UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Web Applications A.Y. 2022-2023

Homework 1 – Server-side Design and Development

Master Degree in Computer Engineering
Master Degree in Cybersecurity
Master Degree in ICT for Internet and Multimedia

Deadline: 28 April, 2023

Group Acronym	**ESSENTLS**	
Last Name	**First Name**	**Badge Number**
Borsato	Alessandro	2089108
Campagnol	Andrea	2091178
Cardillo	Vittorio	2091429
Lenartavicius	Vaidas	2092135
Maglie	Mattia	2095330
Marcato	Francesco	2082155
Pallante	Laura	2092566
Talukder	Md Imran Faruck	2041440
Villani	Matteo	2090299
Zago	Giovanni	2087645

1 Objectives

This project is intended to be a Web App that will be used by ESN Padova, an active section of ESN Italia. It would help them to streamline and facilitate tasks of registering members and maintaining a membership list different every year (currently made by volunteers). The use of a web application would then likely increase the visibility of its events to international students and make it easier to register for events organized by ESN Padua, just from their smartphones.
Furthermore, that allows volunteers to manage events and Erasmus students, since there is the opportunity to join some events only if you're a member of ESN Padova (that corresponds to getting an ESN card and paying a fee).
At the moment, the subscription is made with a paper form, the payment is by cash or POS and the overall database is a spreadsheet. Events are managed by a communication manager, who posts them on Instagram, the website, and the Facebook group. Mainly former platforms.
As a result, this is a very time-consuming process, that requires a lot of effort from volunteers and, what is more, it is not very efficient.
The overall goal is to facilitate the process and build a stronger database.

2 Main Functionalities

The main functionalities are intended to facilitate users and event management, besides a stronger database. Above is the list of functionalities asked by a group member, who's also an active volunteer in ESN Padua. After discussing them with the group, we listed these goals for the web app.

2.1 Database

The main idea and use of the above-presented Web Application are to provide a reliable management system to ESN Padova.
In fact, at the moment to manage all the ESN Padova members the association uses a Google Spreadsheet, where they're inserted through a Google Form.
This procedure, besides not being reliable and not very agile to changes, may be subjected to input errors and several works every time some specific list (i.e. participant list to a specific event).
A centralized DB (i.e. PostgreSQL) and a user-friendly interface where Erasmus students and International people can autonomously become members can reach the goal of slimming down the work done by the volunteers to manage and keep this spreadsheet updated.
As a second but welcome goal, ESN Padova is also looking for a platform where members can see and pay for events.
In fact, at the moment the main place where members get to know about an event is through social networks or the website, where it may not be easy to navigate given the amount of information on the website.

2.2 User Management

Erasmus and International students should be able to register and create a new account autonomously through the Web Application. Once they confirmed their email, they are assigned a tier 0 user.
Tier 0 users have limited users and they can only see and participate in certain events that are open and available for everyone and don't require any kind of registration.
Tier 0 users are not to be considered members since they don't possess and paid for an ESN card (association card), and they didn't insert all the information necessary to be considered a member.
A Tier 0 has to possibility to become a Tier 1 user and become a member of the section once they/she fills in the subscription form and pay the card fee.
Once they did what is above, they can come to the ESN office with a precompiled document that needs a signature, and they will receive their ESN card.
Tier 1 users are, in a nutshell, all the Erasmus and International people that are a member of ESN Padova and can

participate in all the event available for them.
Tier 1 users can indeed see all the events, participate and pay the fee (if needed) through the use of the application.

2.3 Event Management

The Web Application should have an Event section where Erasmus and International students can easily navigate and see what are the next available events, see the details and participate if they want to.
To create, manage, and easily consult events that ESN Padova organizes, we need to define also other two types of Users, in particular, Tier 2 and 3 users.
Tier 2 users are active ESN volunteers, and as one they can create, and manage their events and provide all the necessary information.
Tier 3 users are admin users that can do all the above with no restriction on visibility.

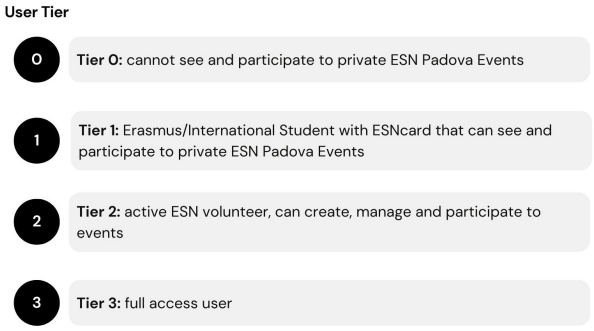

Figure 1: Users tiers.

3 Data Logic Layer

3.1 Entity-Relationship Schema

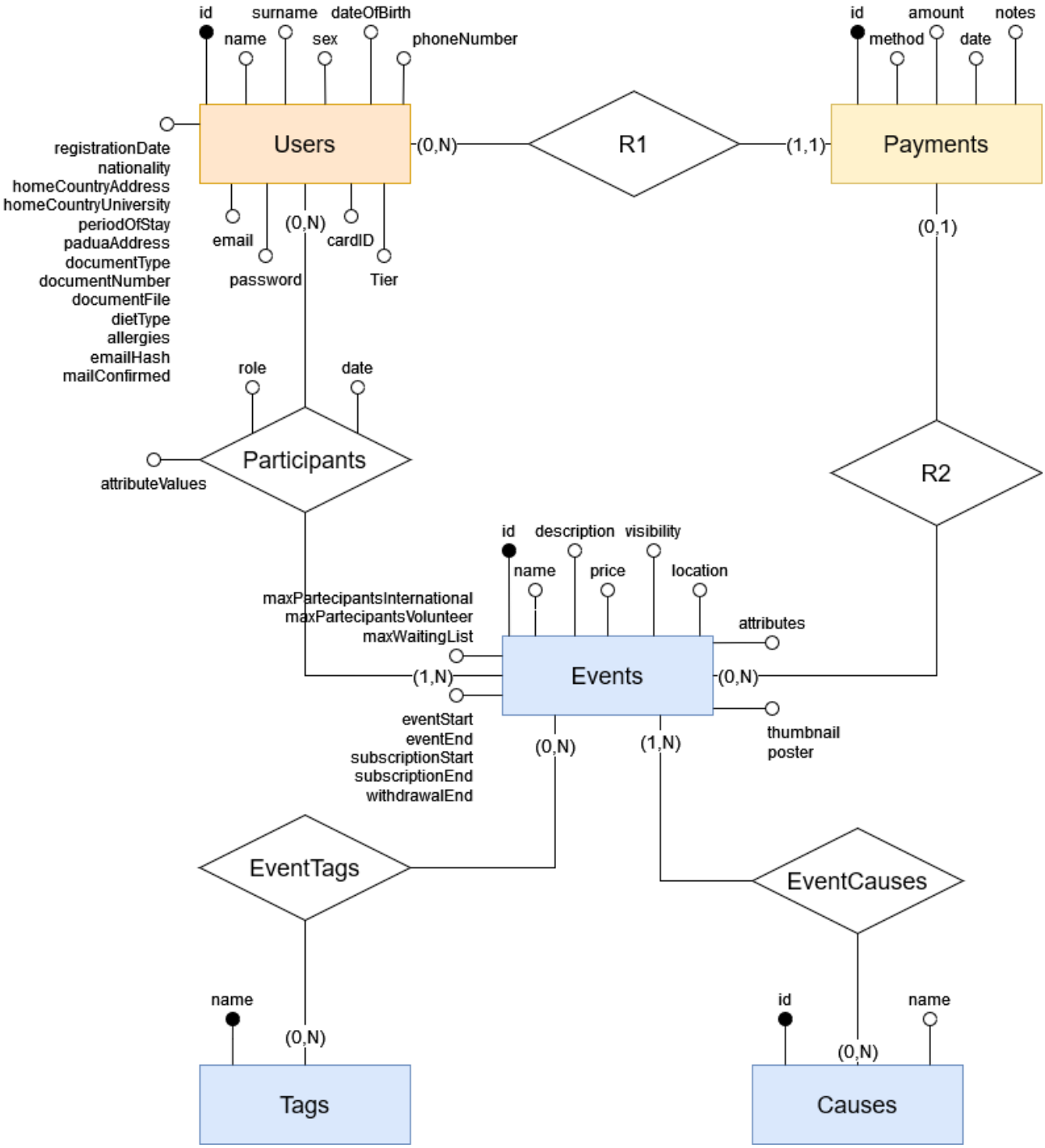

Figure 2: Database ER schema

Our database has 3 main entities, plus an important relation. Others are auxiliary and will be explained shortly at the end. Except for only one entity, all entities have an integer ID as the primary key. The main entities are:

- **Users:** this entity contains all the information about every kind of user, from tier 0 to system admin. This table has new entries every time the registration form is filled: the dummy form will generate a tier 0 user, while the real form will generate a tier 1 user. Every attribute is self-explanatory and stored in a very intuitive way: name and surname are varchar(50), all dates are type date with local time zone, and so on.
  Here, just as examples, some attributes with their SQL code and little information about them.

  ```
  CREATE TYPE diet AS ENUM ('No specific', 'Vegetarian',
                            'Vegan', 'Halal', 'Kosher', 'Pescetarian');

  CREATE TABLE public."Users" (
  id SERIAL PRIMARY KEY,
  email VARCHAR(255) NOT NULL UNIQUE CHECK
          (email ~* '^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$'),
  ...
  "homeCountryAddress" json,
  ...
  "documentFile" text,
  "dietType" diet,
  )
  ```

  In the above snippet, we can see a few special features: RegExpr is used to check mail spelling, addresses, and similar variables are stored as json, for easier data access. Diet and others are stored as enum, to avoid user typos and have more readable attributes.

- **Events:** as for users entity, event has a long list of attributes, wisely chosen about how events are organized. Some events are restricted to certain users, based on tiers, or have a limited number of participants. All these particularities are fully represented in the entity.

- **Payment:** this entity contains all the information about payment for event participation. Of course the actual web application isn't linked to real payment methods, but once it will be, this Entity will be related to real payments, in order to trace them, know most favorite payment methods, etc.

- **Participants:** opposed to previous entities, this is a relation between users and events, collecting user role, as listed in this snippet:

  ```
  CREATE TYPE roleTypes AS ENUM ('Organizer', 'Participant',
                                 'Volunteer', 'WaitingList');
  ```

  Furthermore, date helps managing waiting list and attributes.

3.2 Other Information

Here is some information about minor entities and relations.
Tags and **Causes** are two entities that are used to classify events. Every ESN event supports at least one cause, such as rising founds, meeting freshman, etc. and can be tagged as school, party, cinema, etc. These two entities are used to filter events, to help users find what fits their interests.
EventTag, **EventCauses**, **R1** and **R2** are relations without attributes, used only to join tables.

4 Presentation Logic Layer

Our Web application is mainly developed in eight pages; not all of them are accessible by all users, it depends on their tier (see 1). Also page functionalities depend on tier, i.e. when searching for an event, tier 1 users can see only events for tier 1 or lower, tier 2 users can see events for tier 2 or lower, and so on. More restrictions are made to users that haven't confirmed their mail yet.
Here above most important pages are listed, some are missing, i.e. login, signup pages, or the form to complete the registration.

- `Home`: it's the landing page, when a user is logged it shows a list of events and a search bar. Otherwise, it shows the login page. It's accessible to all users.
- `Profile`: it shows the user's profile info and allows them to change them. It's accessible to all users.
- `My events`: it shows a list of events the user is participating in. It's accessible to all users.
- `Logout`: it allows to logout. It's accessible to all users.
- `(Admin) Create Event`: it allows one to create a new event and eventually change some information about them. It's accessible by tier 2 or above users.
- `(Admin) Search User`: it allows one to search for a user and eventually modify his profile. It's accessible by administrators.
- `(Admin) Create cause`: it allows one to create a new cause. It's accessible by administrators.
- `(Admin) Search cause`: it allows one to search for a cause and eventually modify or delete it. It's accessible by administrators.

4.1 Signup page

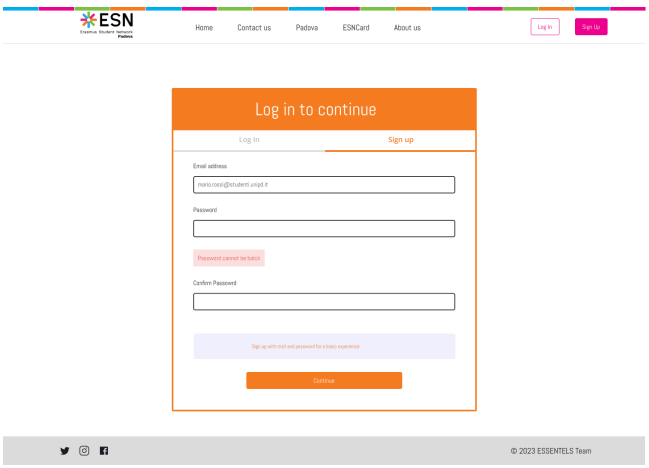

Figure 3: Sign up page

Users can sign up just with their mail and password. They will have access only to tier 0 events. For a better experience and more opportunities, they are asked to complete their profile with a more detailed form, including those from their hometown. This form is shown in 4 and described in next section.

4.2 Registration form

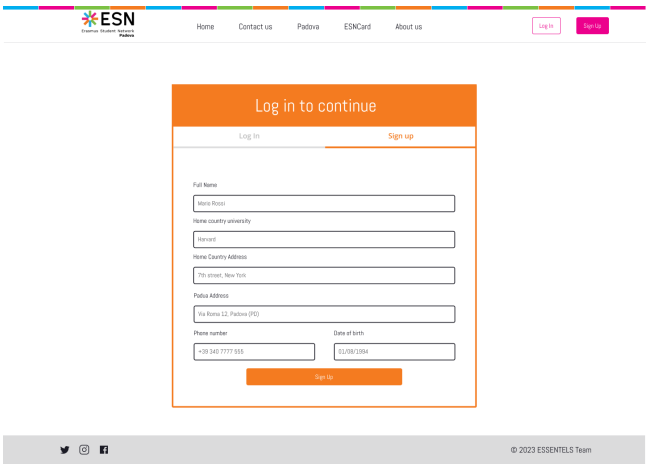

Figure 4: Form for the full registration

When user wants to upgrade to tier 1 they have to pay their subscription (as shown in 12). The form for a full profile asks for any information, here just a few are displayed, but the real form will have more than 10 input boxes.

4.3 Payment page

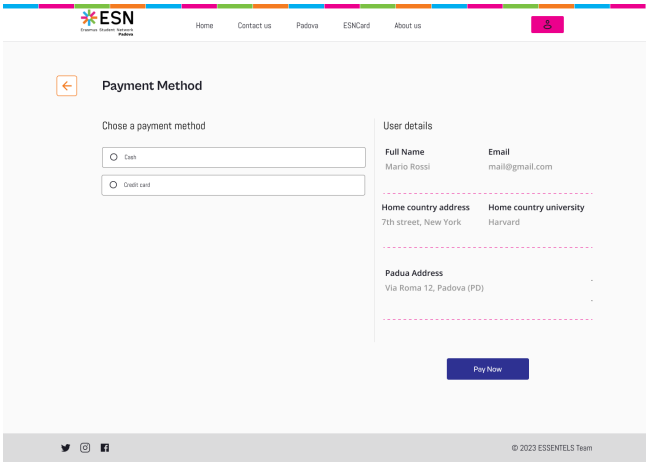

Figure 5: Payment page

When a user pays his subscription, they will become a tier 1 user. They will be able to participate in more events. Still, they cannot create an event (only tier 3 or above users can do that).

4.4 Events list page

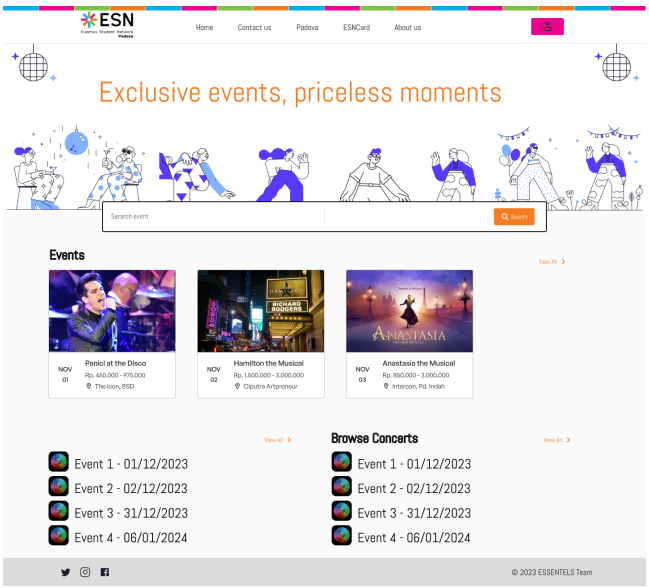

Figure 6: Sign up page

Here all events are listed or filtered by tag or cause. Users are allowed to see only events for their tier or lower ones. Since all events have a location, we decided "according to ESN volunteers" to keep events automatically filtered by tier.
Searching by tag or by cause is actually made by a REST API: using GET, events are listed, and an administrator can also use DELETE to delete an event. With a POST request, a new event can be created.

4.5 Join event page

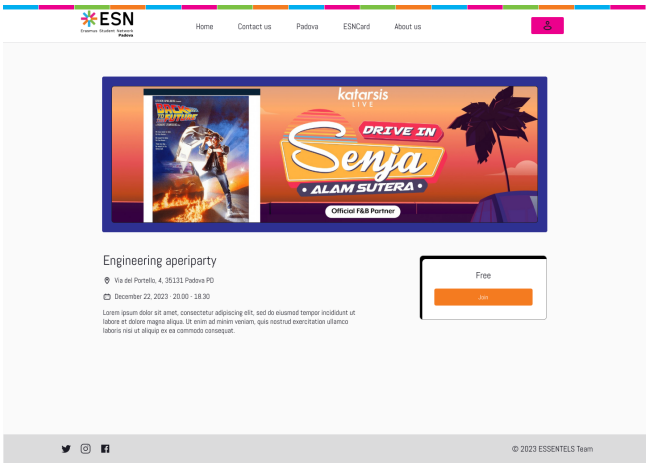

Figure 7: Sign up page

When a user wants to join an event, they can see all the details and the list of participants. Here they can also see more detailed info about the event and confirm his participation. In case some events will ask for a ticket one day, this page will redirect to a payment page, similar to 12, but linked to tickets for cinemas, parties etc.

4.6 Change user info

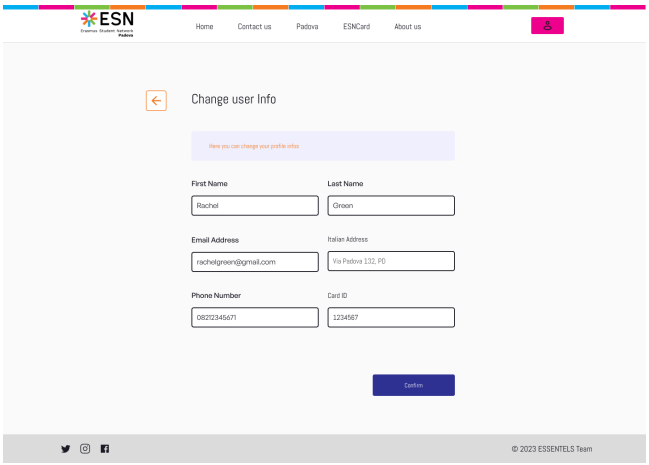

Figure 8: Change user's info

Users can change their profile info. This page can be accessed by each user for his own profile or by an admin for any user.

5 Business Logic Layer

5.1 Class Diagram

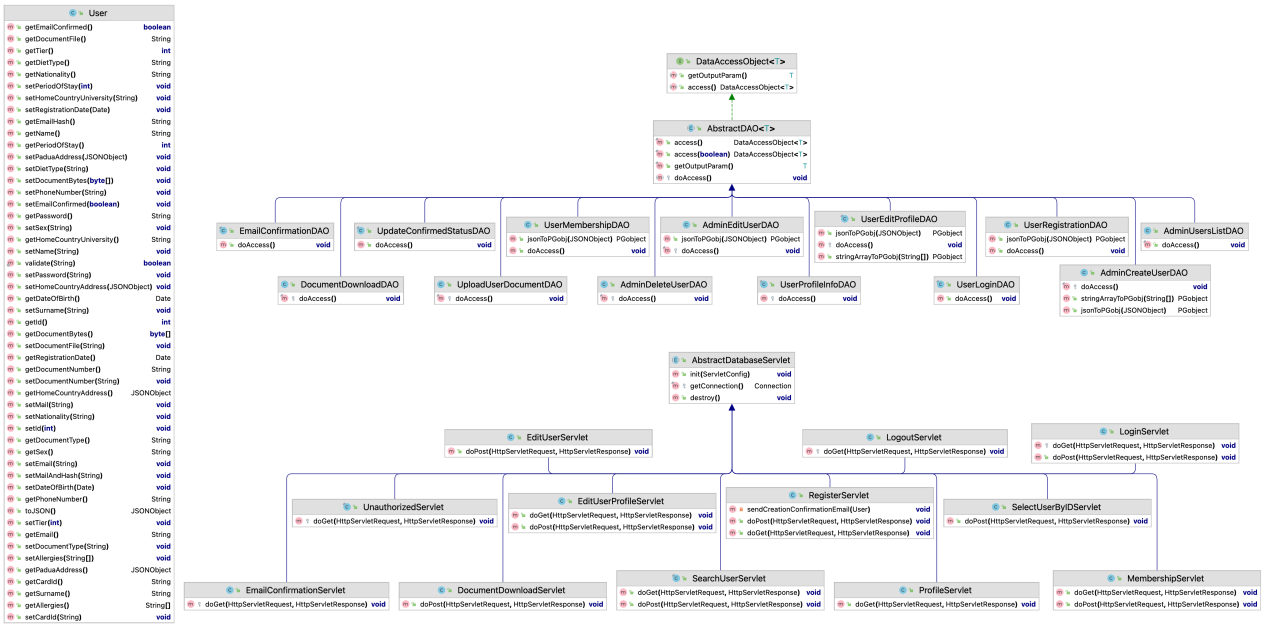

Figure 9: User class diagram

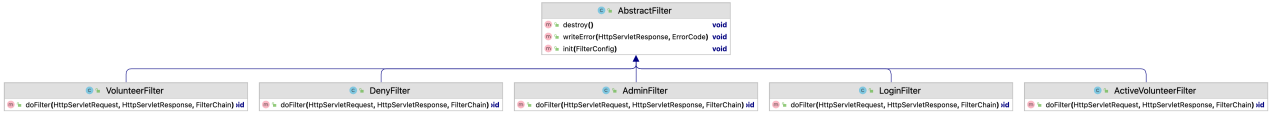

Figure 10: Filters class diagram

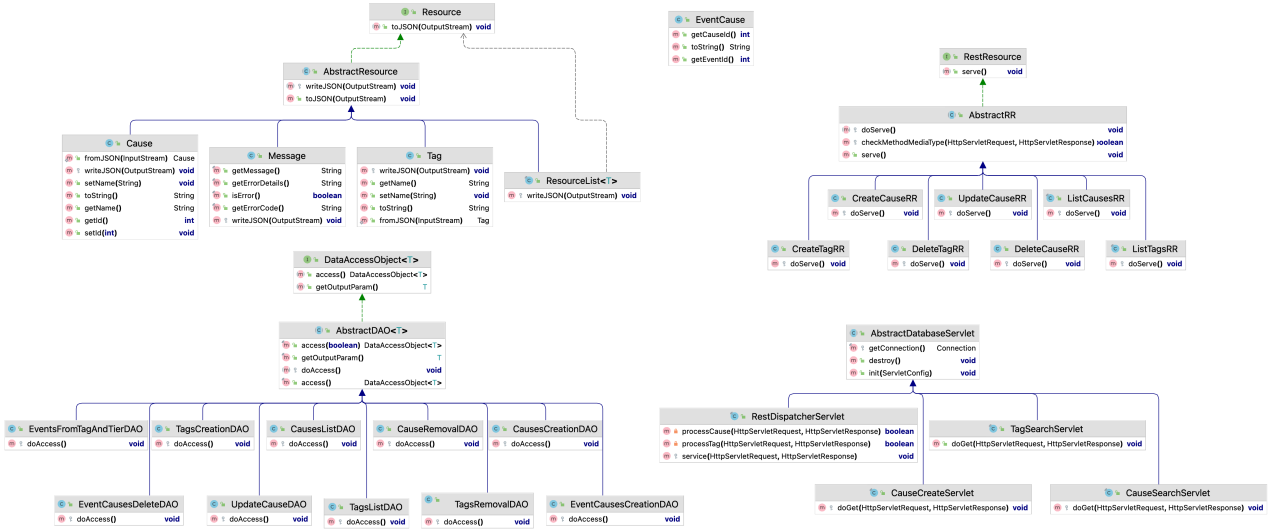

Figure 11: Rest class diagram

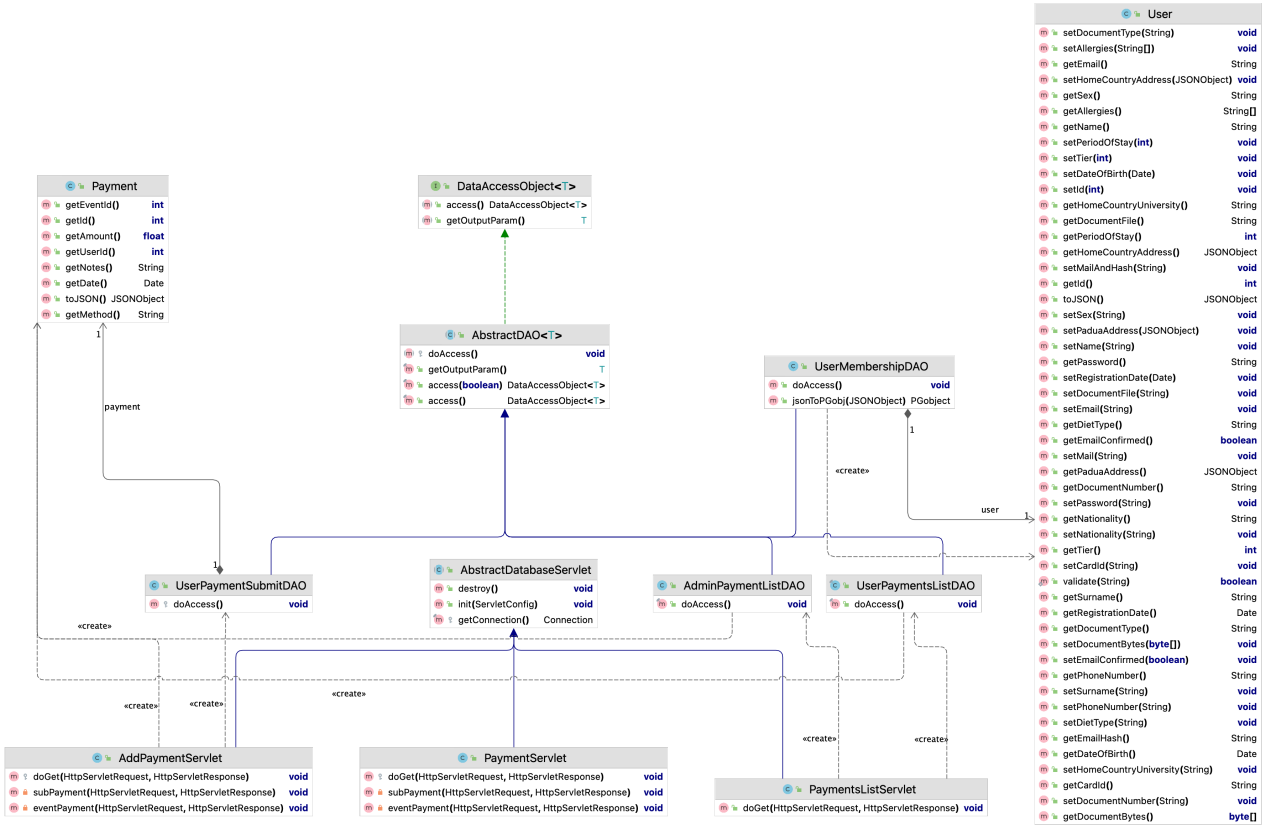

Figure 12: Payment class diagram

To make it as clear as possible, we have divided the class diagram into 3 sections (which do not represent all the classes in the application). The sections were selected based on the entities presented in the data layer.
All twenty-seven servlets that make up the application extend AbstractDatabaseServlet.The role of the abstract servlet is to provide a template for managing the connection pool to the database and extends HttpServlet, thus making all servlets required to override at least one of the following methods: doGet, doPost, doPut, doDelete, getServletInfo, init and destroy.
The servlets can access the database through specific DAOs, which separate the application logic from the database access logic, acting as an intermediary between the application and the database.
Regarding filters 10, they extend the abstract class AbstractFilter, which defines methods for initializing, processing, and destroying requests and responses in a web application.

5.2 Sequence Diagram

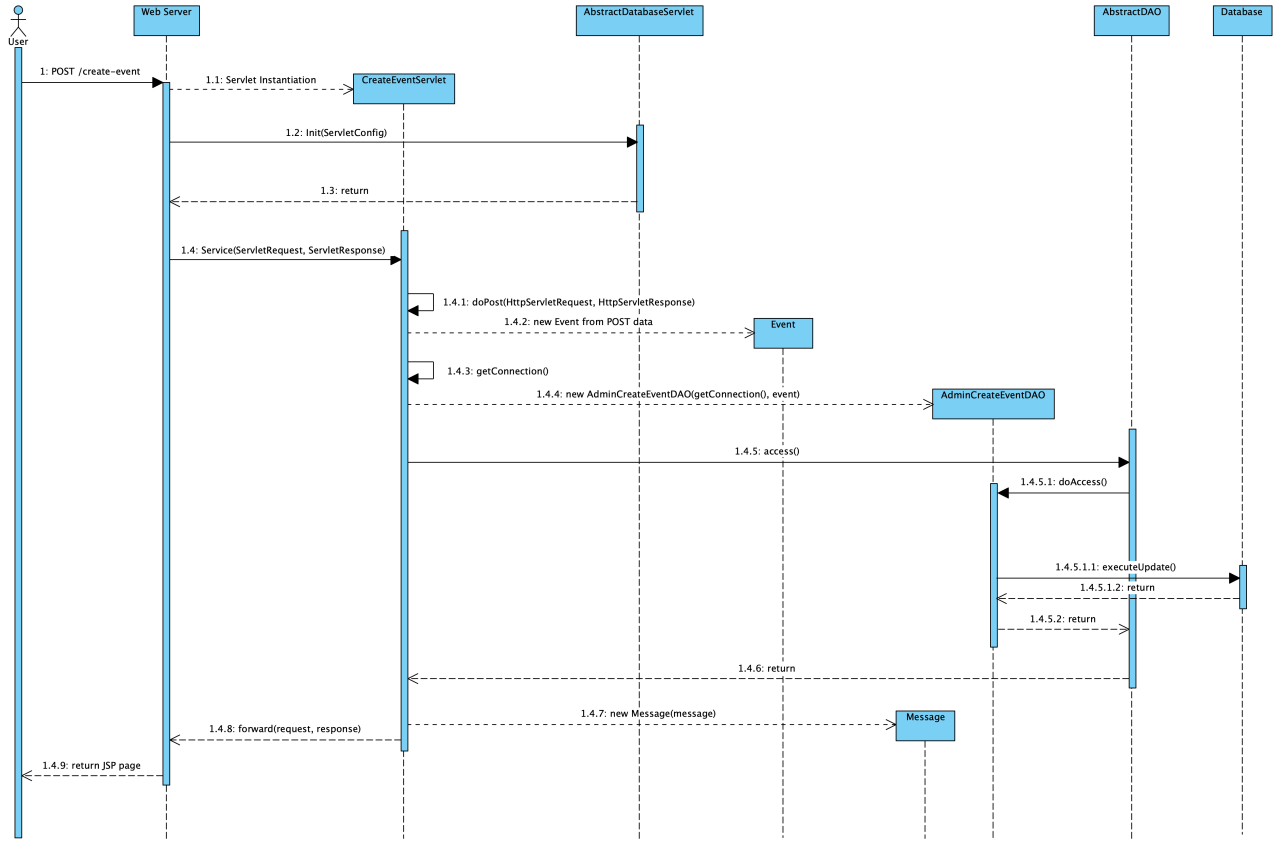

Figure 13: Create Event Sequence Diagram

Here reported the sequence diagram for the event creation operation. To create a new event, an Admin user (at least tier 3), fills the creation form in the Create Event page. By clicking on Continue button, a POST request is issued to the web server, specifying the URI: /create-event.
The data passed to the web server are name, description, price, visibility, place (city, street and number), maximum number of international participants, the maximum number of volunteer participants, start date, end date, start subscription date, end subscription date, withdrawal end date, size of waiting list, attributes, thumbnail image and poster image.

The web server instantiates the CreateEventServlet which calls its doPost method, passing the HttpServletRequest and HttpServletResponse. This method handles some POST data packing city, street and number in a unique location JSON Object and storing the two images (thumbnail and poster) in the "ESSENTLS_Cloud" folder. A new Event object is created, containing the other POST data, the generated location JSON object and the paths that represent the images. The control is passed to the AdminCreateEventDAO which receives as arguments the connection (defined in the AbstractDatabaseServlet extended by the CreateEventServlet) and the instantiated Event. The AdminCreateEventDAO (that extends the AbstractDAO) contacts the Database Server with the access() method that calls the doAccess() override method in the AdminCreateEventDAO, which executes the SQL statement for event creation. The operation ends by returning a message that informs the user that the event is successfully created.

5.3 REST API Summary

For this homework, we implemented REST APIs on the paths "/rest/causes/" and "/rest/tags/" for the Causes and Tags resources. These resources are used as properties of the Events resource to facilitate retrieval and help categorization.
Specifically the REST API is used in the "/tag-search", "/cause-search" and "/cause-create" pages accessible to administrators to manage these resources. To match future development needs or just uniform the methods of information retrieval between pages we will probably extend the area of coverage of REST APIs. The following table shows the structure we followed to implement "/rest/causes/".

URI	Method	Description	Filter
/rest/causes/	GET	It returns a list of all Causes in the database	Behind AdminFilter (only accessible to tier 4 users)
/rest/causes/id/*id*	GET	It returns the Cause with the corresponding *id*, if *id* is empty it returns all Causes	Behind AdminFilter (only accessible to tier 4 users)
/rest/causes/srch/*subCause*	GET	It returns the Causes that contain the string *subCause* in the name field	Behind AdminFilter (only accessible to tier 4 users)
/rest/causes/	POST	It creates a new Cause contained in the request, inserts it into the database and returns it	Behind AdminFilter (only accessible to tier 4 users)
/rest/causes/id/*id*	PUT	It updates the Cause with the corresponding id with the values contained in the request and returns it	Behind AdminFilter (only accessible to tier 4 users)
/rest/causes/id/*id*	DELETE	It deletes the Cause with the corresponding id and returns it	Behind AdminFilter (only accessible to tier 4 users)

Table 2: REST API for the Causes resource

5.4 REST Error Codes

Error Code	HTTP Status Code	Description
E4A1	400 Bad Request	Output media type not specified
E4A2	406 Not Acceptable	Unsupported output media type
E4A3	400 Bad Request	Input Media type not specified
E4A4	415 Unsupported Media Type	Unsupported input media type
E4A5	405 Method Not Allowed	Unsupported Operation
E4A6	404 Not Found	Unknown resource requested
E4A7	400 Bad Request	Wrong URI format
E4A8	400 Bad Request	Wrong resource provided
E5A1	500 Internal Server Error	Unexpected error while processing the resource
E5A3	404 Not Found	Resource not found
E5A4	409 Conflict	Cannot modify resource because other resources depend on it

Table 3: REST API error codes for Causes resource

5.5 REST API Details

The following is the detailed description of one of the resources of our REST API, the resource used for searching Causes.

Search Causes

Returns JSON data about all Causes that contain a certain substring in the Name attribute

- URL: `/causes/srch/*subCause*`
- Method: `GET`
- URL Parameters:
  `Required: subCause=[String]`
- Data Parameters:
  `None`
- Success Response:
  `Code: 200`
  `Content: { esncause: {id : 1, name : "examplecause"}}`
- Error Response:

```
Code:400
Content: { error: "Cannot search Cause(s): wrong format for URI"}
OR
Code:404
Content: { error: "Unknown resource."}
OR
Code:405
Content: { error: "Unsupported method for URI /causes/srch/: ..."}
```

```
OR
Code:500
Content: { error: "Cannot search Cause(s): unexpected error."}
```

- Sample Call:

```
$.ajax({
    url: "/causes/srch/examplecause",
    datatype: "json",
    type : "GET",
    success : function(r) {
        console.log(r);
    }
});
```

6 Group Members Contribution

Everyone has contributed to the project by sharing ideas, suggestions and fixing issues proactively. We all took part in the analyses process and decided to split the development of some parts of the project equally (DAOs, Servlets, jsps, ...).

Alessandro Borsato has developed part of the registration process, and proposed a common approach to managing the project (ticketing and git) and wrote internal guides to let everyone aligned.

Andrea Campagnol has contributed to this project by taking care of some initial setups and developing the login and mail verification parts. He also draws the main schemas present in this document.

Vittorio Cardillo has helped the team in the initial setup and developed some key parts like the homepage. Has also made the first graphic design of the application and implementation of resources.

Vaidas Lenartavicius contributed to the requirements analysis stage, he developed some DAOs regarding user and participant resources, servlets, jsp pages, js scripts and other parts of this project. He was always available to support team members in case of fixing bugs and other problems.

Mattia Maglie helped during the design of the project with requirements, definitions of DAOs and Entity-Relationship diagram. Developed some DAOs regarding events, and servlets with respective jsp. He took care of the payment section of the app and developed attribute insertion logic and filters. Also, he suggested some useful refactoring of the code and best practices.

Francesco Marcato has proposed the idea for this application, as a result of a practical need in Padua ESN management. He wrote the requirements of this application and wrote code mainly for the events part.

Laura Pallante contributed to the requirements analysis stage, she implemented DAOs and some key pages, and structured most of the REST architecture for the project. She was also helpful in case of fixing some issues.

Md Imran Faruck Talukder has contributed to the project by taking care of the database implementation and also helping teammates set up the connection between the application and db.

Matteo Villani has developed some filter parts and edit pages. He took care of aligning the style of the app pages.

Giovanni Zago has set up the Trello application we used to divide the work among us, also providing a simple tutorial on how to use it properly. He took care of writing most of the parts in the documentation (including pages mockups) and also some application pages, i.e. editing users information as user or as administrator.