TÉCNICO
LISBOA

MEEC

Distributed Applications over the Internet

Distributed System using Proxy, Databases and Webpages

Authors:

Bruno Cebola (93030) bruno.m.cebola@tecnico.ulisboa.pt
Rui Abrantes (93176) rui.miguel.abrantes@tecnico.ulisboa.pt

2022/2023 – 1st Semester, P1

Contents

1 Objective

This project has the objective of developing a multipart system that allows students to control their efforts by keeping records of their daily activities. In order to ensure scalability every part of the system will be independent having its own runtime environment, being composed of three different blocks.

The interaction with the system, which represents the first block, will be done through two different portals: one for users (which is done yet available), where it will be possible to register class attendances, evaluate services, among other, and one for administrators (which was developed during this intermediate project), where it will to be able to track every interaction with the system, register services, among others. Next, the second block will consist of all the databases, which are independent and thus every relation among thus will have be achieve through logic. Finally, the third block will be a proxy to facilitate the communication between every system.

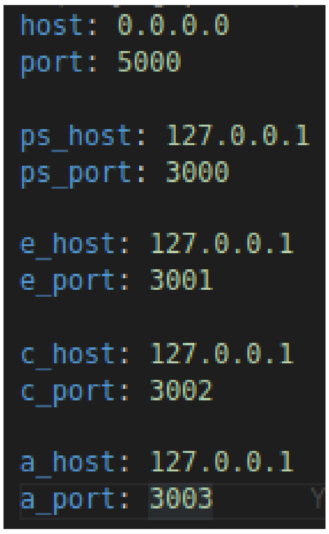
2 Solution presentation

2.1 Proxy

The proxy is at the center of our solution, since it will handle all the business logic in order to connect the different blocks and minimize the overall workload. As an example, if an administrator needs to access all the evaluations of a service, it will be the proxy to handle the different necessary requests and filter the data before returning the final result. This way no user needs to interact directly with the databases.

Furthermore, the proxy will also be in charge of validating who can access the information, because administrators and users have different levels of access.

It is also of high importance to mention that in order to configure the proxy it was created a YAML configuration file, which is mandatory and has a fixed structure, with all of the needed information (proxy's host and port and the existing databases respective hosts and ports), as it is displayed in Figure 1.



```
host: 0.0.0.0
port: 5000

ps_host: 127.0.0.1
ps_port: 3000

e_host: 127.0.0.1
e_port: 3001

c_host: 127.0.0.1
c_port: 3002

a_host: 127.0.0.1
a_port: 3003
```

Figure 1: Proxy configuration file

Bellow, for reference, are listed all the endpoints available in the proxy, along with their descriptions, required fields and examples.

Services:

[GET] **/services** - gets all presential services

[GET] **/services/filter** - gets presential services filtered by column values (the filters are indicated via query)

Query example: ?location=Main building&name=Printer

[GET] **/service/<service_id>** - gets a specific service

[GET] **/services/<service_id>/evaluations** - gets all evaluations of a specific service

[POST] **/services/create** - creates a new service

JSON mandatory fields: name, location

[DELETE] **/service/<service_id>** - deletes a specific service

---

Evaluations:

[GET] **/evaluations** - gets all evaluations information

[GET] **/evaluations/filter** - gets evaluations filtered by column values (the filters are indicated via query)

Query example: ?rating=4

[GET] **/evaluation/<evaluation_id>** - gets a specific evaluation

[POST] **/evaluation/create** - creates a new evaluation. Before creation, checks if service exists

JSON mandatory fields: service_id, rating, student_id

[DELETE] **evaluation/<evaluation_id>** - deletes a specific evaluation

---

Courses:

[GET] **/courses** - gets all courses

[GET] **/courses/filter** - gets courses filtered by column values (the filters are indicated via query)

Query example: ?name=ADInt

[GET] **/course/<course_id>** - gets a specific course
[POST] **/course/create** - creates a new course
JSON mandatory fields: name, professor, school_year
[DELETE] **/course/<course_id>** - deletes a specific course

Activities:
[GET] **/activities** - gets all activities
[GET] **/activities/filter** - gets courses filtered by column values (the filters are indicated via query)
Query example: ?type_id=2&sub_type_id=1
[GET] **/activities/types** - gets all types of activities and corresponding information
[GET] **/activity/type/<type_id>/<sub_type_id>/db** - gets name of the database associated with a specific activity type
[GET] **/activity/<activity_id>** - gets a specific activity
[POST] **/activity/create** - creates a new activity
JSON mandatory fields: type_id, sub_type_id, student_id, start_time, stop_time
[POST] **/activity/start** - creates a new activity with start_time equal to the server datetime and no stop_time
JSON mandatory fields: type_id, sub_type_id, student_id
[PUT] **/activity/<activity_id>/stop** - saves stop_time equal to the server datetime for a specific activity
[DELETE] **/activity/<activity_id>** - deletes a specific activity

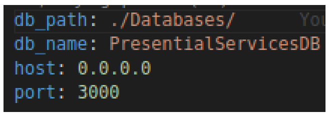
2.2 Databases

The different databases play a crucial role on our system due to being the ones that store all the information. There are four different databases: **PresentialServicesDB**, **EvaluationsDB**, **CoursesDB** and **ActivitiesDB**. Since this is a distributed system, it is necessary to notice that there are no direct relations between the databases.

Nevertheless, there are records that need to register relations with other records and thus our group created a "fake foreign key" to create this connection using the business logic coded into the proxy. This way it is possible to establish relations between tables without the need for them to know the existence of each other.

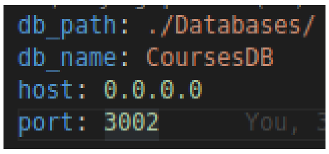
To ensure data security and privacy, when a request is received in any of the databases, no matter the request method, a token validation is performed to ensure the sender of the request can actually access it. At the moment, to restrict at a maximum the accesses, only the proxy is allowed access to the data by providing a token at the time of the request. In the future, there will also be validations for the users and administrators, in case that the proxy fails to validate those.

Again it is important to mention the existence of YAML configuration files, which are mandatory and have a fixed structure. Below it is possible to see the different files for the different databases' environments.



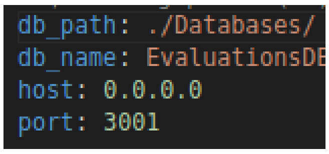
```
db_path: ./Databases/        Yo
db_name: PresentialServicesDB
host: 0.0.0.0
port: 3000
```

Figure 2: Presential services configuration file



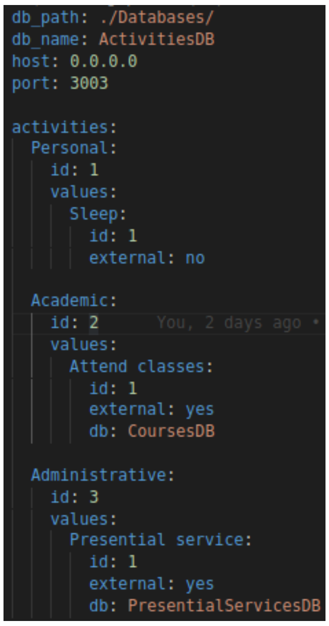
```
db_path: ./Databases/
db_name: CoursesDB
host: 0.0.0.0
port: 3002        You, 
```

Figure 3: Courses configuration file



```
db_path: ./Databases/
db_name: EvaluationsDB
host: 0.0.0.0
port: 3001
```

Figure 4: Evaluations configuration file

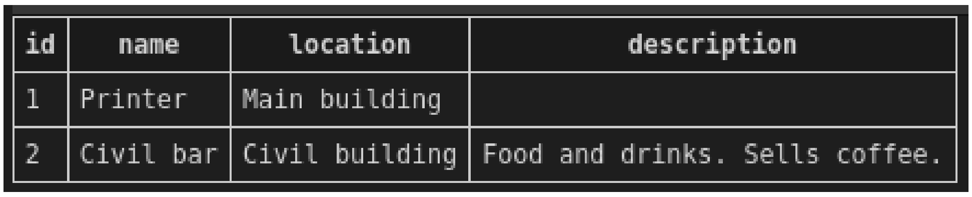


```
db_path: ./Databases/
db_name: ActivitiesDB
host: 0.0.0.0
port: 3003

activities:
  Personal:
    id: 1
    values:
      Sleep:
        id: 1
        external: no

  Academic:
    id: 2          You, 2 days ago •
    values:
      Attend classes:
        id: 1
        external: yes
        db: CoursesDB

  Administrative:
    id: 3
    values:
      Presential service:
        id: 1
        external: yes
        db: PresentialServicesDB
```

Figure 5: Activities configuration file

2.2.1 PresentialServicesDB

The PresentialServicesDB has four columns: **id**, **name**, **location** and **description**, which are described in Table 1. In Figure 6 it is possible to see an example of the data stored in this database.



id	name	location	description
1	Printer	Main building	
2	Civil bar	Civil building	Food and drinks. Sells coffee.

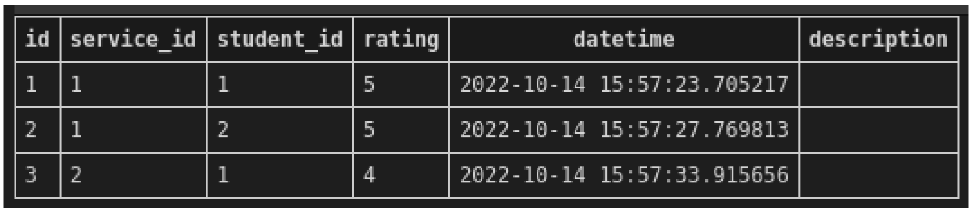
Figure 6: Presential Services Database example

Name	Type	Description
id	Integer	Presential service unique identifier
name	String	Name of the presential service
location	String	Location of the presential service at IST
description	String	Description of what the presential service is

Table 1: Presential Services Database Structure

2.2.2 EvaluationsDB

The EvaluationsDB has six columns: **id**, **service_id**, **student_id**, **rating**, **datetime** and **description**, which are described in Table 2. In Figure 7 it is possible to see an example of the data stored in this database.



id	service_id	student_id	rating	datetime	description
1	1	1	5	2022-10-14 15:57:23.705217	
2	1	2	5	2022-10-14 15:57:27.769813	
3	2	1	4	2022-10-14 15:57:33.915656	

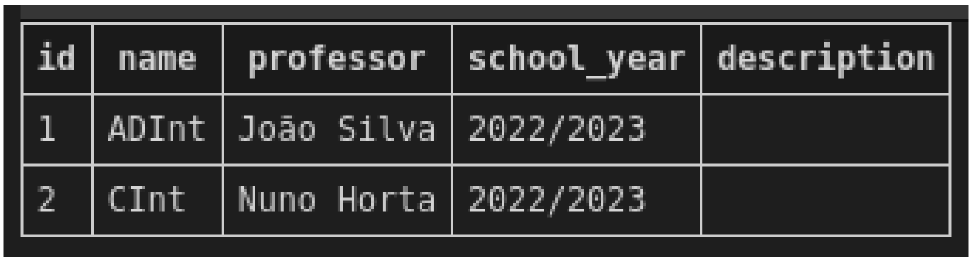
Figure 7: Evaluations Database example

Name	Type	Description
id	Integer	Evaluation unique identifier
service_id	Integer	Service unique identifier to act as foreign key
student_id	Integer	Student unique identifier to act as foreign key
rating	Integer	Value attributed to the service
datetime	DateTime	Date and time of when the evaluation was submitted
description	String	Description of the service and small comment on how was the experience

Table 2: Evaluations Database Structure

2.2.3 CoursesDB

The CoursesDB has five columns: **id**, **name**, **professor**, **school_year** and **description**, which are described in Table 3. In Figure 8 it is possible to see an example of the data stored in this database.



id	name	professor	school_year	description
1	ADInt	João Silva	2022/2023	
2	CInt	Nuno Horta	2022/2023	

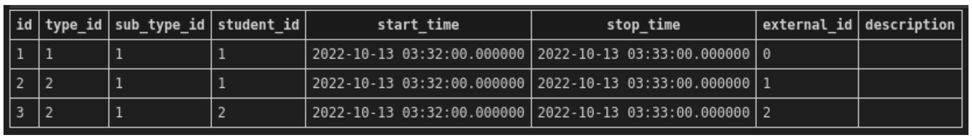
Figure 8: Courses Database example

Name	Type	Description
id	Integer	Course unique identifier
name	String	Name of the course
professor	String	Name of the Professor lecturing the course
school_year	String	Scholar year when the course is being offered
description	String	Description of what will be lectured in the course

Table 3: Courses Database Structure

2.2.4 ActivitiesDB

The ActivitiesDB has eight columns: **id**, **type_id**, **sub_type_id**, **student_id**, **start_time**, **stop_time**, **external_id** and **description**, which are described in Table 4. In Figure 9 it is possible to see an example of the data stored in this database.



id	type_id	sub_type_id	student_id	start_time	stop_time	external_id	description
1	1	1	1	2022-10-13 03:32:00.000000	2022-10-13 03:33:00.000000	0	
2	2	1	1	2022-10-13 03:32:00.000000	2022-10-13 03:33:00.000000	1	
3	2	1	2	2022-10-13 03:32:00.000000	2022-10-13 03:33:00.000000	2	

Figure 9: Activities Database example

Name	Type	Description
id	Integer	Activity unique identifier
type_id	Integer	Identifier of the group of the activity
sub_type_id	Integer	Identifier of the sub group of the activity
student_id	Integer	Student unique identifier to act as foreign key
start_time	DateTime	Date and time of when the student started the activity
stop_time	DateTime	Date and time of when the student finished the activity
external_id	Integer	External DB unique identifier to act as foreign key
description	String	Description of the activity being done

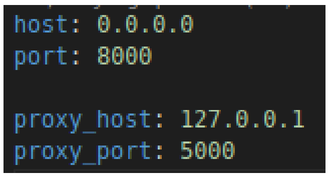
Table 4: Activities Database Structure

2.3 Web App Interface

Finally, in order for the administrator to be able to interact with the information available, there was the need to implement a Web App, which only communicates with the proxy.

Since this Web App is the entry point to our system,we tried to make it as user friendly as possible. This way, everything is pretty intuitive and, even thought we did not use any UI/UX rule, we think that without minimal interaction with the platform, the administrators will easily understand and know how to use it.

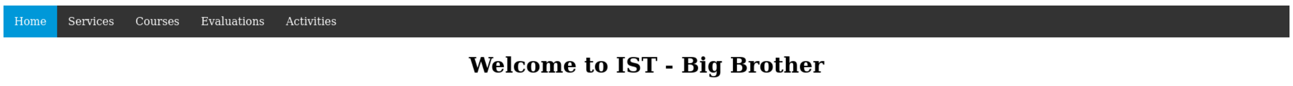
Once more, it is important to mention the existence of a YAML configuration file, which is mandatory and has a fixed structure. This file can be seen in Figure 10.



```
host: 0.0.0.0
port: 8000

proxy_host: 127.0.0.1
proxy_port: 5000
```

Figure 10: Web App configuration file

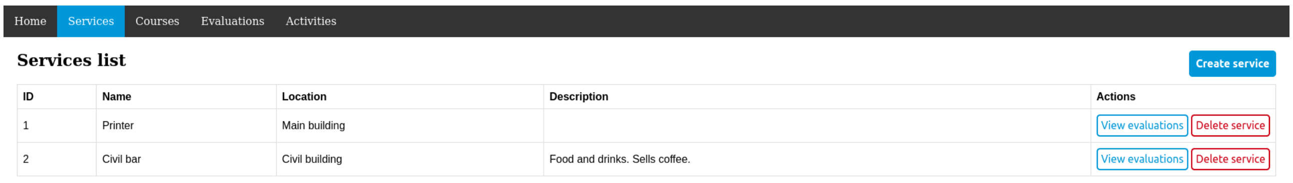
As it can be seen in Figure 11, our home page has the title of the project (IST - Big Brother) and there is navigation bar, which is present in all pages, so it is easy to change to the page that we want to interact with.



Welcome to IST - Big Brother

Figure 11: Web App home page

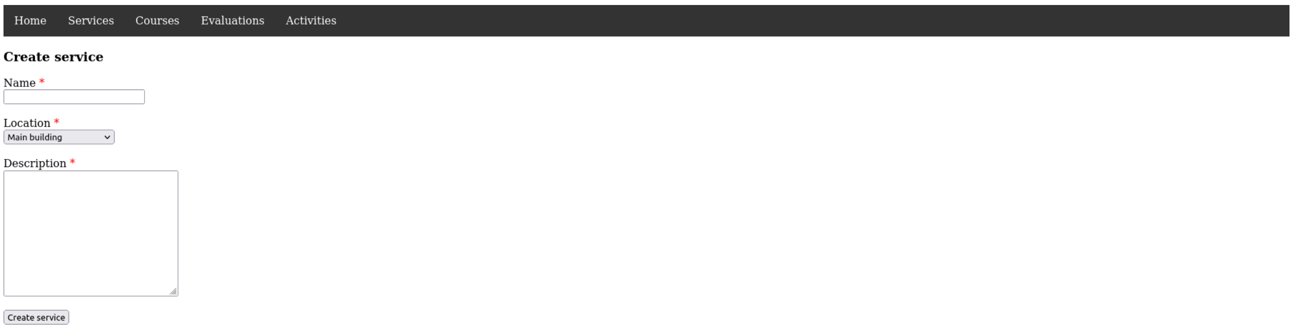
Bellow it is possible to observe all the available pages on the administrative portal. Each page corresponds to the list of all the items of a certain Database, for example, when we go to the Course page, Figure 14, we get the list of all the courses, with all the necessary information about it. Note that, all the pages follow the same design, as previously mentioned.



Services list

ID	Name	Location	Description	Actions
1	Printer	Main building		View evaluations Delete service
2	Civil bar	Civil building	Food and drinks. Sells coffee.	View evaluations Delete service

Figure 12: Web App list services page



Create service

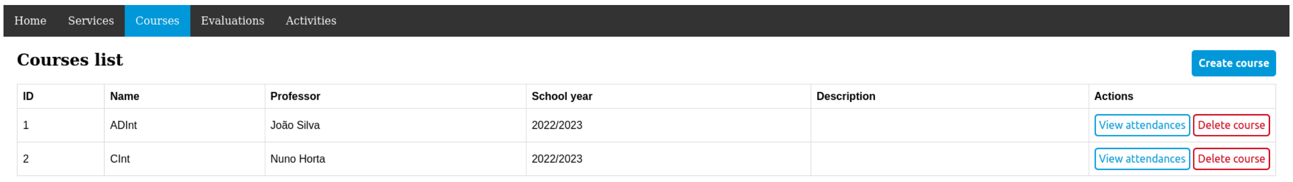
Name *

Location *

Main building

Description *

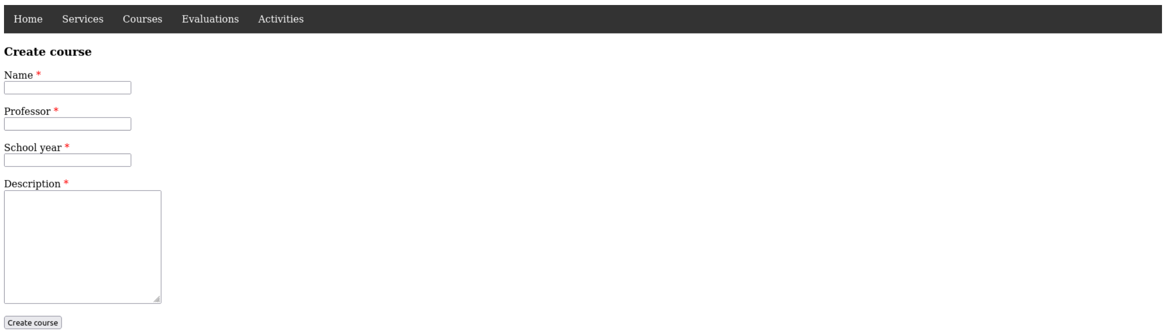
Figure 13: Web App create service page



Courses list

ID	Name	Professor	School year	Description	Actions
1	ADInt	João Silva	2022/2023		View attendances Delete course
2	CInt	Nuno Horta	2022/2023		View attendances Delete course

Figure 14: Web App list courses page



Create course

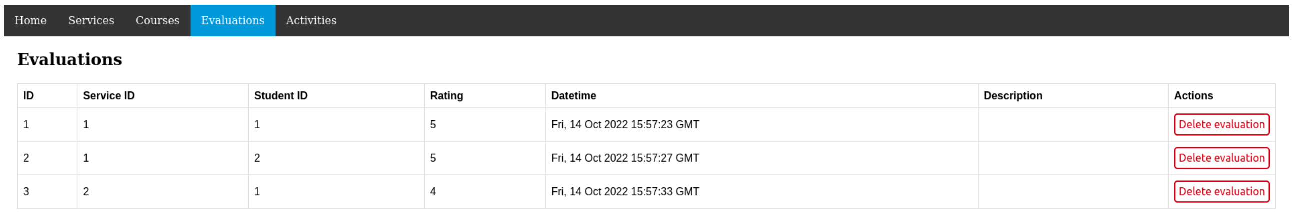
Name *

Professor *

School year *

Description *

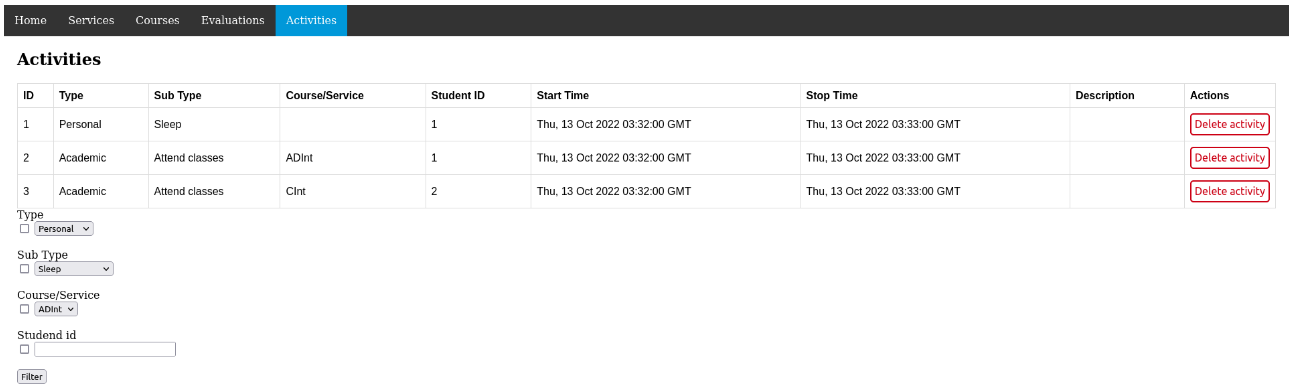
Figure 15: Web App create course page



Evaluations

ID	Service ID	Student ID	Rating	Datetime	Description	Actions
1	1	1	5	Fri, 14 Oct 2022 15:57:23 GMT		Delete evaluation
2	1	2	5	Fri, 14 Oct 2022 15:57:27 GMT		Delete evaluation
3	2	1	4	Fri, 14 Oct 2022 15:57:33 GMT		Delete evaluation

Figure 16: Web App list evaluations page



Activities

ID	Type	Sub Type	Course/Service	Student ID	Start Time	Stop Time	Description	Actions
1	Personal	Sleep		1	Thu, 13 Oct 2022 03:32:00 GMT	Thu, 13 Oct 2022 03:33:00 GMT		Delete activity
2	Academic	Attend classes	ADInt	1	Thu, 13 Oct 2022 03:32:00 GMT	Thu, 13 Oct 2022 03:33:00 GMT		Delete activity
3	Academic	Attend classes	CInt	2	Thu, 13 Oct 2022 03:32:00 GMT	Thu, 13 Oct 2022 03:33:00 GMT		Delete activity

Type: Personal
Sub Type: Sleep
Course/Service: ADInt
Studend id:
Filter

Figure 17: Web App list activities page

It is also important to mention that, in case the user inserts a URI that does not exist, the user will get redirect to home page. Also, if a connection to the proxy occurs, the user will be redirect to the page shown in Figure 18.

Something seems to be offline!

Figure 18: Web App offline page

3 Conclusion

In conclusion, the system developed is able to be scaled easily, since if the need to create a new service arises, the programmer can just add it to the code following the pre-established pattern (which is currently equal to all services). Also, if any of the blocks of the system change their host or port, the programmer just needs to update the YAML configuration files with the new values.

For now, the web app only has the Administrative related components, which allows to, among other functionalities, to create a presidential service, list all evaluations of a service, list all attendances of a course and list all activities of a student (which in this scenario is still an imaginary student, since we are neither connected to the FENIX API or have a student database). The proxy, on the other hand, allows for almost every type of interaction, excluding the ones requiring students information or FENIX authentication methods.