST GET request: for all students with at least one activity registed Return: Json with a list of all students
Service DB server
GET
DELETE
POST
Get all presential services
Add new service and description
Delete a service
Get data
Delete data
Add data
Services Database
Services
Evaluation DB server
GET
Evaluation
Get all evaluations of a service
Evaluation Database
GET data
Courses DB server (attendances)
GET
Courses
Get all distinct courses or Get all attendances of a course
Courses Database
Get data
Activity DB server
GET
Activity
Get all distinct students or Get all activities of a student
Activity Database
Get data

Figura 2: Admin WebApp Architecture

2 Modules

2.1 Presential Services DB

Tabela 1: Presential Services DB

Parameters	**Data**		
Presential_Services.service_name	Secretaria	Secção de Folhas	Cantina
Presential_Services.data_time	13/10/2022	13/10/2022	13/10/2022
Presential_Services.service_content	(Descrição)	(Descrição)	(Descrição)

a) /API/services/<path:service>

This endpoint handle both GET and POST request.
If a "GET"request is received and the argument *service* is equal to a service name, it is intended to verify if that service exists in the database, returning the respective status code. If the argument *service* is equal to "all", it is intended to list all the services. Then, a json file with this list is returned to the proxy server.
If a "POST"request is received, *service* represents a service name. If the value of *content* of the json file received as the request's response is equal to "Delete", then the service will be deleted from the database. If not, it is intended to create a new service with the name of *service* and with the content from the json file. a status code is returned.

2.2 Courses DB

Tabela 2: Courses DB

Parameters	**Data**		
Courses.course_name	ADInt	CInte	AED
Courses.data_time	13/10/2022	13/10/2022	13/10/2022
Courses.student	Miguel	João	Matilde

a) /API/courses/<path:course>

This endpoint handle both GET and POST request.

If a "GET"request is received and the argument *course* is a course name, it is intended to verify if that course exists in the database and it is intended to list all students who attended that course. If the argument *course* is equal to "all", else is intended to return a list with all courses with at least one attendance. Then, a json file with this list is returned to the proxy server.
If a "POST"request is received, *course* represents the course name. Then it is intended to create/add a the course to the data base with the name of *course* and with the content from the json file, which is the student name. In the end a status code is returned.

2.3 Activities records DB

Tabela 3: Activities records DB

Parameters	Data		
Activities.student	Miguel	João	Matilde
Activities.data_time	06/11/2022	06/11/2022	06/11/2022
Activities.area	Personal	Academic	Administrative
Activities.activity	Sports	Study	NMCI
Activities.start	06/11/2022 16:00	06/11/2022 16:00	06/11/2022 16:00
Activities.end	06/11/2022 18:00	06/11/2022 18:00	06/11/2022 18:00

a) /API/activities/<path:arg>

This endpoint handles both GET and POST request.
If a "GET"request is received, the argument *arg* may have multiple values. If it is equal to "plot"is intended to list the time(return a json file with the start and end time) the student spent doing an activity of each area, so a line plot can be done, if the argument *arg* is equal to "table"it is intended to list all activities done by the student and return a json file with all the information about the activity, name, area, start and end time, so a table can be drawn, if the argument *arg* is equal to "all"it is intended to list all the students that have done an activity, and return a json file with them, if the argument *arg* isn't equal to neither it will be equal to the student name, and in this case we may want two different things from the DB, or to list all the activities done by the student, if the *area* argument is equal to student or to list all the distinct administrative activities done by the student, if the *area* argument is equal to administrative. In all cases it return also a error code.
If a "POST"request is received, *arg* represents the name of the activity. It is intended to create a new activity in a certain area, Personal, Academic or Administrative, for a certain student, with a start and end time, and a status code is returned.

2.4 Presential Services Evaluation DB

Tabela 4: Presential Services Evaluation DB

Parameters	Data		
Evaluations.service	Secretaria	Secção de Folhas	Cantina
Evaluations.data_time	13/10/2022	13/10/2022	13/10/2022
Evaluations.evaluation	(avaliação)	(avaliação)	(avaliação)

a) /API/evaluations/<path:service>

This endpoint handles both GET and POST requests.
If a "GET"request is received and the argument *service* represents a service name, it is intended to get the list of evaluations of that service. First, it is verified if the service has evaluations, and then the list with the services is returned in a JSON file. If the *service* equals "all", then

the list of all services and respective evaluations is returned, in a JSON file.
If a "POST"request is received, it is intended to register a new evaluation with the information in the JSON file received.

2.5 Proxy

In this server we implemented four REST endpoints, one for each database:

a) /API/services/<path:service>

This endpoint handles both GET and POST requests.
If a GET request is received it sends a GET request to the Services Database Server REST endpoint with the variable service equal to the service name or equal to "all"in the case of listing all Presential services. In this last case, it receives from the Services Database Server a JSON file containing the list of all Presential Services registered in the Database.
If a Post request is received, it sends a POST request to the Services Server Database REST endpoint with the variable service equal to the service name and a JSON file that contains the description of the new service that will be added to the database. In other cases it contains the string "Delete", when wanted to delete a service from the Database.

b) /API/courses/<path:course>

This endpoint handles both GET and POST requests.
If a GET request is received it sends a GET request to the Courses Database Server REST endpoint with the variable *course* equal to the course name or equal to "all"depending on the functionality. Receives from the Courses Database Server a JSON file containing the list of all courses registered in the Database in case of the variable *course* is equal to "all"or a JSON file with all attendances of a course where the variable *course* is equal to the course name.
If a Post request is received it sends a POST request to the Courses Server Database REST endpoint with the variable *course* equal to the course name and a JSON file that contains the name of the student that attended this course.

c) /API/activities/<path:arg>

This endpoint handles both GET and POST requests.
If a GET request is received it sends a GET request to the Activities Database Server REST endpoint with the variable *arg* equal to the "student name"or equal to "all"or equal to "plot"or "table"it also sends a JSON file that contains a parameter "area"which is equal or to "Student"or to "Administrative". These different options of variables value will be used to decide what will be returned from the Activities Server Database. Receives from the Activities Database Server a JSON file. The content of this JSON file depends on the *arg* value, this is explained on subsection 2.3.

If a Post request is received it sends a POST request to the Activities Server Database REST endpoint with the variable *arg* equal to the activity name and a JSON file that contains four parameters: student_name, area, start, end.

d) /API/evaluations/<path:service>

This endpoint handles both GET and POST requests.
If a GET request is received it sends a GET request to the Evaluations Database Server REST endpoint with the variable *service* equal to the "service name"or equal to "all". These different options of variables value will be used to decide what will be returned from the Evaluations Server Database. Receives from the Evaluations Database Server a JSON file. The content of this JSON file depends on the *service* value, this is explained on subsection 2.4.
If a Post request is received it sends a GET request to the Services Server Database Rest endpoint to verify if the service that will be evaluated exists. Then, if it exists,it sends a POST request to the Evaluations Server Database REST endpoint with the variable *service* equal to the service name and a JSON file that contains two parameters: service_name and evaluation content.

2.6 Admin Web App/server

a) /API/services

This endpoint handles both GET and POST requests.

If a GET is received, it is returned the HTML responsible for retrieving the information to create a new service.
If a POST is received, this information is forwarded to the proxy server services REST endpoint, through another POST request, with the variable "service"equal to the service name and a JSON file with the rest of the information.

b) /API/list_services

This endpoint handles both GET and POST requests.

If a GET is received, a GET request is sent to the proxy server service REST endpoint with the variable "service"equal to "all". The response to this request is the list of all services, in a JSON file. A template is returned with the list of services. This template also allows the user to select a service to delete. This will be dealt with through a POST request: the information is retrieved from the HTML and then a POST request is made to the proxy server, with the variable "service"equal to the service name and a JSON file with the variable *content* equal do "Delete".

c) /API/evaluations/

This endpoint handles both GET and POST requests.
First, it sends a GET request to the Proxy Server evaluations REST endpoint with the variable *service* equal to "all". It receives from the Proxy Server a JSON file containing the list of all services. If a GET request is received, it is returned to the HTML where it is possible to see all services list and select one to see its evaluations.

If a Post request is received it sends a GET request to the Proxy Rest endpoint with the variable *service* equal to the service name, to obtain the evaluations of that service. The response is a JSON file with the list of all evaluations of service.

d) /API/activities/

This endpoint handles both GET and POST requests.
First, it sends a GET request to the Proxy Server REST activities endpoint with the variable *activity* equal to "all". It receives from the Proxy Server a JSON file containing the list of all students. If a GET request is received, it is returned to the HTML where it is possible to see all student's lists and select one to see his activities.

If a Post request is received it sends a GET request to the Proxy Rest endpoint with the variable *activity* equal to the student name and a JSON file with *area* equal to "Student", to indicate that the list of activities of a student is being requested. The response is a JSON file with that list.

e) /API/courses/

This endpoint handles both GET and POST requests.
First, it sends a GET request to the Proxy Server course REST endpoint with the variable *courses* equal to "all". It receives from the Proxy Server a JSON file containing the list of all courses. If a GET request is received, it is returned the HTML where it is possible to see all courses listed and select one to see its attendance.

If a Post request is received it sends a GET request to the Proxy Rest endpoint with the variable *course* equal to the course name. The response is a JSON file with the attendance for that course.

2.7 Students web App/server

a) Student info DB

In this Database is registered the information of the user that is obtained when a student does the login.

Tabela 5: Student info DB

Parameters	Data
User.id	1
User.username	ist196288
User.name	Miguel Mendes Vicente
User.email	miguel.mendes.vicente@tecnico.ulisboa.pt
User.courses	Cinte, ADint (...)

In the following sections will be explained the Student WebApp endpoints and the respective requests:

b) Endpoint: /

This endpoint returns the main page template (student_webpage.html) in case the user is logged or if not returns the login page template (login_page.html). When the main page template is returned, it also sends along with it the Student name, Student IST ID and Student email.

c) Endpoint: /evaluations/new

This endpoint handles POST and GET requests. When a GET request is received, sends a GET request to the proxy server to activities REST endpoint with the variable *student* equal to the student name and a JSON file with a parameter *area* equal to "Administrative" to obtain all Presential Services that that student has frequented, because the user can only evaluate a service which has already been to. Returns to the web page a JSON file with a parameter *lista* that contains all presential services the student frequented.
When a POST request is received, it means that the student evaluated a service, so it sends a POST request to the proxy server to evaluations REST endpoint with the variable *service* equal to the name of the service that was evaluated and a JSON file with two parameters: *name* that contains the service name and *content* that includes the evaluation of the service.

d) Endpoint: /activities/new

This endpoint handles POST and GET requests. If a GET or POST request is received it sends a GET request to the proxy server to service REST endpoint with the variable *service* equal to "all" to obtain all available Presential Services.

In case of GET request, it also returns to the webpage a JSON file with two parameters, "lista": a list of all available Presential Services and "courses": a list of all courses that the student is registered.
Besides that, when a POST request is received, it also sends a POST request to the proxy server to activities REST endpoint with the variable *activity* equal to the name of the activity registered by the student and a JSON file that contains four parameters: "student_name": name of the student; "area": type of the activity (Personal, Academic or Administrative); "start: Time and date when the activity started; "end": Time and date when the activity ended.
It also sends a POST request to the proxy server to courses REST endpoint if the student registered the activity "Attend Classes", so it is necessary to save this in the Courses DB because it means that the student attended a class from a specific course, so it sends the variable *course* equal to the name of the course frequented and a JSON file with the student name.

e) Endpoint: /courses

This endpoint only handles GET requests. If a GET request is received, returns a JSON file with a list of the courses in which the student is registered.

f) Endpoint: /output

Here the plot of activities registered by the student is created.
This endpoint only handles GET requests.
When a GET request is received, it sends a GET request to the proxy server to activities REST endpoint with the variable *arg* equal to "plot"and a JSON file with a parameter "area": type of the activity (Personal, Academic or Administrative). Returns to the webpage a JSON file with the plot url.

g) Endpoint: /table

This endpoint only handles GET requests.
When a GET request is received it sends a GET request to the proxy server to activities REST endpoint with the variable *activity* equal to "table"and a JSON file with a parameter "name": student name.
Returns to the webpage a JSON file with a parameter "table": a list of all activities information registered by the student.

3 Complex interactions

3.1 Login

When a student first opens the student webapp will appear the login page. After pressing the Login button, it will be redirected to fenix login to put the IST id and password.
At this moment is called the endpoint: "/login/callback". Here all the information needed from fenix is obtained through GET requests to specific Fenix endpoints depending on which information is required. Then, this information is registered in the Students Info DB as shown in section a). Saving this information on the Database facilitates its utilization in the rest of the code.

3.2 Creation of an activity

This functionality is more complex because requires getting a lot of information from different databases.

Here the student can register three types of activities: Personal, Academic and Administrative. When the user clicks on the button to "Create a New Activity"in the "Menu"area, two functions are called, "plot()"and "table()"explained here: d). Then it is sent an AJAX GET request to the Student Web Server to the endpoint: /activities/new. It receives a JSON file with two parameters: "courses": list of all courses that the student is registered; "lista": list of all presential services available. Then, these lists are printed in the respective div.

a) Personal Activities

In this Tab, the user must choose between predefined options of activities or create a new one by selecting "other". To submit, the user must also choose a start and end date/time.
The user can only submit if he fulfils all the camps.
When the user clicks on the button "Submit", it is sent an AJAX POST request to the Student Web Server to the endpoint: /activities/new; with a JSON file with six parameters: "Personal": name of the activity of type *Personal* chosen by the student; "Academic": name of the activity of type *Academic* chosen by the student; "Administrative": name of the activity of type *Administrative* chosen by the student; "newactivity": name of the new activity added by the student when he chooses the option "Other"; "date_Personal_start": date/time of the start of the activity; "date_Personal_end": date/time of the end of the activity. After the post request, two functions are called, "plot()"and "table()"explained here: d).

b) Academic Activities

In this Tab, the user must choose between two predefined options, "Study"or "Attend Classes". If he chooses the second option, a list of all courses that the student is registered in (that was obtained when the student chose "Create a New Activity"in the "Menu"area) appears. To submit he also must choose a start and end date/time.

The user can only submit if he fulfils all the camps.
When the user clicks on the button "Submit", is sent an AJAX POST request to the Student Web Server to the endpoint: /activities/new; with a JSON file with six parameters: "Personal": name of the activity of type *Personal* chosen by the student; "Academic": name of the activity of type *Academic* chosen by the student; "Administrative": name of the activity of type *Administrative* chosen by the student; "Courses": name of the course that was attended; "date_Academic_start": date/time of the start of the activity; "date_Academic_end": date/time of the end of the activity. After the post request, two functions are called, "plot()"and "table()"explained here: d).

c) Administrative Activities

In this Tab, the user must choose an option from a list of all Presential services available (that was obtained when the student chose "Create a New Activity"in the "Menu"area). To submit he also must choose a start and end date/time.
The user can only submit if he fulfils all the camps.
When the user clicks on the button "Submit", it is sent an AJAX POST request to the Student Web Server to the endpoint: /activities/new; with a JSON file with five parameters: "Personal": name of the activity of type *Personal* chosen by the student; "Academic": name of the activity of type *Academic* chosen by the student; "Administrative": name of the activity of type *Administrative* chosen by the student; "date_Academic_start": date/time of the start of the activity; "date_Academic_end": date/time of the end of the activity. After the post request, two functions are called, "plot()"and "table()"explained here: d).

d) Javacript table() and plot() functions

plot(): This function sends an AJAX GET request to the Admin Web Server to the endpoint: /output. Then, after receiving the data, the plot with student's activities' statistics is printed.

table(): This function sends an AJAX GET request to the Admin Web Server to the endpoint: /table. Then after receiving the data, a table with all student activities information is printed.

3.3 Evaluation of presential services

Here the student can register an evaluation of a Presential Service. When the user clicks the button to "Create a New Evaluation"in "Menu"area, it is sent an AJAX GET request to the Student Web Server to the endpoint: /evaluations/new. It received a JSON file with one parameter: "lista": list of all presential services that the student frequented. Then, this list is printed in a select div. When the user clicks on the button "Submit"to submit a new evaluation, it is sent an AJAX POST request to the Student Web Server to the endpoint: /evaluations/new, with a JSON file with two parameters: "service": name of service that was evaluated; "evaluation": the content of the evaluation.

3.4 List of all courses

Here it is presented to the student all courses in which he is registered. When the user clicks the button to "List all Courses"in thf "Menu"area, it is sent an AJAX GET request to the Student Web Server to the endpoint: /courses. It received a JSON file with one parameter: "courses": list of all courses that the student is registered. Then, prints this in the respective div.

4 Changes to the intermediate project

- Activities DB: addition of more parameters: area of the activity and start and end time.

Besides that, we improve the admin app by adding some select options in order to facilitate the usage of the interface by the user. To do this we have to add some GET and POST requests making use of what we have already implemented.

5 Non implemented functionalities

- getting the courses' schedules from FENIX when registering an academic activity. Instead, a start and end time of the academic activity is requested when registering this new activity.
- The functionality: "Delete a presential service", is implemented, but the REST communication between servers is not correct. We don't use "requests.delete"when deleting something from a database as we should do. Instead of it, we use a POST request, and to distinguish this request from other POSTs, we send on the JSON file a parameter with the string "DELETE". It works but its not the correct way.