Software Engineering

2022./2023.

Learning Together Platform

Documentation, Rev. *1*

Group: *Team Eder*

Leader: *José Matos*

Date: *18. 11. 2022*

Teacher: *Nikolina Frid*

Contents

1. Documentation change log

Rev.	Change description	Authors	Date
0.1	Functional and non-functional requirements. Project description. All done in a team meeting.	Every member	23/10/2022
0.2	Added use cases and description for each one.	Every member	4/11/2022
0.3	Added the use case diagram after team meeting where it was constructed.	José Matos	15/11/2022
0.4	Added the 4 sequence diagrams.	Lourenço Carvalho, Maria Costa, Francisco Antunes, Rui Guimarães	16/11/2022
0.5	Added Class and Database diagrams.	Lourenço Carvalho	18/11/2022
0.6	Described system architecture and design.	José Matos	18/11/2022
0.7	Final refinement before 1st revision.	Every member	18/11/2022
1.0			

Continued on next page

Continued from previous page

Rev.	Change description	Authors	Date
1.1	Corrected mistakes pointed out in the 1st revision.	José Matos	10/01/2023
1.2	Added Tools and Technologies.	José Matos	11/01/2023
1.3	Added Deployment Diagram, Deployment Instructions, Conclusion and further work.	José Matos	13/01/2023
1.4	Updated the Database tables, the class and ER Diagrams.	Lourenço Carvalho	13/01/2023
1.5	Added Component and System testing.	Lourenço Carvalho	13/01/2023
1.6	Added state machine diagram and review of the Enroll course sequence diagram	Francisco Antunes	13/01/2023
1.7	Added Activity diagram	Rui Guimarães	13/01/2023
2.0			

2. Project assignment description

Learning Together is a new web platform for knowledge exchange. The idea of this platform is that anyone can create their own course on any topic and upload written, video and audio materials. In addition, this platform enables interactive communication with the person holding the course through comments and live chats. An unregistered public user can view the courses that are available, but to access the course, it is necessary to register with an email address. Registered users can browse available courses and enroll in any available course. Some courses may require payment. To create a new course, the following information is required: name, category (choose one of the existing ones in the system), an estimate of how much time the average user needs to master all the materials, price (optional) and the course content. The content of the course is divided into points (teaching units). Each unit, along with a mandatory short description, can contain written, audio and video materials that registered users can download (the materials are not available to users who are not course participants). The course owner can add or remove materials at any time. The owner of the course can enable the option of online chat with participants and indicate the dates and times when it will be available and the maximum number of participants. Interested course participants can register for the offered dates1. Enrolled course participants can comment and rate the course. Comments and ratings are publicly visible (also available to non-registered users). Registered users will have a private and a public profile. Private profile will contain users' personal information, payment details and information about courses taken and/or created. Also, on their private profile, users can select one or more categories of courses that they are interested to get recommendations and updates. A public profile is required only for users who create and offer courses to other users. The application is maintained by system administrators who can add or delete any user, change the category of one of the existing courses and delete any course.

3. Software specification

3.1 Functional requirements

Stakeholders:

1. Students
2. Teachers
3. Team Leaders
4. Developers
5. Administrators
6. Google Calendar API Provider
7. Tawk.to Integration Provider

Actors and their functional requirements:

1. Unregistered users can:
   (a) View available courses
   (b) Register on the website
   (c) See public users profile

1. Registered users can:
   (a) View available courses
   (b) Edit profile
   (c) Interact with courses
      i. Enroll into a course
      ii. Comment and rate a specific course
      iii. Communicate through a live chat with the creator of the course
      iv. Access course materials
   (d) Create and edit their own courses
      i. Add or remove materials at any time
      ii. Set live chat properties

2. System administrators can:

(a) Add or delete any user
(b) Change category of any existing course
(c) Delete any existing course

3.1.1 Use cases

UC1 - View Available Courses

- **Main participant:** Unregistered and Registered Users
- **Goal:** Search for a specific course the user wants to apply or just see the available courses in the website
- **Participants:** Database
- **Prerequisites:** None
- **Description of the basic course:**
  1. In the main page, the website shows to the user some course recommendations
  2. The user can filter the courses by categories
  3. If the user wants to search for a specific course, he can do it directly by writing the course's name
  4. System presents a list of courses with the given input
- **Description of possible deviations:**
  1. The user cannot access the course
     - The user needs to be registered.
  2. The user tries to search for a specific course that doesn't exist in the website

UC2 - See public profile

- **Main participant:** Unregistered and Registered Users
- **Goal:** Users can search for specific public user to find available courses(UC1-View Available Courses)
- **Participants:** ≪Database
- **Prerequisites:** None
- **Description of the basic course:**
  1. Registered and unregistered users can search for public profiles, searching for the public user name, to see specific courses
  2. System presents a list of public profiles according to the input
  3. Users can select another user and view their profile
- **Description of possible deviation:**
  1. Main actor search for a non-existent public user

UC3 - Register

- **Main participant:** Unregistered User
- **Goal:** Having access to a course or create one
- **Participants:** Database
- **Prerequisites:** None
- **Description of the basic course:**
  1. The future user needs to input their name, surname, email, username and password.
  2. Submit the information.
  3. If all of the data is correct, according with the parameters, then the new actor is successfully registered and goes to the initial page of the website (already logged).
- **Description of possible deviations:**
  1. If there is already a username and/or an email equal to the one filled in.
     - System will return a warning saying that the user needs to change that data.
  2. System warns if the password is not in accordance with the security conditions.
     - The user needs to change is password for one that is in conformity with the specifications.
  3. The email is not valid.
     - The website will return an warning message saying that the user has to change the email to one that is valid (with @).

UC4 - Login

- **Main participant:** Registered Users
- **Goal:** Access more functionalities of the website
- **Participants:** Database
- **Prerequisites:** None
- **Description of the basic course:**
  1. The main actor enters their login credentials (username and password)
  2. The login credentials are validated in the database
  3. It's granted access to more functionalities of the website (enroll in a course, create courses, etc...)
- **Description of possible deviations:**

  2.a The login credentials are wrong
  1. The main actor tries to login again with a different username/password

2. The main actor resets the password
3. The main actor cancels the login

UC5 - Edit user's profile

- **Main participant:** Registered user
- **Goal:** Edit profile information
- **Participants:** Database
- **Prerequisites:** User should be registered and logged in (UC3 and UC4)
- **Description of the basic course:**
  1. Main actor selects form to edit profile
  2. Profile information is listed
  3. Main actor selects which information to update
  4. User enters updated information
  5. System allows main actor to save his updated information
- **Description of possible deviations:**

  5.a Main actor tries to exit without saving
  1. System prompts user to allow saving of changes
  2. Main actor exits without saving

UC6 - Enroll course

- **Main participant:** Registered user
- **Goal:** User can enroll a course
- **Participants:** Payment API
- **Prerequisites:** Have an account
- **Description of the basic course:**
  1. The main actor enroll on one or multiple courses
  2. In case of paid course the payment information is validated by the payment API.
  3. System grants the user access to all of that courses functionalities (material, live chat scheduling, etc.)
- **Description of possible deviation:**
  1. The payment details are incorrect/ payment fails (in case of paid course)
  2. Enrollment deadline already expired

UC7 - Rate/Comment on Course

- **Main participant:** Registered Users
- **Goal:** Comment on another user's course
- **Participants:** Database, Registered User
- **Prerequisites:** User must be logged in (UC4), have selected a course to rate/comment on and be enrolled in that same course
- **Description of the basic course:**
  1. The main actor selects the course he wants to rate/comment on
  2. The main actor writes the comment and/or rates the course(on a scale from 0 to 5)
- **Description of possible deviations:**

UC8 - Communicate through a live chat with creator

- **Main participant:** Registered Users
- **Goal:** An user communicate with the creator during a live chat
- **Participants:** Database, Registered User
- **Prerequisites:** User must be logged in (UC4 - Login) and enrolled in that course.
- **Description of the basic course:**
  1. Main actors enters the live chat at a given schedule
  2. System allows for exchanging of messages between the two participants
- **Description of possible deviations:**
  1. The creator does not turn on the live chat
     - The website returns a message saying that the chat will be turned on in a few moments.
     - If 10 minutes have passed the administrator will be contacted and system notifies the user.
  2. The user is not registered in that course.
     - The user is notified that it is not enrolled in the course.
     - The user can enroll in the course and wait to be accepted by its creator.

UC9 - Access course materials

- **Main participant:** Registered Users
- **Goal:** Access materials from a course
- **Participants:** Database

- **Prerequisites:** User must be logged in (UC4 - Login) and must be enrolled in the course (UC6 - Enroll course)
- **Description of the basic course:**
  1. The main actor enters the course page
  2. The main actor chooses the option to access the materials
  3. System lists all the available materials from that course
- **Description of possible deviations:**
  1 There are no materials available yet
  - 1. The main actor can go back to the course page

UC10 - Create course

- **Main participant:** Registered Users
- **Goal:** Create a new course in the website
- **Participants:** Database
- **Prerequisites:** User must be logged in (UC4 - Login) and must to be using his public profile
- **Description of the basic course:**
  1. The user selects the option to create a course
  2. Step by step the user has to input the course's details
     - Course's name
     - Category
     - An estimate of how much time the average user needs to master all the materials
     - Price(optional)
     - A description about the content of the course
  3. Confirm the course creation
  4. System now lists the course as available
- **Description of possible deviations:**
  1. The course name already exists
  2. Going back to the main page before finishing the course creation

UC11 - Edit course

- **Main participant:** Course Owner
- **Goal:** Make changes to the details
- **Participants:** Database

- **Prerequisites:** Course owner must be logged in (UC4 - Login) with a public profile and the course should belong to him
- **Description of the basic course:**
  1. The Main Actor selects his course
  2. The Main Actor selects the option to edit his course
  3. Then the course owner chooses what he wants to change
     - Add materials(UC12)
     - Remove materials
     - Update existent materials
     - Enable/disable the option of online chat
- **Description of possible deviations:**

UC12 - Add Materials

- **Main participant:** Registered User
- **Goal:** Add materials on the page of the course.
- **Participants:** Database
- **Prerequisites:** Must be logged in (UC4- Login) and the creator of the course.
- **Description of the basic course:**
  1. Main Actor selects their course
  2. Main Actor selects the option to add materials
  3. The creator submits the materials and saves the changes
  4. The system returns a success message and all the materials are available to the users that are enrolled in that course
- **Description of possible deviations:**
  1. The attachment is not in an acceptable format
     - The website returns a warning saying that the file is not in a acceptable format. The main actor needs to submit the file in eligible format.
  2. Actor try to upload a file which size is higher than the imposed limit
     - The website returns a warning saying that the file is too large
  3. Creator attempts to leave without saving the changes.
     - The website returns a message asking to submit or discard.

UC13 - Remove Materials

- **Main participant:** Registered User

- **Goal:** Remove materials from the page of the course
- **Participants:** Database, User
- **Prerequisites:** Main actor must be logged in and be the creator of the course. The course needs to have materials already
- **Description of the basic course:**
  1. Main Actor clicks in their course
  2. Main Actor clicks in a button with the name ”Edit Materials”
  3. Main Actor selects which material(s) to remove from the platform
  4. Main Actor confirms the material’s deletion
- **Description of possible deviations:**
  1. Main actor decides not to delete the selected materials when presented with the confirmation prompt

UC14 - Set live chat properties

- **Main participant:** Registered Users
- **Goal:** Change the properties of a course live chat
- **Participants:** Database, Google Calendar API Provider
- **Prerequisites:** User must be logged in (UC4 - Login) and have created the course (UC10 - Create course)
- **Description of the basic course:**
  1. The main actor, creator of the course, enters the course page
  2. The main actor clicks the Definitions
  3. The main actor clicks the Live Chat
  4. All the properties of the course Live Chat are shown and editable (Schedule, Maximum number of users, etc...)
  5. The main actor can change any property
  6. To exit the live chat properties the main actor needs to save or discard the changes
  7. System updates all changes(if saved)
- **Description of possible deviations:**
  1 The main actor discards all changes

2 The schedule is changed

  1. The Google Calendar API Provider is updated
  2. System updates the course page

UC15 - Add User

- **Main participant:** System administrator
- **Goal:** Add new user
- **Participants:** Database
- **Prerequisites:** Must be administrator
- **Description of the basic course:**
  1. Administrators can create new user accounts
  2. System recognizes the creation of the new user
- **Description of possible deviation:**
  1. Administrator creates user with already existing credentials (username)
     (a) System does not allow user with the same username to be created.

UC16 - Delete User

- **Main participant:** System administrator
- **Goal:** Delete user
- **Participants:** Database
- **Prerequisites:** Must be administrator
- **Description of the basic course:**
  1. In case of any violation the administrator can delete an user account
  2. User account is erased from the system
- **Description of possible deviation:**
  1. Administrator tries to delete it's own account
     (a) System does not allow an administrator to delete it's own account

UC17 - Change Course Category

- **Main participant:** System administrator
- **Goal:** Change the category of a course
- **Participants:** Database
- **Prerequisites:**
- **Description of the basic course:**
  1. The main actor enters the course page
  2. The main actor selects course properties
  3. The creator selects Category
  4. All categories present on the website are shown and the main actor can choose which category their course belongs to

5. Main Actor saves the changes
6. The system now lists the course under the new category

- **Description of possible deviations:**
  1. The creator attempts to leave without saving the changes
     - System allows user to save changes. If not saved they are discarded.

UC18 - Delete Course

- **Main participant:** System administrator
- **Goal:** Delete a course
- **Participants:** Database
- **Prerequisites:**
- **Description of the basic course:**
  1. Main Actor accesses his list of courses
  2. The Main Actor selects the course page that he wants to delete
  3. The Main Actor selects the option to delete his course
  4. The course in no longer existent in the system
- **Description of possible deviations:**
  1. The creator attempts to leave without saving the changes
     - System allows user to save changes. If not saved they are discarded.

Use case diagrams

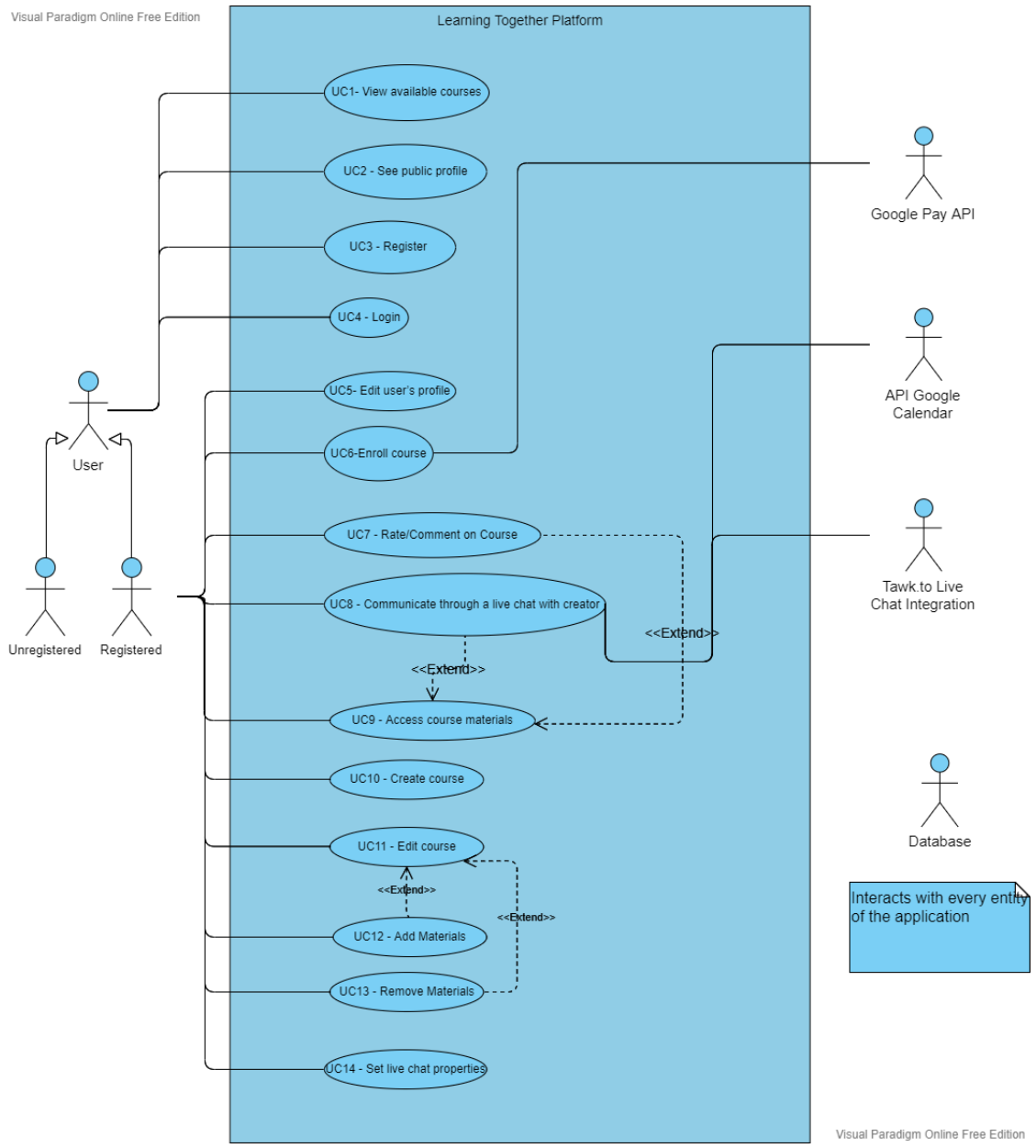

Figure 3.1: Common user use case diagram

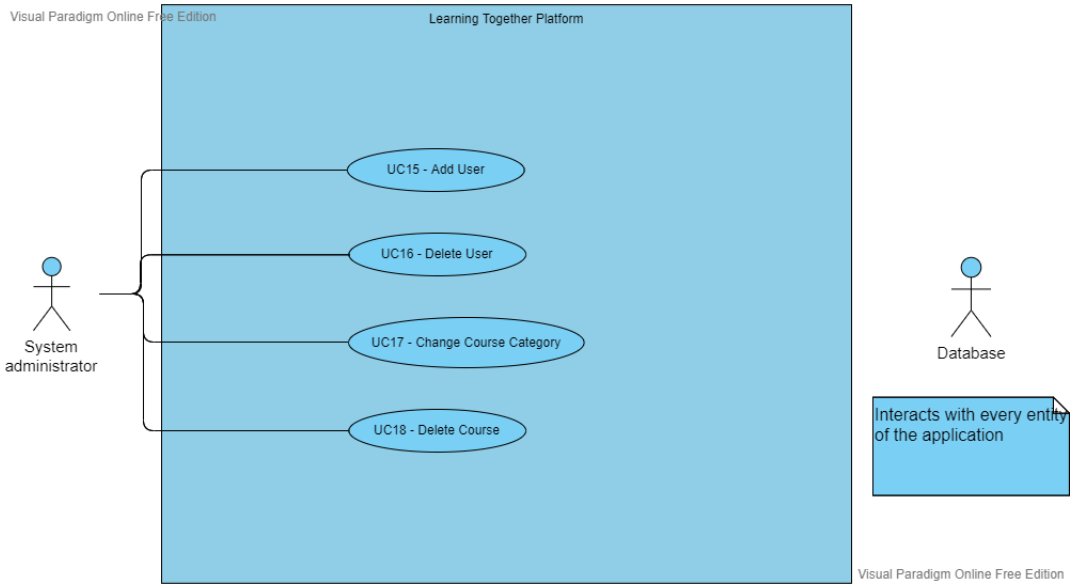

Figure 3.2: System administrator use case diagram

3.1.2 Sequence diagrams

The figure 3.3 represents the sequence diagram for the action of registering an user. The user chooses to register. After this the web application presents them with the register form, where they need to put their name, surname, email, username and password. The system verifies if the e-mail/username already exists in the Database. In such cases where the credentials already exist the web application returns an error message. If the credentials inserted are not valid, the website returns an error message, otherwise the user is successfully registered.

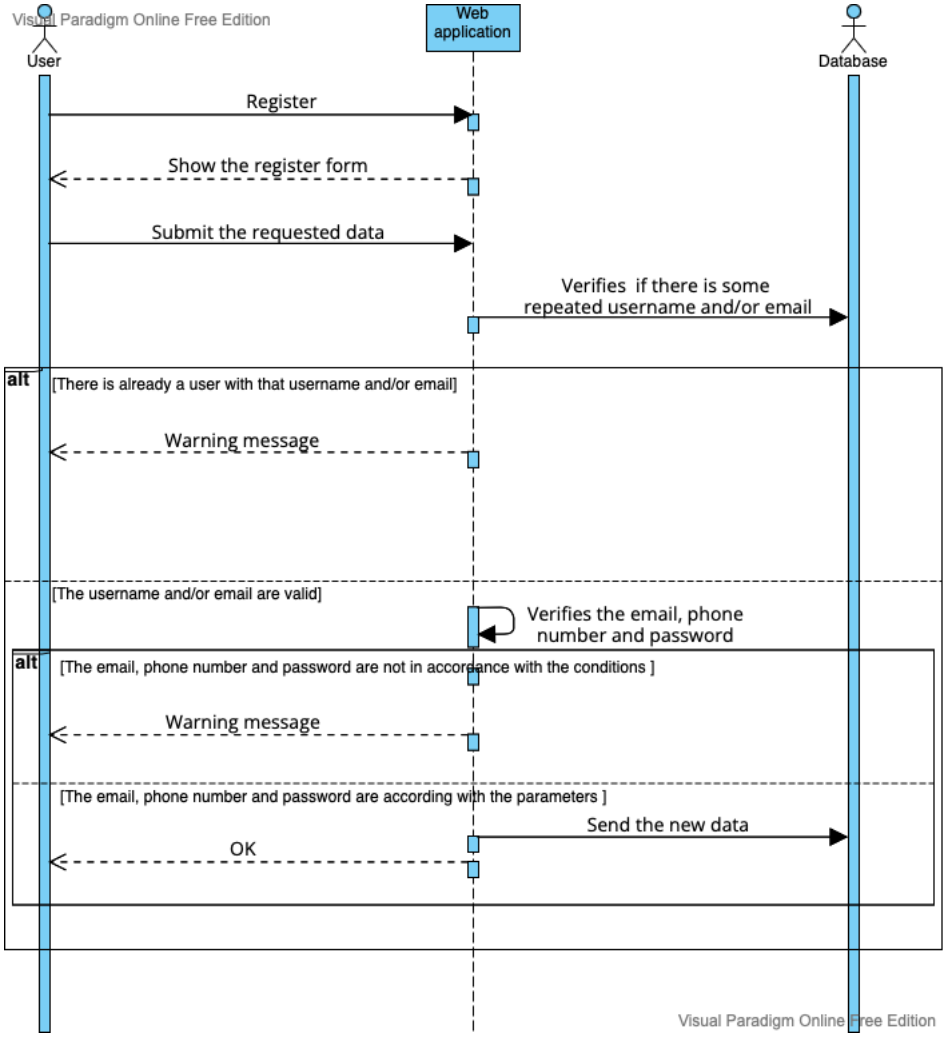

Figure 3.3: Register User Sequence Diagram

The diagram in figure 3.4 represents the enroll course action. First the system will verify if the user exists and retrieve their information from the data base, in case the search is successful. Then in case of the course being paid the system searches for the user's payment information, sending an error message in case of

failure. In case of success, it informs the user that they are enrolled in the course. If the course doesn't require payment the system informs the user that they are enrolled in the course.

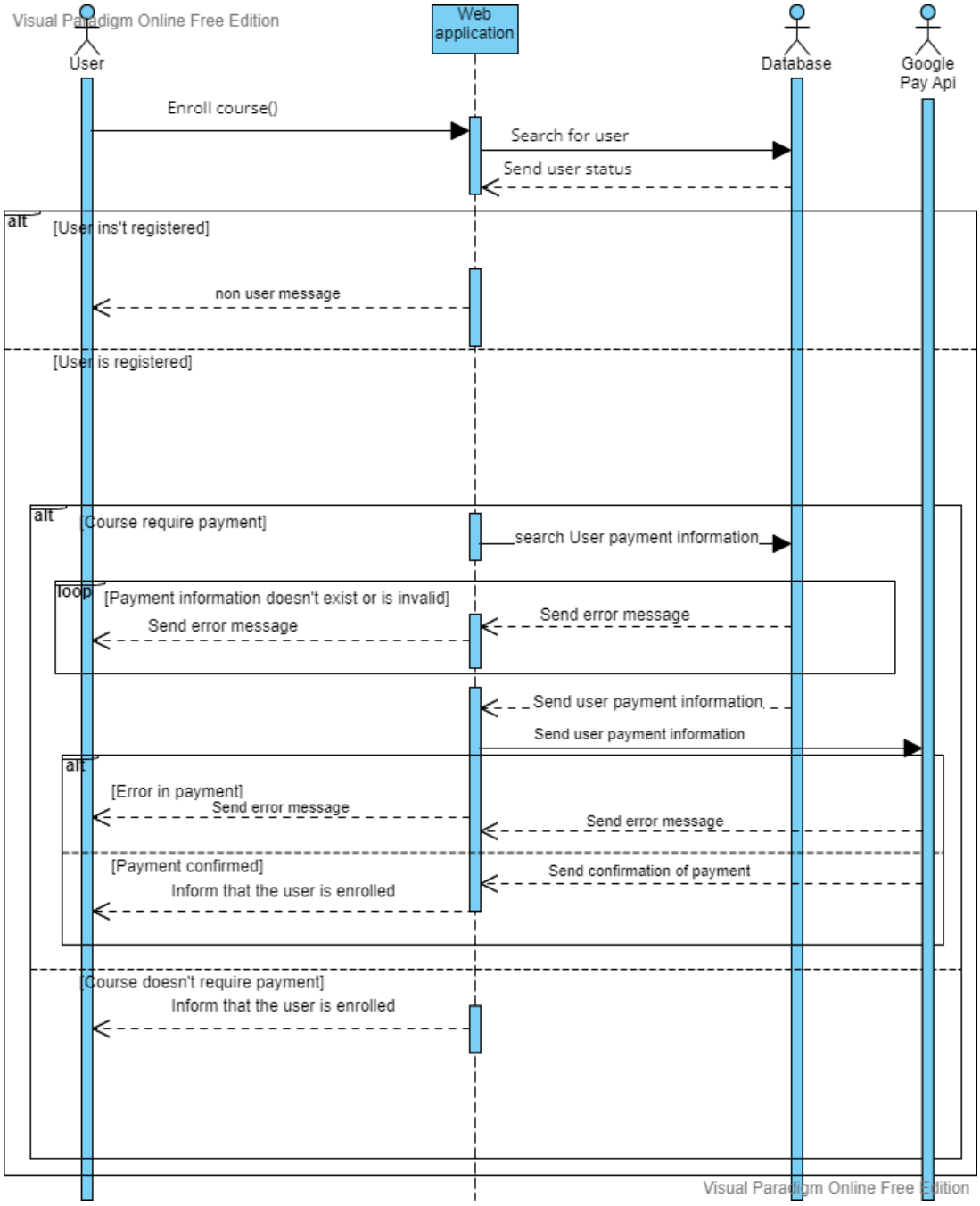

Figure 3.4: Enroll course Sequence Diagram

The figure 3.5 represents the sequence diagram for setting the Live Chat Properties. The user starts by selecting the course, followed by Definitions, and finally Live Chat. The user can then set and change all of the Live Chat properties which

are saved temporarily in the web application. At the end the user can either save the changes made, updating the database with the new settings for the Live Chat or discard them, returning back to how they were at the start. One way or another it's shown to the user a descriptive message about their action and the page course is loaded on the screen.

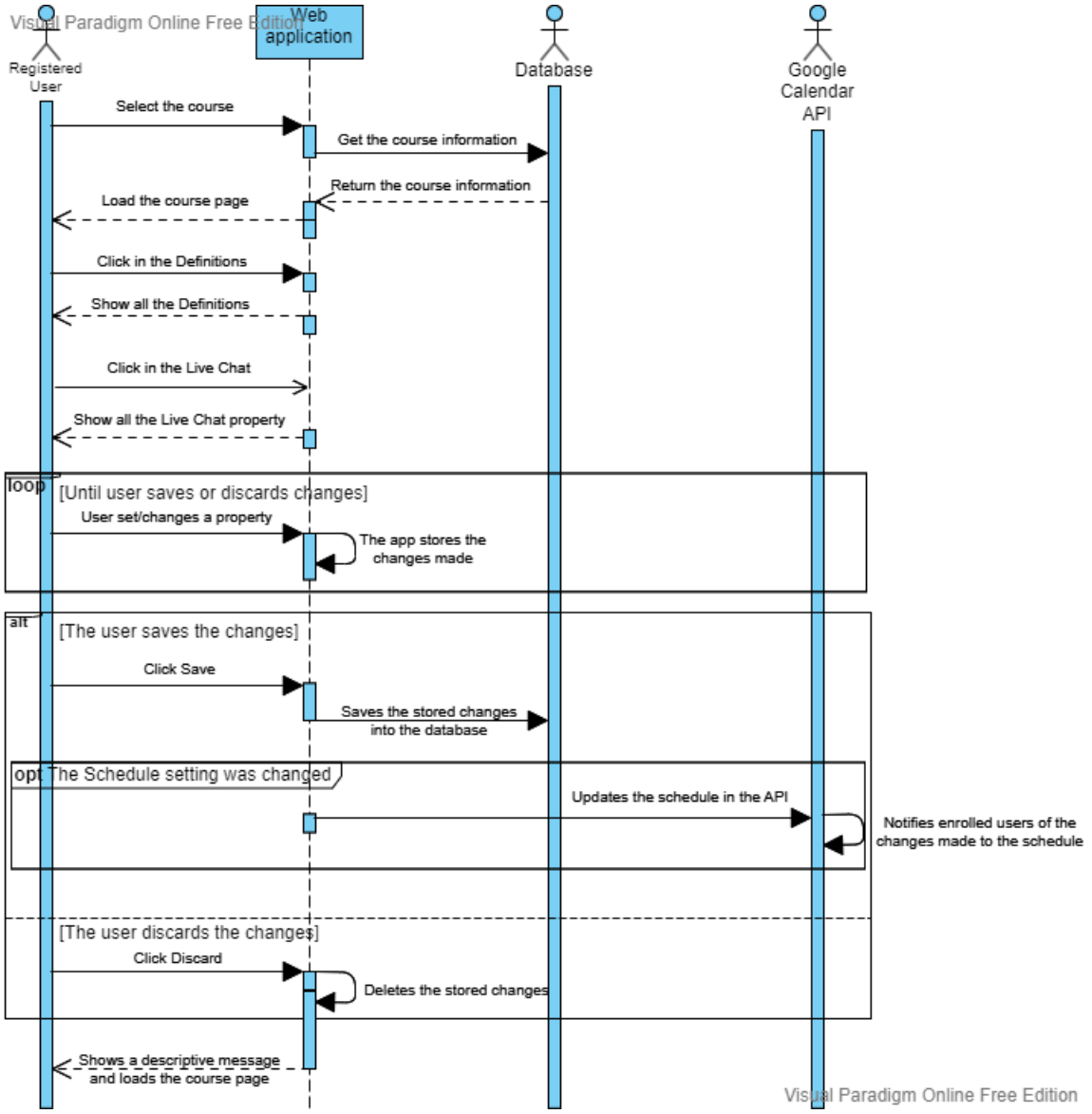

Figure 3.5: Change live chat properties sequence diagram

The diagram in figure 3.6 represents the use case create course. If a registered user wants to create a course, first he choose the option to create a course and through a form the user fill it with the information about the course (name, category,etc). If the course's name chosen already exists, when he try to confirm the course creation the system will show the problem to him and he will have to fill

the form again.

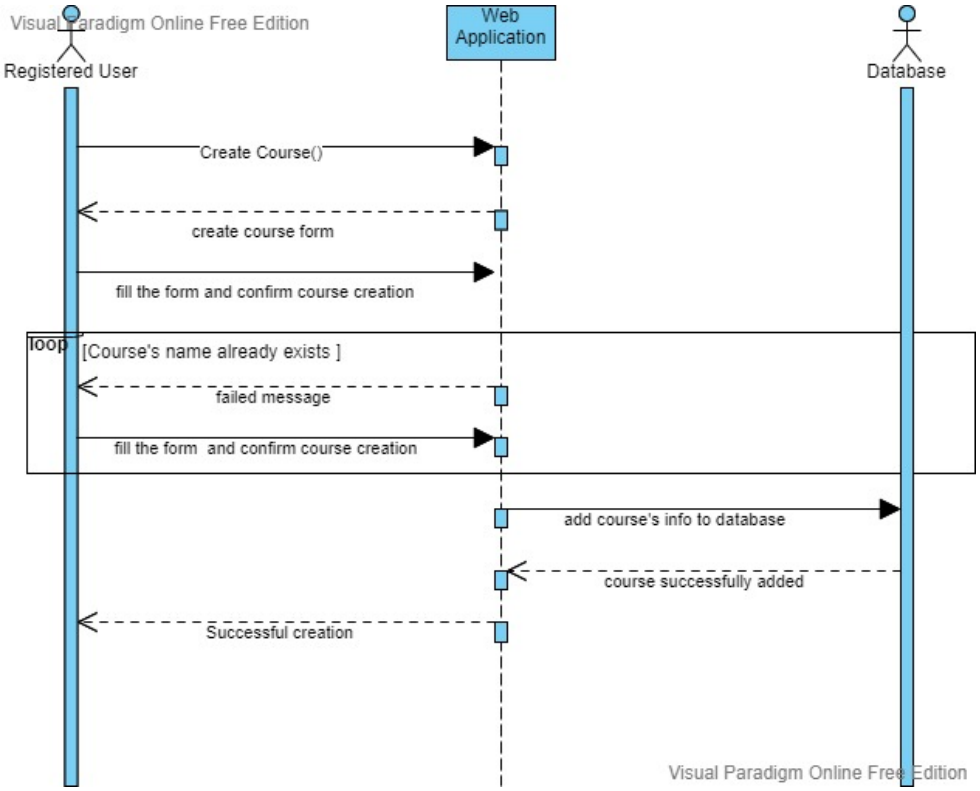

Figure 3.6: Create Course Sequence Diagram

3.2 Other requirements

The website will be accessible from most browsers and their mobile variants. It will be responsive and comfortable to use on mobile devices in various sizes. Sensitive information such as passwords will be encrypted. There will be token-based authentication.

4. System architecture and design

Our web application has a separation between the implementation of frontend and backend. Even though they are separated, they are both built around the Python Django Framework (as of the 1st revision). The backend uses Python while the frontend revolves around HTML, CSS and Javascript. They communicate and are connected via a REST API.

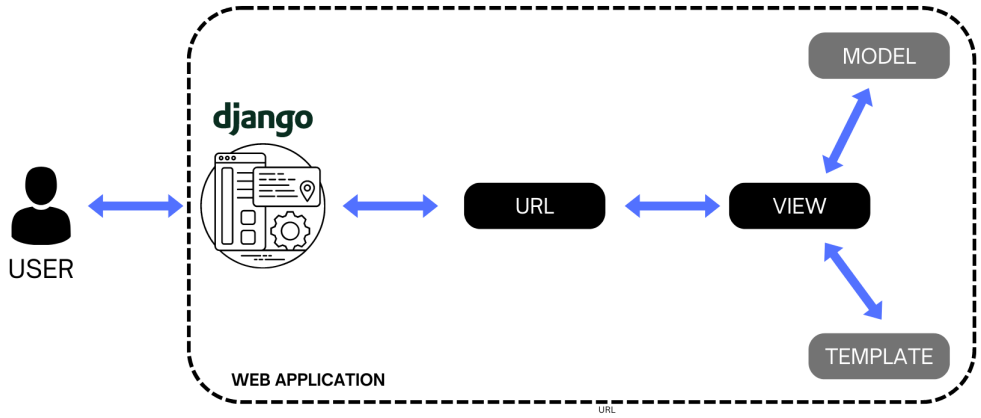

Figure 4.1: Django's Model View Template diagram

In our application, a Model-View-Template architecture is used, being generally represented on the diagram above. This architecture allows it to be separated in 3 parts, each with it's functionality. Model, which represents the logical data structure behind the whole application(as of 1st revision: a SQLite Database); View, being what the user sees when the browser renders a web page, in this case the HTML/CSS/Javascript files and Template, which consists of the static parts of HTML output and how it will be displayed. This architecture allows content to be separated according to it's use in the whole system, alllowing for a more efficient and dynamic development of code.

4.1 Database

A SQLite Database was used for local testing and development, but the deployment to the server was made using a PostgreSQL Database.

4.1.1 Table description

First column is the name of the entry, followed by the type and a brief description. For every table primary key is highlighted green and foreign keys blue.

User (from Django)		
id	INT	User's ID
username	VARCHAR	User's username
password	VARCHAR	User's password

Admin		
id	INT	Admin's ID
userId	INT	User's ID

Profile		
id	INT	Profile's ID
userId	INT	User's ID associated with the profile

Public		
id	INT	Public profile's ID
profileId	INT	Profile's ID associated with the public profile

Continued on next page

Continued from previous page

Public		
name	VARCHAR	Public name
surname	VARCHAR	Public surname
avatar	INT	Public avatar
createdDate	DATE	Public creation date

Private		
id	INT	Private profile's ID
profileId	INT	Profile's ID associated with the private profile
email	VARCHAR	Private profile's email

PaymentDetails		
id	INT	Payment Details' ID
privateId	INT	Private profile's ID
cardNumber	VARCHAR	The card's number
expirationMonth	INT	The card's expiration month
expirationYear	INT	The card's expiration year
cvv	INT	The card's CVV

Category		
id	INT	Category's ID
category	VARCHAR	Category label

CategoriesLiked		
id	INT	Categories Liked's ID
privateId	INT	Private profile's ID
categoryId	INT	Category's ID

Course		
id	INT	Course's ID
categoryId	INT	Category's ID
name	VARCHAR	Course's name
averageMasterTime	INT	Course's average time to master in minutes
price	DECIMAL	Course's price
description	VARCHAR	Course's description
link	VARCHAR	Link

CoursesMade		
id	INT	Courses Made's ID
publicId	INT	Public profile's ID
courseId	INT	Course's ID

CoursesEnrolled		
id	INT	Courses Enrolled's ID
privateId	INT	Private profile's ID
courseId	INT	Course's ID
paymentMethod	INT	Payment Method's ID

Rating		
id	INT	Rating's ID
userId	INT	User's ID
courseId	INT	Course's ID
comment	VARCHAR	Rating's comment
rating	INT	Rating's rating (0-5)

TeachingUnit		
id	INT	Teaching Unit's ID
courseId	INT	Course's ID
name	VARCHAR	Teaching Unit's name

Material		
id	INT	Material's ID
unitId	INT	Teaching Unit's ID
materialName	VARCHAR	Material's name
content	VARCHAR	Material's content
type	VARCHAR	Material's type

Written		
id	INT	Written's ID
materialId	INT	Material's ID
title	VARCHAR	Written's title

Photo		
id	INT	Photo's ID
materialId	INT	Material's ID
label	VARCHAR	Photo's label

Audio		
id	INT	Audio's ID
materialId	INT	Material's ID
time	INT	Audio's time in seconds

Video		
id	INT	Video's ID
materialId	INT	Material's ID
time	INT	Video's time in seconds

LiveChat		
id	INT	Live Chat's ID
courseId	INT	Course's ID
maxParticipants	INT	Live Chat's max number of participants
chat_enable	BOOL	Live Chat's enable

Schedule		
id	INT	Schedule's ID
liveChatId	INT	Live Chat's ID
date	DATE	Schedule's date

Continued on next page

Continued from previous page

Schedule		
time	TIME	Schedule's time

Participants		
id	INT	Participants' ID
userId	INT	User's ID
liveChatId	INT	Live Chat's ID

4.1.2 Database diagram

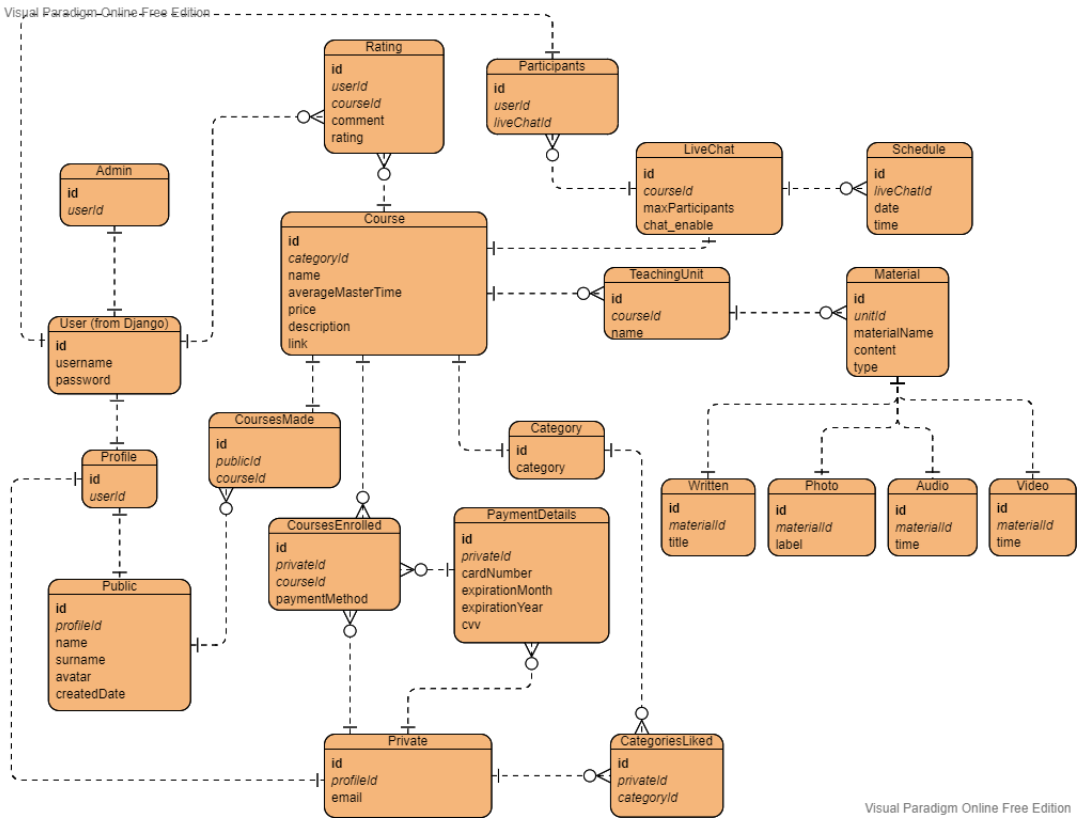

Figure 4.2: ER Database diagram

4.2 Class diagram

The diagram below shows main relationships between objects in our system.

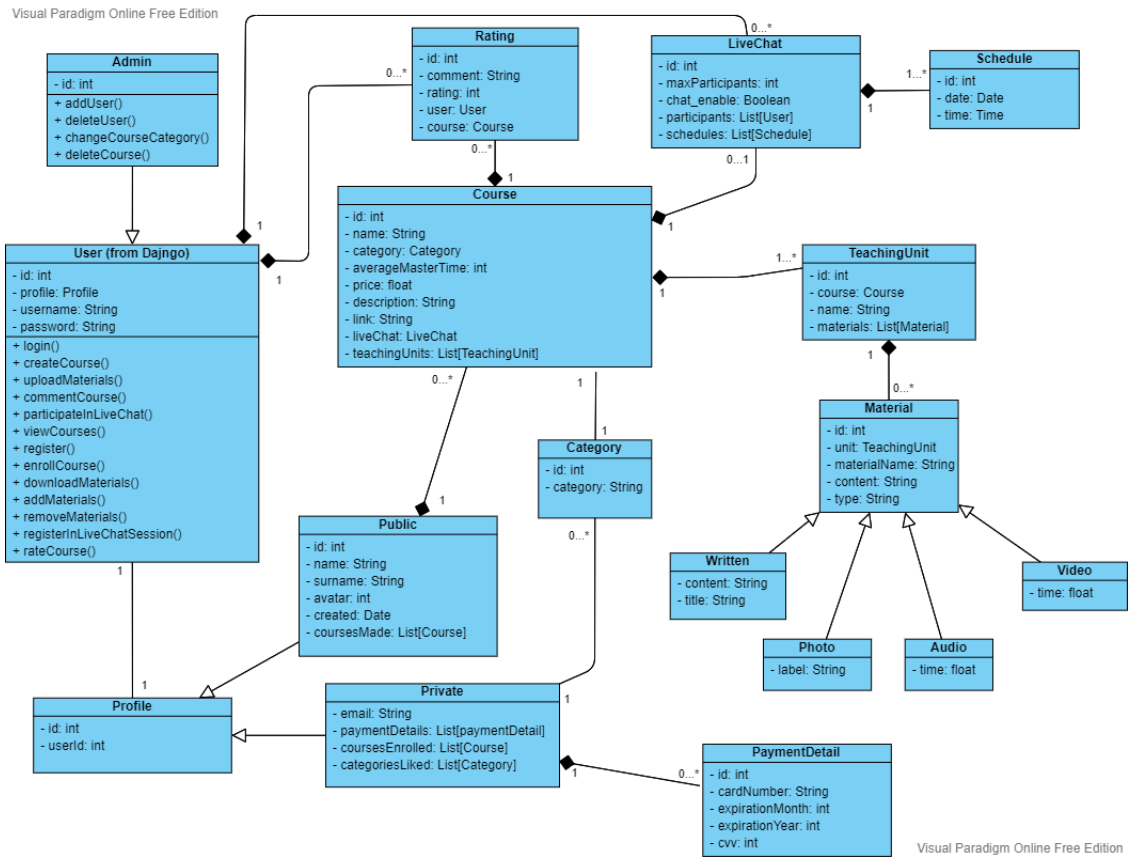

Figure 4.3: General functionalities class diagram (1st revision)

4.3 State machine diagram

When the user that owns the course access a teaching unit pages it is given the option to add new materials. For that, the user needs to choose between the four types of materials (audio, video, written and image) that need to fill out a form with some information about the materials. The system will wait until the form is posted and then store it in the database.

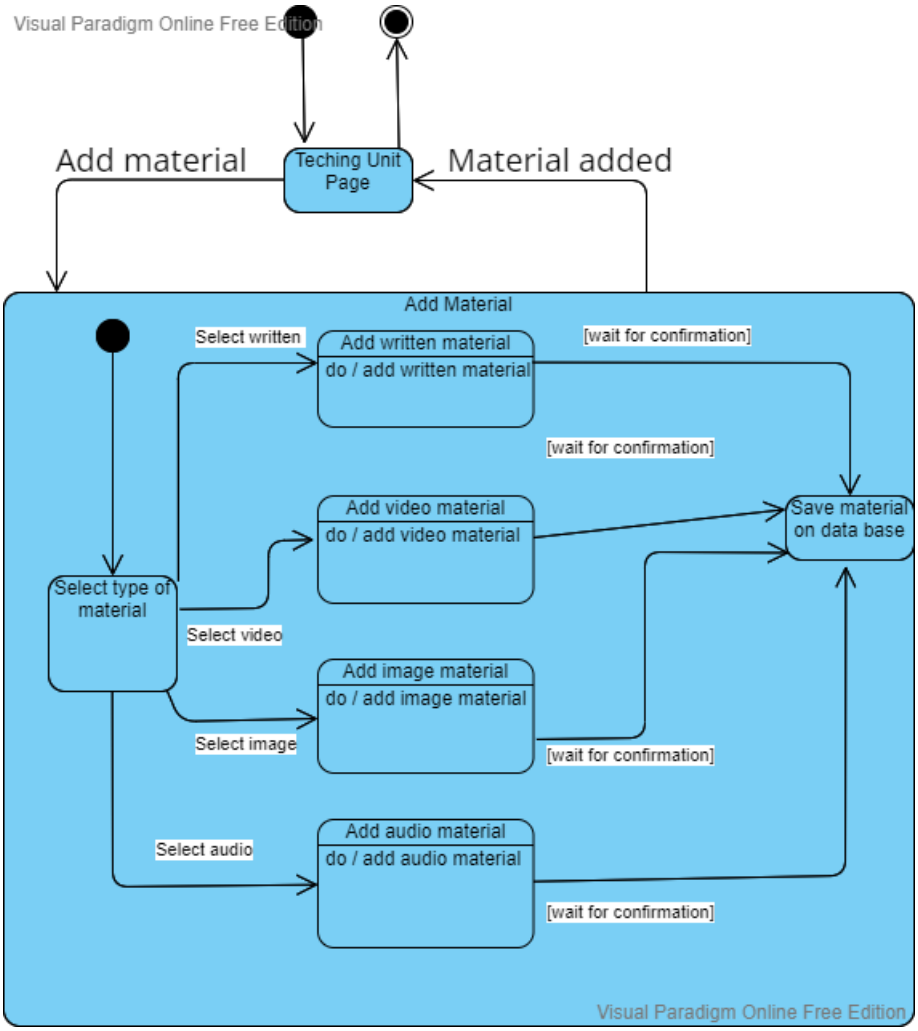

Figure 4.4: State machine diagram for adding materials

4.4 Activity diagram

When the user enters in the ”Courses created” page, the application frontend will request the backend to retrieve all the information about the courses belonging to the current user in order to display it, along with a button to create a new course. The user must then enter the course details and submit them. The application frontend will again request the backend to store the respective course, giving the information entered by the user. According to the success or not of the creation of the course,an error message will be shown in case of failure, or the new course's page will be displayed.

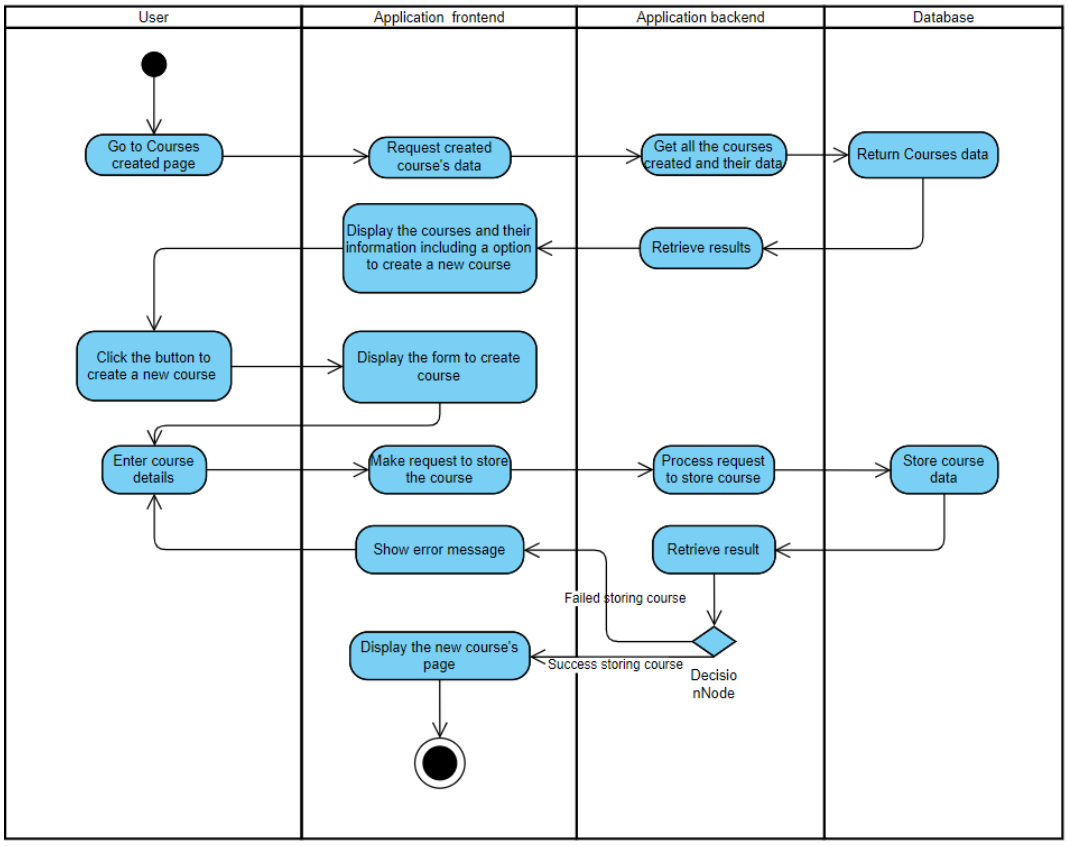

Figure 4.5: Activity Diagram for creating a new course

4.5 Component diagram

A component diagram is often used to describe the relationships between the physical components of a system in the way of a static model. As seen in the diagram, the application is split in two parts: Frontend and Backend. They communicate via HTTP Requests. Users only see the ”finished product” which is presentend to them by the Frontend. All accesses to the models and database happen in the Backend, which consists of a Django app and is connected via TCP/IP to a PostgreSQL database.

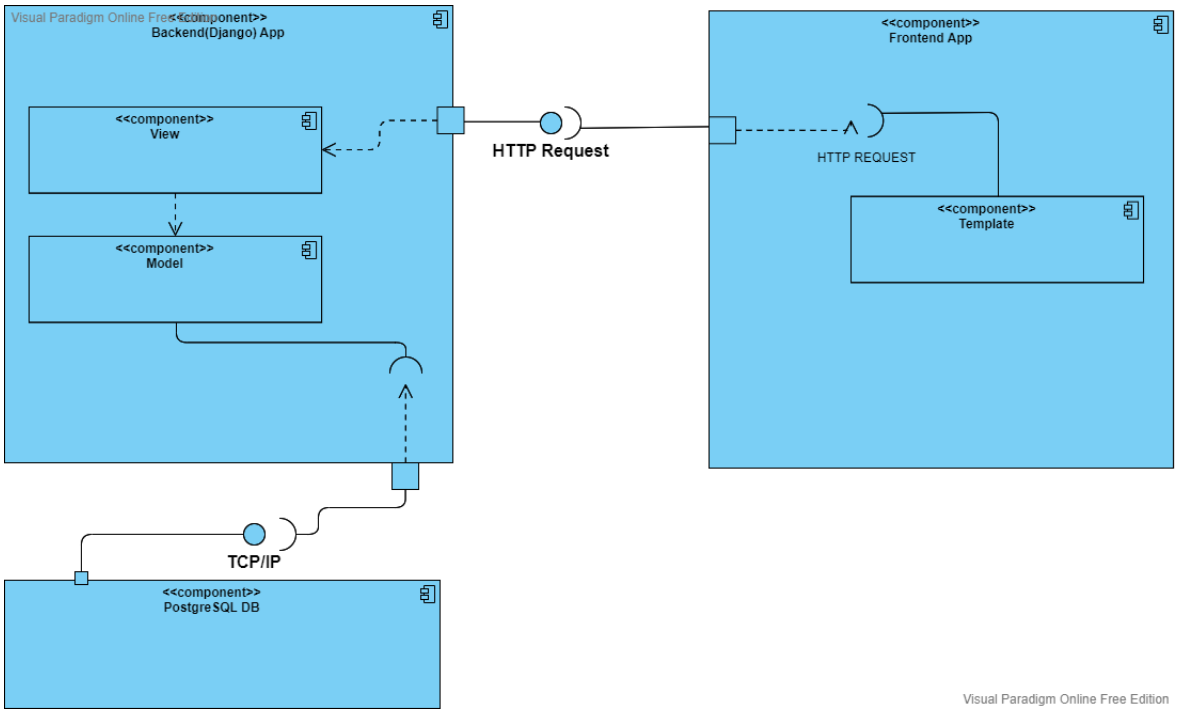

Figure 4.6: Component Diagram

5. Implementation and user interface

5.1 Tools and technologies

- The documentation of the project was written in *LaTex(1)*, with help of the collaborative writing platform *Overleaf(2)*.
- UML Diagrams were designed using *Visual Paradigm Online(3)*.
- Implementation of the backend relies on the *Python(4)* framework *Django(5)*.
- Frontend consists of code written in *HTML(6)*, *CSS(7)* and *Javascript(8)* languages.
- *Docker(9)* was used as a containerization tool, to facilitate requirement and dependency gathering for development.
- Frontend and Backend of the application communicate via a *REST API(10)*.
- When developing locally, a *SQLite(11)* database was used. On deployment, *PostgreSQL(12)* was used as server host database.
- For testing purposes, *Selenium IDE(13)* was used for system testing, and *unittest(14)*, a *Python* framework with *Django* affinity was used for unit testing.
- *Render(15)* was the cloud-based platform which we used to deploy our web application.
- All files regarding the project were continuously updated and stored using *GitLab(16)*.

1.https://www.latex-project.org
2.https://www.overleaf.com/
3.https://online.visual-paradigm.com
4.https://www.python.org
5.https://www.djangoproject.com
6.https://www.w3schools.com/html/
7.https://www.w3schools.com/css/
8.https://www.javascript.com
9.https://www.docker.com
10.https://restfulapi.net
11.https://www.sqlite.org/index.html
12.https://www.postgresql.org
13.https://www.selenium.dev/selenium-ide/
14.https://docs.djangoproject.com/en/4.0/topics/testing/overview/
15.https://render.com
16.https://about.gitlab.com

5.2 Software testing

Software Testing is used to exam the behavior of the software by validation and verification to ensure that there are no errors and that it works in accordance to the specific requirements of the project.

5.2.1 Component testing

A component is an individual element of an application that along others forms the whole application. Component testing is a method of validating the single component without testing the entire system. We made a total of 6 tests with some positive and negative outcomes. We tested register, login, search results, profile, enrolled course and manage payment details.

Test 1 - Register

The test starts by getting the Client() object, the register URL and creating some user data.

```
class TestRegister(TestCase):

    def setUp(self):
        # Get the client object and the url
        self.client = Client()
        self.register_url = reverse('register')
        # Create test data
        self.userData = {'first_name': 'Test',
                         'last_name': 'User',
                         'username': 'testUser',
                         'email': 'testUser@test.com',
                         'password1': 'testUser',
                         'password2': 'testUser'}
```

1.1: Registers the User by sending a POST request to the register page with the user data created earlier. Checks that the correct records were created.

```
    def test_register_new_user(self):
        # Get the response to a POST request on the register page
        response = self.client.post(self.register_url, self.userData)
        # Because it is a redirect it has code 302
        self.assertEqual(response.status_code, 302)
```

```
        # Check if a user was created as well as a profile, public and private
        user = models.User.objects.get(id=1)
        profile = models.Profile.objects.get(userId=user.id)
        public = models.Public.objects.get(profileId=profile.id)
        private = models.Private.objects.get(profileId=profile.id)
        # Assert results
        self.assertEqual(user.username,'testUser')
        self.assertEqual(public.name,'Test')
        self.assertEqual(private.email, 'testUser@test.com')
```

Results: The single test was passed in 0.313s.

```
python .\manage.py test app.tests.test_views.TestRegister
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.313s

OK
Destroying test database for alias 'default'...
```

Test 2 - Login

The test starts by getting the Client() object and the login URL. Then it creates some good and bad user data and creates a user with the good data.

```
class TestLogin(TestCase):

    def setUp(self):
        # Get the client object and the url
        self.client = Client()
        self.login_url = reverse('loginpage')
        # Create test data
        self.userData = {'username': 'testUser',
                         'password1': 'testUser'}
        self.badUserData = {'username': 'wrongCredentials',
                            'password1': 'wrongCredentials'}
        # Create a user
        self.user =
        ↪  models.User.objects.create_user(username=self.userData['username'],
        ↪  password=self.userData['password1'])
```

2.1: Checks that the user was created successfully.

```
    def test_if_user_is_created(self):
        # Get the user with id=1 (in tests the db is empty so it will be the first
        ↪  one)
        user = models.User.objects.get(id=1)
        # Check if it has the same username
        self.assertEqual(user.username, 'testUser')
```

2.2: Logs in the user by sending a POST request to the login page with the good data. Checks that the landing page was rendered as expected.

```
    def test_successful_login(self):
        # Get the response to a POST request on the login page
        response = self.client.post(self.login_url, self.userData)
        # Because it is a render to page landing.html it has code 200
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'app/landing.html')
```

2.3: Tries to login the user by sending a POST request to the login page with the bad data. Checks that it was redirected to another page.

```
    def test_unsuccessful_login(self):
        # Get the response to a POST request on the login page
```

```
        response = self.client.post(self.login_url, self.badUserData)
        # Because it is a redirect it has code 302
        self.assertEqual(response.status_code, 302)
```

Results: The 3 tests were passed in 1.412s.

```
python .\manage.py test app.tests.test_views.TestLogin
Found 3 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
...
----------------------------------------------------------------------
Ran 3 tests in 1.412s

OK
Destroying test database for alias 'default'...
```

Test 3 - Search Results

The test starts by getting the Client() object and the search results URL. Then it creates a category and 5 courses.

```
class TestSearchResults(TestCase):

    def setUp(self):
        # Get the client object and the url
        self.client = Client()
        self.results_url = reverse('searchResults')
        # Create test data
        # Create a category
        self.category = models.Category.objects.create(category='General')
        # Create courses
        num = 5
        for i in range(num):
            models.Course.objects.create(categoryId=self.category, name="Test
            ↪  Course "+str(i), averageMasterTime=i+5, price=i, description="Test
            ↪  Description"+str(i))
```

3.1: Sends a GET request to the search results page with the name "test". Checks that it got 5 results in courses, the 5 created earlier.

```
def test_successful_results(self):
        # Get the response to a GET request on the results page
        response = self.client.get(self.results_url, {'name': 'test'})
        # Because it is a render to page searchResults.html it has code 200
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'app/searchResults.html')
        # Check that it had 5 result in courses
        self.assertEqual(len(response.context['courses']), 5)
```

3.2: Sends a POST request to the search results page with the name "test". Checks that it was redirected to another page.

```
    def test_unsuccessful_results(self):
        # Get the response to a POST request on the results page
        response = self.client.post(self.results_url, {'name': 'test'})
        # Because it is a redirect it has code 302
        self.assertEqual(response.status_code, 302)
```

Results: The 2 tests were passed in 0.037s.

```
python .\manage.py test app.tests.test_views.TestSearchResults
Found 2 test(s).
```

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 0.037s

OK
Destroying test database for alias 'default'...
```

Test 4 - Profile

The test starts by getting the Client() object. Then it creates a user as well as it's profile, public and private.

```
class TestProfile(TestCase):

    def setUp(self):
        # Get the client object and the url
        self.client = Client()
        # Create test data
        # Create a user
        self.user = models.User.objects.create_user(username='testUser',
        ↪  password='testUser')
        # Create it's profile, public and private
        self.profile = models.Profile.objects.create(userId=self.user)
        self.public = models.Public.objects.create(profileId=self.profile,
        ↪  name='Test', surname='User', avatar=1)
        self.private = models.Private.objects.create(profileId=self.profile,
        ↪  email='testUser@test.com')
```

4.1: Sends a GET request to the view profile page with the id of the user created earlier. Checks that the profile page of the correct user was rendered.

```
    def test_user_profile_page(self):
        # Get the profile page for the id of the user
        profile_url = reverse('viewProfile', kwargs={'id': self.user.id})
        response = self.client.get(profile_url)
        # Because it is a render to page viewProfile.html it has code 200
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'app/viewProfile.html')
        # Check if it has the info of the user created
        self.assertEqual(response.context['public'].name, self.public.name)
        self.assertEqual(response.context['public'].surname, self.public.surname)
        self.assertEqual(response.context['public'].avatar, self.public.avatar)
```

4.2: Sends a GET request to the view profile page with a random id. Checks that it was redirected to another page.

```
    def test_error_user_profiel_page(self):
        # Get the profile page for a random id
        profile_url = reverse('viewProfile', kwargs={'id': 5})
        response = self.client.get(profile_url)
        # Because it is a redirect it has code 302
        self.assertEqual(response.status_code, 302)
```

Results: The 2 tests were passed in 0.483s.

```
python .\manage.py test app.tests.test_views.TestProfile
Found 2 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
..
----------------------------------------------------------------------
Ran 2 tests in 0.483s

OK
Destroying test database for alias 'default'...
```

Test 5 - Enrolled Course

The test starts by getting the Client() object and the payments URL. Then it creates user and creator data and creates the 2 users. It also creates the user's profile and private as well as a category.

```
class TestEnrolledCourse(TestCase):

    def setUp(self):
        # Get the client object and the url
        self.client = Client()
        self.payments_url = reverse('payments')
        # Create user test data
        self.creatorData = {'username': 'courseCreator',
                            'password': 'courseCreator'}
        self.userData = {'username': 'courseEnroller',
                         'email': 'courseEnroller@test.com',
                         'password': 'courseEnroller'}
        # Create the users
        self.creator =
        ↪  models.User.objects.create_user(username=self.creatorData['username'],
        ↪  password=self.creatorData['password'])
        self.user =
        ↪  models.User.objects.create_user(username=self.userData['username'],
        ↪  password=self.userData['password'])
        # Create the users profile and private
        self.userProfile = models.Profile.objects.create(userId=self.user)
        self.userPrivate =
        ↪  models.Private.objects.create(profileId=self.userProfile,
        ↪  email=self.userData['email'])
        # Create a category
        self.category = models.Category.objects.create(category='General')
```

5.1: Starts by creating a free course and creating a CoursesEnrolled record linked to both the course and the user created earlier. Then it logs in the user with it's data and makes a GET request to the payments URL. Finally it checks that there's a enrolled course without a payment method.

```
    def test_enroll_free_course(self):
        # Create a free course
        course = models.Course.objects.create(categoryId=self.category, name="Test
        ↪  Course", averageMasterTime=5, price=0, description="Test Description")
        # Make it so that the user is enrolled in the course
```

```
        models.CoursesEnrolled.objects.create(privateId=self.userPrivate,
        ↪  courseId=course)
        # Login the user
        self.client.login(username=self.userData['username'],
        ↪  password=self.userData['password'])
        # Go to the payments page and check if there was an enrolled course
        ↪  without a payment method
        response = self.client.get(self.payments_url)
        self.assertEqual(response.context['enrolledCourses'][0].paymentMethod,
        ↪  None)
```

5.2: Starts by creating a paid course, a payment detail linked to the user's private created earlier and creating a CoursesEnrolled record linked to the course, the payment detail and the user created earlier. Then it logs in the user with it's data and makes a GET request to the payments URL. Finally it checks that there's a enrolled course with the same payment method created.

```
    def test_enroll_paid_course(self):
        # Create a paid course
        course = models.Course.objects.create(categoryId=self.category, name="Test
        ↪  Course", averageMasterTime=5, price=5, description="Test Description")
        # Create a payment detail
        paymentDetail =
        ↪  models.PaymentDetails.objects.create(privateId=self.userPrivate,
        ↪  cardNumber='1234123412341234', expirationMonth=1, expirationYear=24,
        ↪  cvv=123)
        # Make it so that the user is enrolled in the course
        models.CoursesEnrolled.objects.create(privateId=self.userPrivate,
        ↪  courseId=course, paymentMethod=paymentDetail)
        # Login the user
        self.client.login(username=self.userData['username'],
        ↪  password=self.userData['password'])
        # Go to the payments page and check if there was an enrolled course with a
        ↪  payment method
        response = self.client.get(self.payments_url)
        self.assertEqual(response.context['enrolledCourses'][0].paymentMethod,
        ↪  paymentDetail)
```

Results: The 2 tests were passed in 1.601s.

```
python .\manage.py test app.tests.test_views.TestEnrollCourse
Found 2 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
```

```
..
----------------------------------------------------------------------
Ran 2 tests in 1.601s

OK
Destroying test database for alias 'default'...
```

Test 6 - Manage Payment Details

The test starts by getting the Client() object and the manage payments URL. Then it creates user and card data. It also creates the user, it's profile and it's private

```
class TestManagePaymentDetails(TestCase):

    def setUp(self):
        # Get the client object and the url
        self.client = Client()
        self.manage_payments_url = reverse('managePaymentDetails')
        # Create user test data
        self.userData = {'username': 'testUser',
                         'email': 'testUser@test.com',
                         'password': 'testUser'}
        self.cardData = {'cardNumber': '1234123412341234',
                         'expirationMonth': 12,
                         'expirationYear': 24,
                         'cvv': 123 }
        # Create the user
        self.user =
        ↪  models.User.objects.create_user(username=self.userData['username'],
        ↪  password=self.userData['password'])
        # Create the users profile and private
        self.userProfile = models.Profile.objects.create(userId=self.user)
        self.userPrivate =
        ↪  models.Private.objects.create(profileId=self.userProfile,
        ↪  email=self.userData['email'])
```

6.1: Starts by creating a payment detail with the card data. Then it logs in the user and makes a GET request to the manage payments URL and checks that the page was rendered and the payment detail created is displayed.

```
    def test_add_card(self):
        # Create a payment detail
        paymentDetail =
        ↪  models.PaymentDetails.objects.create(privateId=self.userPrivate,
        ↪  cardNumber=self.cardData['cardNumber'],
        ↪  expirationMonth=self.cardData['expirationMonth'],
        ↪  expirationYear=self.cardData['expirationYear'],
        ↪  cvv=self.cardData['cvv'],)
        # Login the user
        self.client.login(username=self.userData['username'],
        ↪  password=self.userData['password'])
```

```
        # Do a GET request in the managePaymentsDetails
        response = self.client.get(self.manage_payments_url)
        # Because it is a render to page managePaymentDetails.html it has code
        ↪  200
        self.assertEqual(response.status_code, 200)
        self.assertTemplateUsed(response, 'app/managePaymentDetails.html')
        # Check if the payment detail is displayed
        self.assertEquals(response.context['paymentDetails'][0], paymentDetail)
```

Results: The single test was passed in 0.483s.

```
python .\manage.py test app.tests.test_views.TestManagePaymentDetails
Found 1 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
.
----------------------------------------------------------------------
Ran 1 test in 0.483s

OK
Destroying test database for alias 'default'...
```

5.2.2 System testing

We used the Selenium framework to test the completed system. We tested 5 scenarios some with positive and negative outcomes. We tested register, login, remove users, payment methods and log out.

System Test 1: Register

Test 1.1: Unsuccessful Register

Input:

- **first_name:** 'Test'
- **last_name:** 'User'
- **username:** 'testUser'
- **email:** 'testUser@fer.hr'
- **password1:** 'TESTUSER'
- **password2:** 'testuser'

Expected: *Register unsuccessful*

Actual Result: *Register unsuccessful*

Status: *Passed*

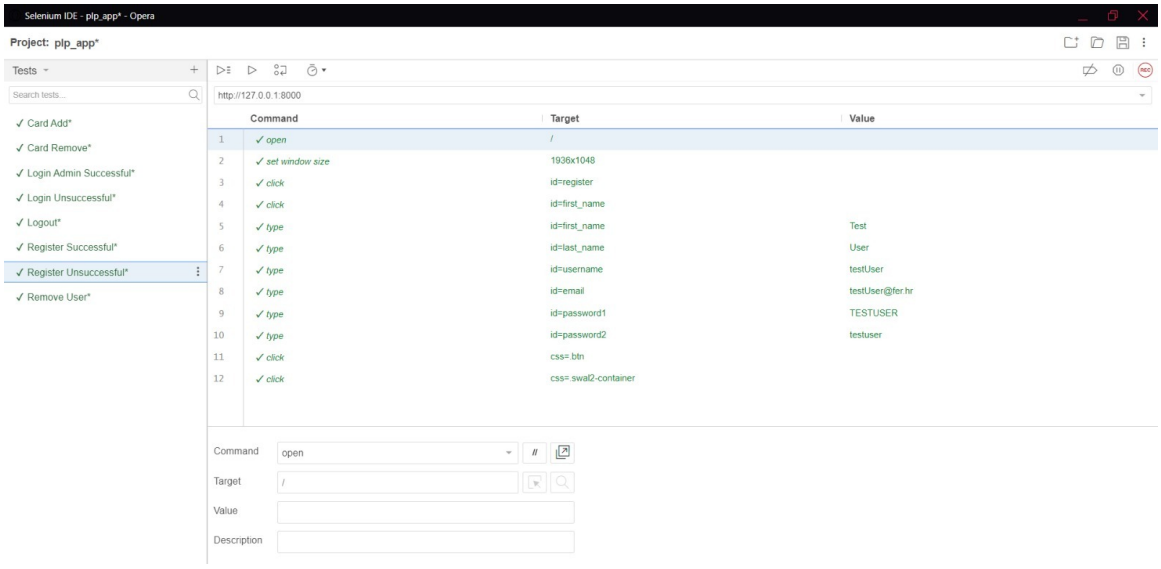

Figure 5.1: Selenium UI displaying the Test 1.1

Test 1.2: Successful Register

Input:

- **first_name:** 'Test'
- **last_name:** 'User'
- **username:** 'testUser'
- **email:** 'testUser@fer.hr'
- **password1:** 'testUser'
- **password2:** 'testUser'

Expected: *Register Successful*
Actual Result: *Register Successful*
Status: *Passed*

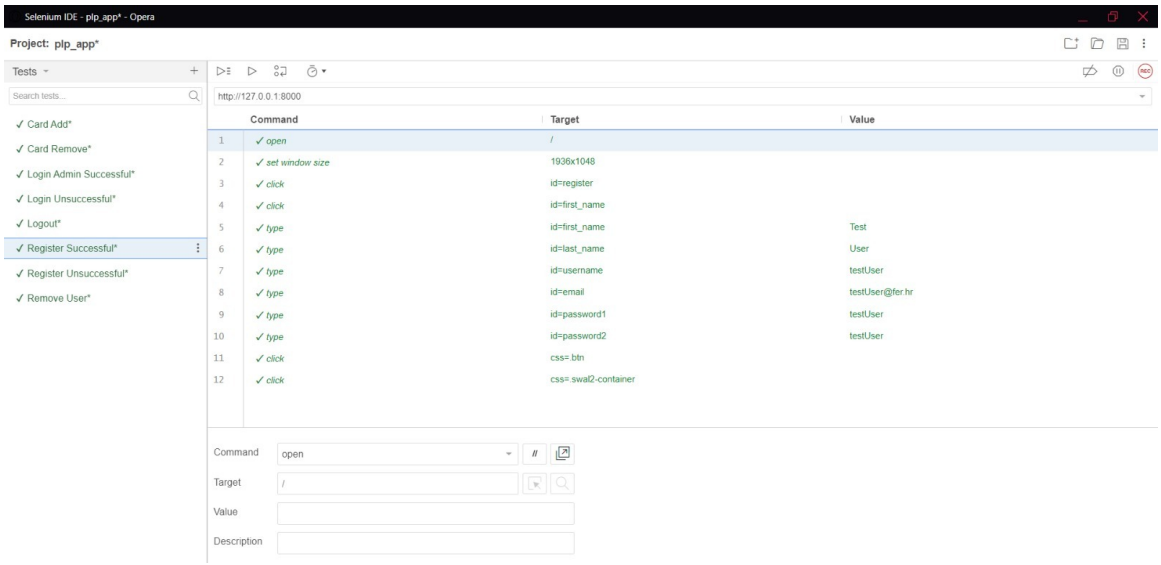

Figure 5.2: Selenium UI displaying the Test 1.2

System Test 2: Login

Test 2.1: Unsuccessful Login

Input:

- **username:** 'wrongUser'
- **password1:** 'wrongUser'

Expected: *Login unsuccessful*
Actual Result: *Login unsuccessful*
Status: *Passed*

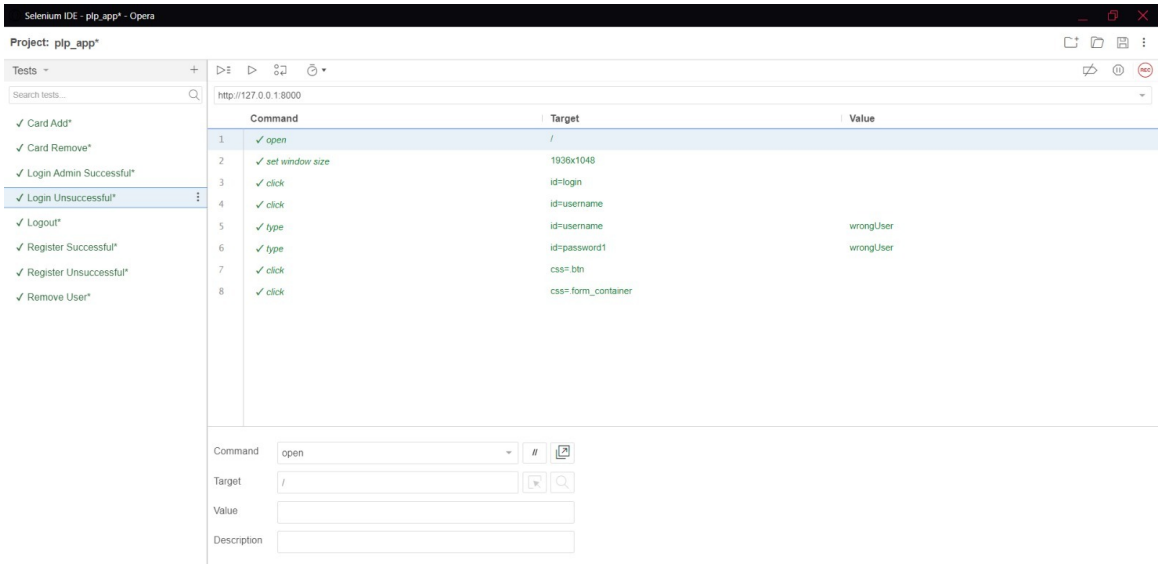

Figure 5.3: Selenium UI displaying the Test 2.1

Test 2.2: Successful Login

Input:

- **username:** 'admin'
- **password1:** 'admin'

Expected: *Login Successful*
Actual Result: *Login Successful*
Status: *Passed*

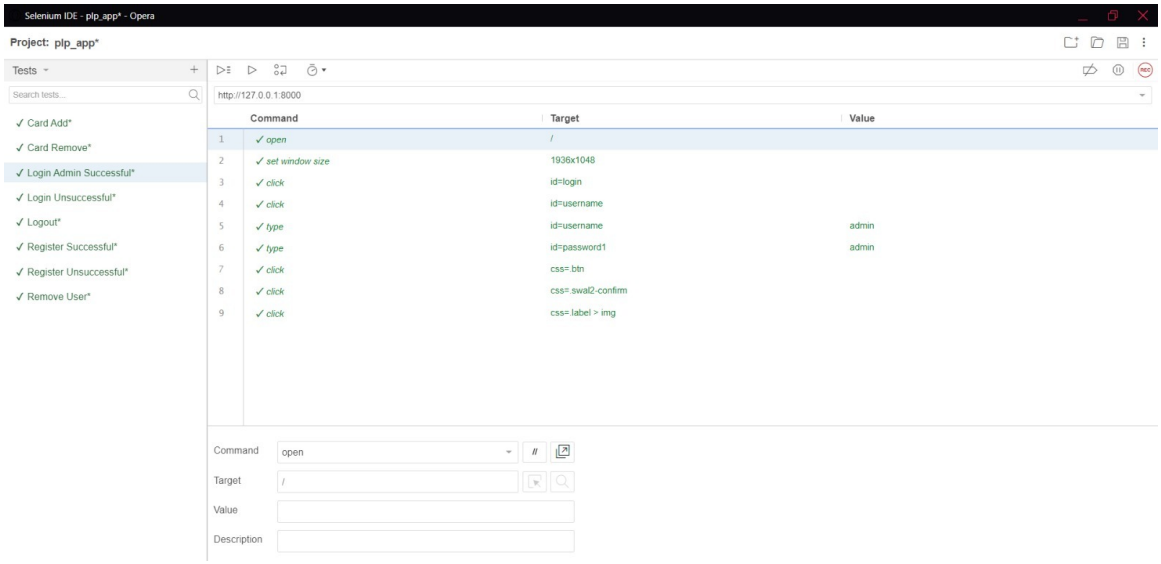

Figure 5.4: Selenium UI displaying the Test 2.2

System Test 3: Remove Users

Test 3.1: Remove User

Input:

- **username:** 'testUser'

Expected: *User Removed Successfully*

Actual Result: *User Removed Successfully*

Status: *Passed*

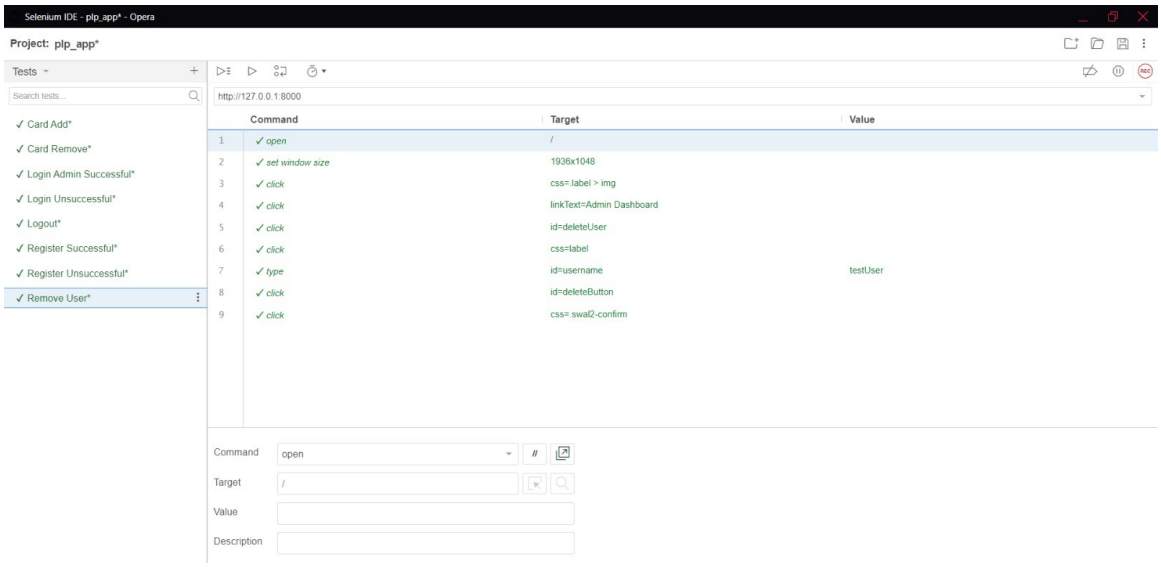

Figure 5.5: Selenium UI displaying the Test 3.1

System Test 4: Payment Methods

Test 4.1: Add Card

Input:

- **cardNumber:** ’4567456745674567’
- **expirationMonth:** ’12’
- **expirationYear:** ’24’
- **cvv:** ’123’

Expected: *Add a Payment Method*
Actual Result: *Added a Payment Method*
Status: *Passed*

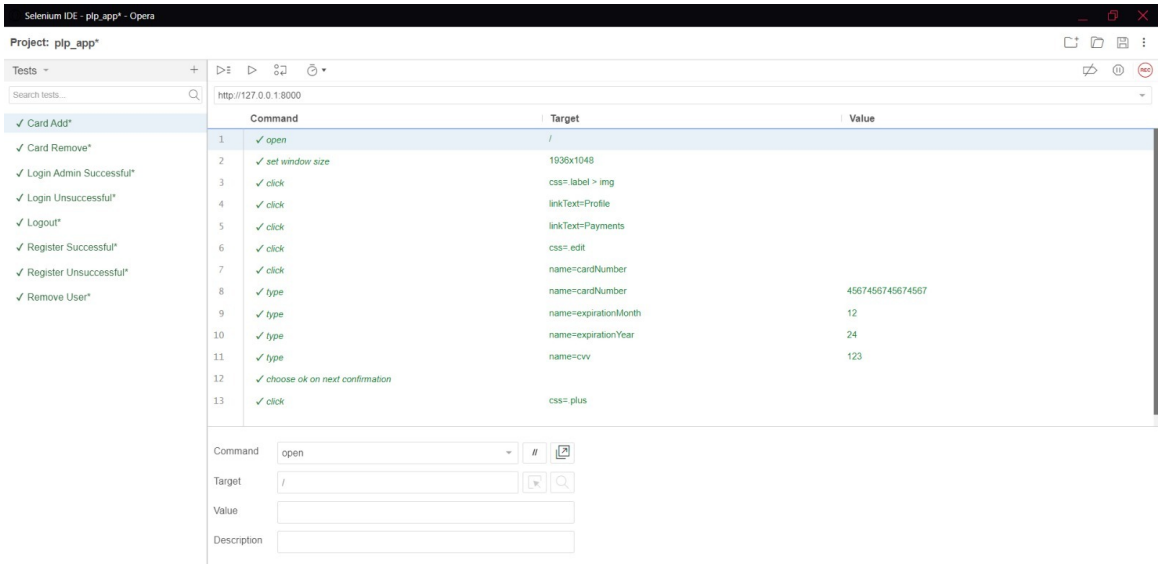

Figure 5.6: Selenium UI displaying the Test 4.1

Test 4.2: Remove Card

Expected: *Remove a Payment Method*
Actual Result: *Removed a Payment Method*
Status: *Passed*

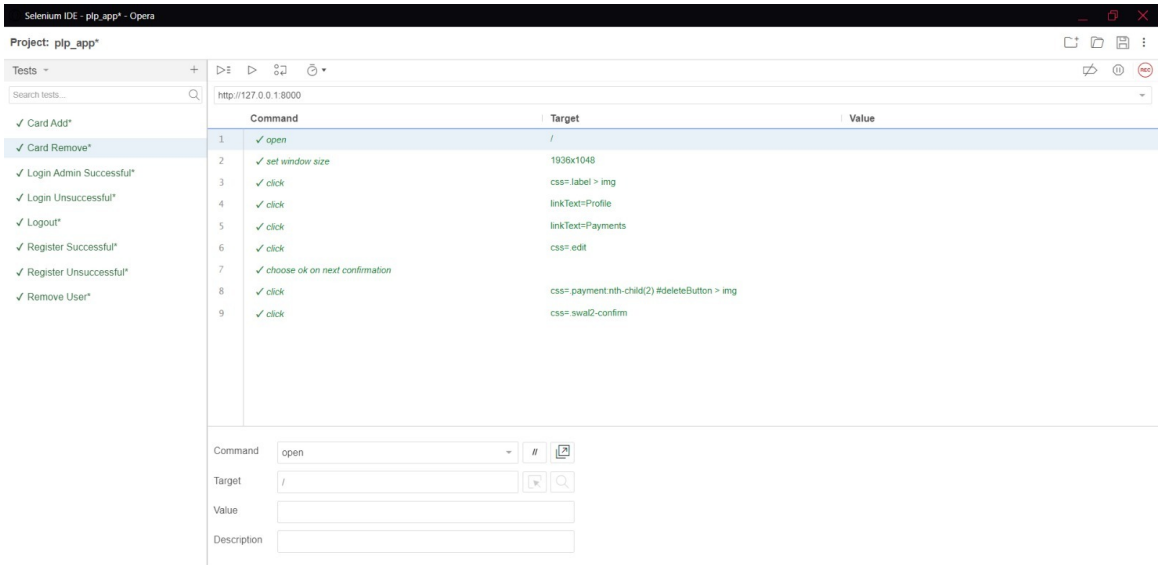

Figure 5.7: Selenium UI displaying the Test 4.2

System Test 5: Log Out

Test 5.1: Log Out

Expected: *Log Out User*

Actual Result: *Logged Out User*

Status: *Passed*

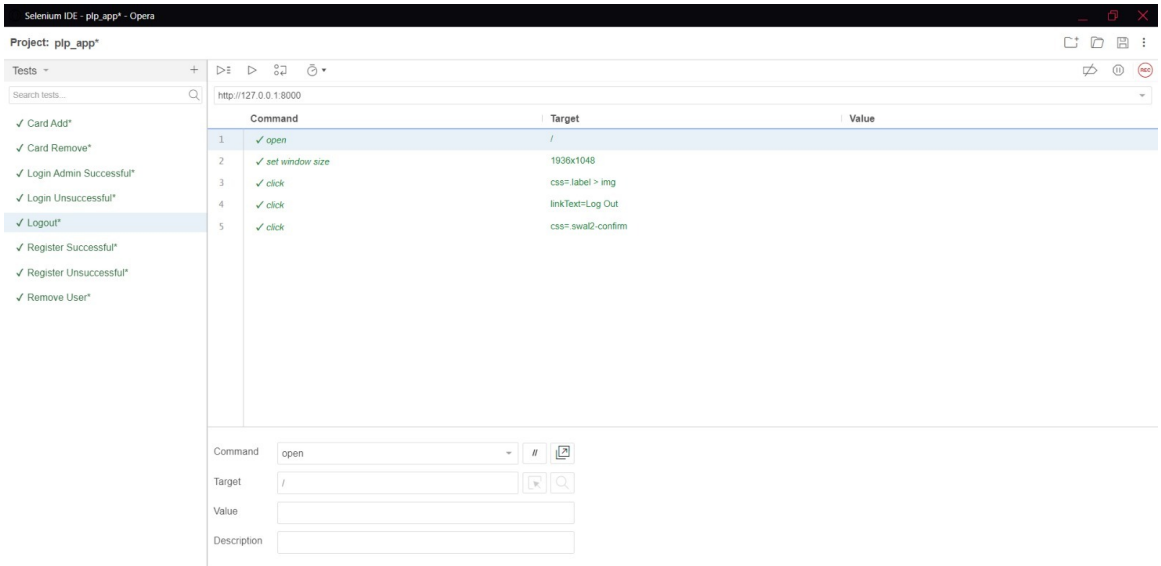

Figure 5.8: Selenium UI displaying the Test 5.1

5.3 Deployment diagram

Deployment diagrams are often used to show system design after the decision of the software and hardware mapping has been made, and of the run-time processing elements and processes that exist within them.
In the following specification layout diagram, frontend and backend communicate via HTTP requests. The Django backend is connected to the PostgreSQL database and can exchange information with it via TCP/IP. Then the web application is deployed on the cloud-based service Render. There are also requests being made through HTTP to the Google Calendar API and the Tawk.to Live Chat Integration. Users can access the web application in their machines, as long as they do so in a browser that supports our system.

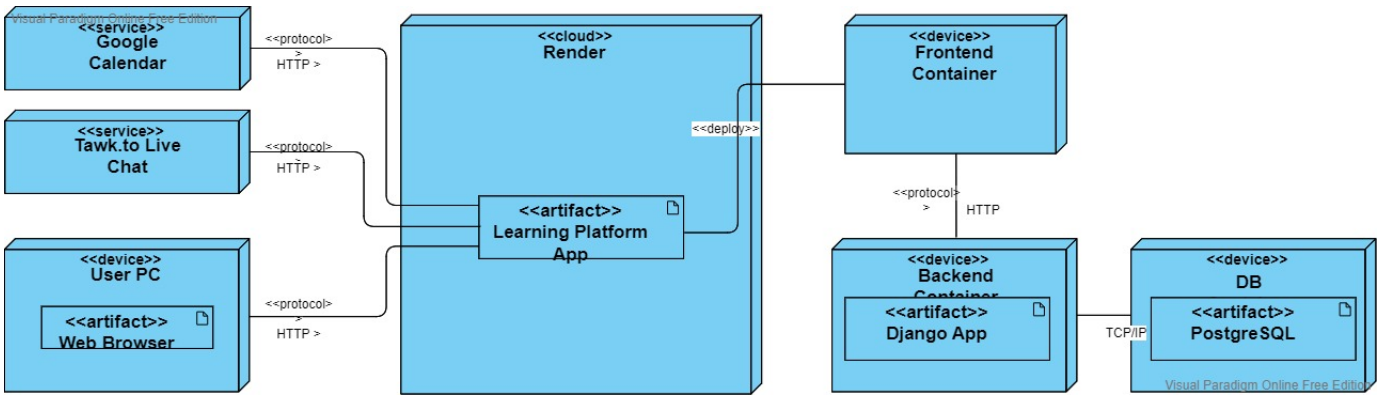

Figure 5.9: Deployment specification layout diagram

5.4 Deployment instructions

5.4.1 Deployed project

Our deployed project which is running on Render can be accessed through the following link:

- `https://plp3-test.onrender.com`
- `https://plp3-test.onrender.com/admin` (Admin Backend)

5.4.2 Local deployment instructions

Deployment of our application locally, mainly for development purposes, can be made using the following instructions:

1. Have docker installed in your machine. `https://www.docker.com`
2. Set DOCKER=1 in the settings.py file
3. Go to shell and cd into the plp_app folder
4. Run: ”docker compose up –build”
5. The application should start on `https://localhost:8080`
6. If you want to boot down the application run: ”docker compose down”

6. Conclusion and future work

Ultimately the main objective of this project was to build a platform where anyone could share their own knowledge in the format of a course, and also gain knowledge in new areas from other users. Our goal was to develop it in the form of a web application, while documenting the process and respecting the rules of software engineering.
This way, it is possible to divide the project into two parts: the documentation part and the coding part, both of them having their own challenges for our team. Having to determine all the user and functional requirements before writing any code was something which we were not very familiar with, but adapted pretty quickly. The same can be said about the UML diagrams and database tables. However, the already existing ideas about the architecture and design of the application really helped us when it came to the point of starting to write the code.
From all the technologies used, *Python* was the only one we all had experience with before, so gaining familiarity with the rest of them was quite challenging in the beginning. Unfortunately, we could not get the live chat integration nor the *Google Calendar* working completely, even though they are integrated in the app and provide some functionality. Taking this into account, the main objective of the project was mostly achieved in the time given, although some improvements can still be made in the future such as: implementing user notifications, better handling of the Google Calendar API and the live chat integration, embedding videos into the teaching unit instead of using a *url* and having using different subclasses for different types of materials.
Nevertheless, the whole process helped us develop ourselves not only in the development and design of software, but also in a social and cooperative way, which is imperative in order to work in collaboration with others. Working as a team and organizing tasks between ourselves, taking into matter each ones best qualities, was something enriching and definitely eye-opening for what's to come if we choose to pursue this area in the future.

References

1. Oblikovanje programske potpore, FER ZEMRIS, http://www.fer.hr/predmet/opp
2. I. Sommerville, "Software engineering", 8th ed, Addison Wesley, 2007.
3. T.C.Lethbridge, R.Langaniere, "Object-Oriented Software Engineering", 2nd ed. McGraw-Hill, 2005.
4. I. Marsic, Software engineering book", Department of Electrical and Computer Engineering, Rutgers University, http://www.ece.rutgers.edu/~marsic/books/SE
5. The Unified Modeling Language, https://www.uml-diagrams.org/
6. Astah Community, http://astah.net/editions/uml-new
7. The Django Framework, https://www.djangoproject.com
8. Getting Started with Django on Render, https://render.com/docs/deploy-django

Index of figures

Appendix: Group activity

Meeting log

1. Meeting
    - Date: October 23, 2022
    - Attendees: José Matos, Francisco Antunes, Lourenço Carvalho, Maria Inês Costa, Rui Guimarães
    - Meeting subjects:
        - General project discussion and organization
        - Functional and non-functional requirements
2. Meeting
    - Date: October 27, 2022
    - Attendees: José Matos, Francisco Antunes, Lourenço Carvalho, Maria Inês Costa, Rui Guimarães
    - Meeting subjects:
        - Technologies and architecture discussion
        - Planning of diagram development
3. Meeting
    - Date: November 17, 2022
    - Attendees: José Matos, Francisco Antunes, Lourenço Carvalho, Maria Inês Costa, Rui Guimarães
    - Meeting subjects:
        - Final diagram discussion
        - Final group revision before first version deadline
        - Defining tasks for the coding part

4. Meeting
    - Date: January 11, 2023
    - Attendees: José Matos, Francisco Antunes, Lourenço Carvalho, Maria Inês Costa, Rui Guimarães
    - Meeting subjects:
        - Final feature distribution
        - Read-through and checking of all features
        - Quick code fixes
        - Distribution of missing documentation parts for final deadline

Activity table

	José Matos	Francisco Antunes	Lourenço Carvalho	Maria Inês Costa	Rui Guimarães		
Project management	3	0	0.25	0	0		
Project task description	0.25	0.25	0.25	0.25	0.25		
Functional requirements	1	1	1	1	1		
Individual use case description	1.5	1	1	1	1		
Use case diagram	1	1	1	1	1		
Sequence diagram	0	0.5	0.5	0.5	1		
Other requirements description	0.25	0.25	0.25	0.25	0.25		
System architecture and design	1.5	0	0	0	0		
Database	0	0	1	0.25	0		
Class diagram	0	0	1	0	0		
State diagram	0	1	0	0	0		
Activity diagram	0	0	0	0	1		
Components diagram	1	0	0	0	0		
Used technologies and tools	1	0	0	0	0		
Solution testing	0	0	4	0	0		
Layout diagram	1.5	0	0	0	0		
Deployment instructions	1.5	0	0	0	0		
Meeting log	0.25	0	0	0	0		
Conclusion and future work	1	0	0	0	0		
References	0.25	0	0	0	0		

Continued on next page

Continued from previous page

	José Matos	**Francisco Antunes**	**Lourenço Carvalho**	**Maria Inês Costa**	**Rui Guimarães**		
Docker Integration	2.5	0	0	0	0		
Backend for Login and Registration	1.5	0	0	0	0		
Frontend for Login and Registration	2	0	0	4	0		
SQLite database creation	0.25	0	0	0	0		
SQLite connecting to the database	0.25	0	0	0	0		
Frontend for Landing Page	1.5	0	0	3	0		
Backend for Landing Page	0.5	0	0	0	0		
Frontend for Search	0	0	1	1	0		
Backend for Search	0	0	2	0	0		
Frontend for Profile	0	0	0	3	0		
Backend for Profile	0	0	2	0	0		
Frontend for Course	0	0	2	2	0		
Backend for Course	0	0	4	0	0		
Frontend for Teaching Unit/Materials	1	0	0	4	0		
Backend for Teaching Unit/Materials	1	2	0	0	0		
Frontend for Admin Features	1	0	0	1	0		
Backend for Admin Features	3	0	0	0	0		
Frontend for Edit Course	0	0	0	1	0		
Backend for Edit Course	0	0	1	0	0		
Frontend for Add Course	0	0	0	1	0		
Backend for Add Course	0	0	1	0	0		

Continued on next page

Continued from previous page

	José Matos	**Francisco Antunes**	**Lourenço Carvalho**	**Maria Inês Costa**	**Rui Guimarães**		
Frontend for Menu	0	0	0	1	0		
Backend for Menu	0	0	0	0	0		
Frontend for Edit Profile	0	0	1	1.5	0		
Backend for Edit Profile	1	0	0	0	0		
Frontend for Rate Course	0	0	0	1	0		
Backend for Rate Course	0	0	1	1	0		
Frontend for Courses Created	0	0	0	1.5	0		
Backend for Courses Created	0	0	1	0	0		
Frontend for Courses Enrolled	0	0	0	1.5	0		
Backend for Courses Enrolled	0	0	1	0	0		
Frontend for Add Teaching Unit	0	0	0	1.5	0		
Backend for Add Teaching Unit	0	1.5	0	0	0		
Frontend for Add Material	0	0	0	3	0		
Backend for Add Material	0	2	0	0	0		
Frontend for Remove Material	0	0	0	0.5	0		
Backend for Remove Material	0	1	0.5	0	0		
Frontend for Remove Teaching Unit	0	0	0	0.25	0		
Backend for Remove Teaching Unit	0	0	0.5	0	0		
Migrating to PostgreSQL	2	0	0	0	0		
Google Calendar integration	0	0	0	0	1		
Frontend Live Chat definitions	0	0	0	0.5	0.5		
Backend Live Chat definitions	0	0	0	0	3		

Continued on next page

Continued from previous page

	José Matos	**Francisco Antunes**	**Lourenço Carvalho**	**Maria Inês Costa**	**Rui Guimarães**		
Frontend Add Payment Details	0	0	0	1	0		
Frontend Add Payment Details	0	0	1	0	0		
Deploying on Render	4	0	0	0	0		
Migrating to PostgreSQL	2	0	0	0	0		

Change log diagrams

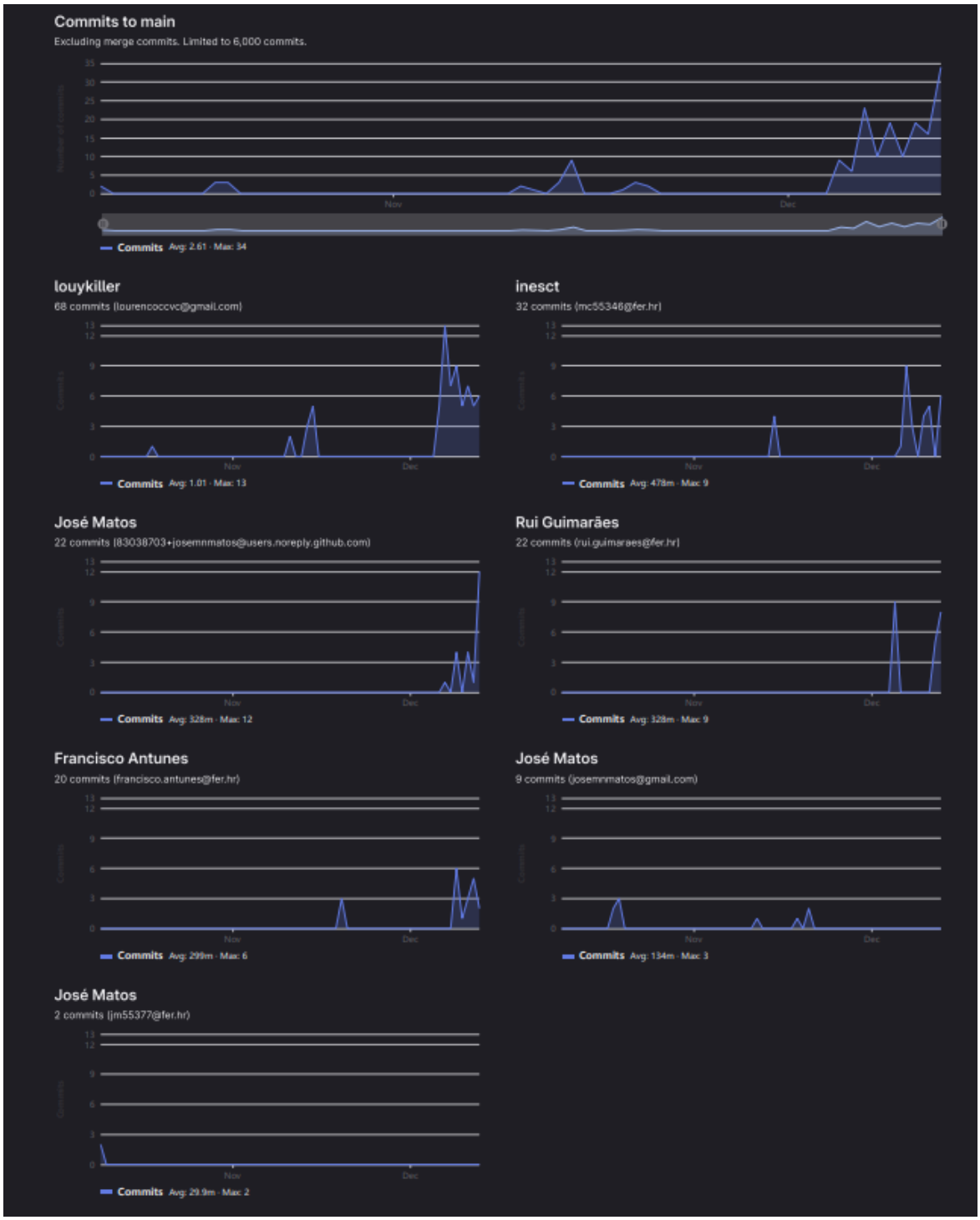

Figure 6.1: Main branch changelog

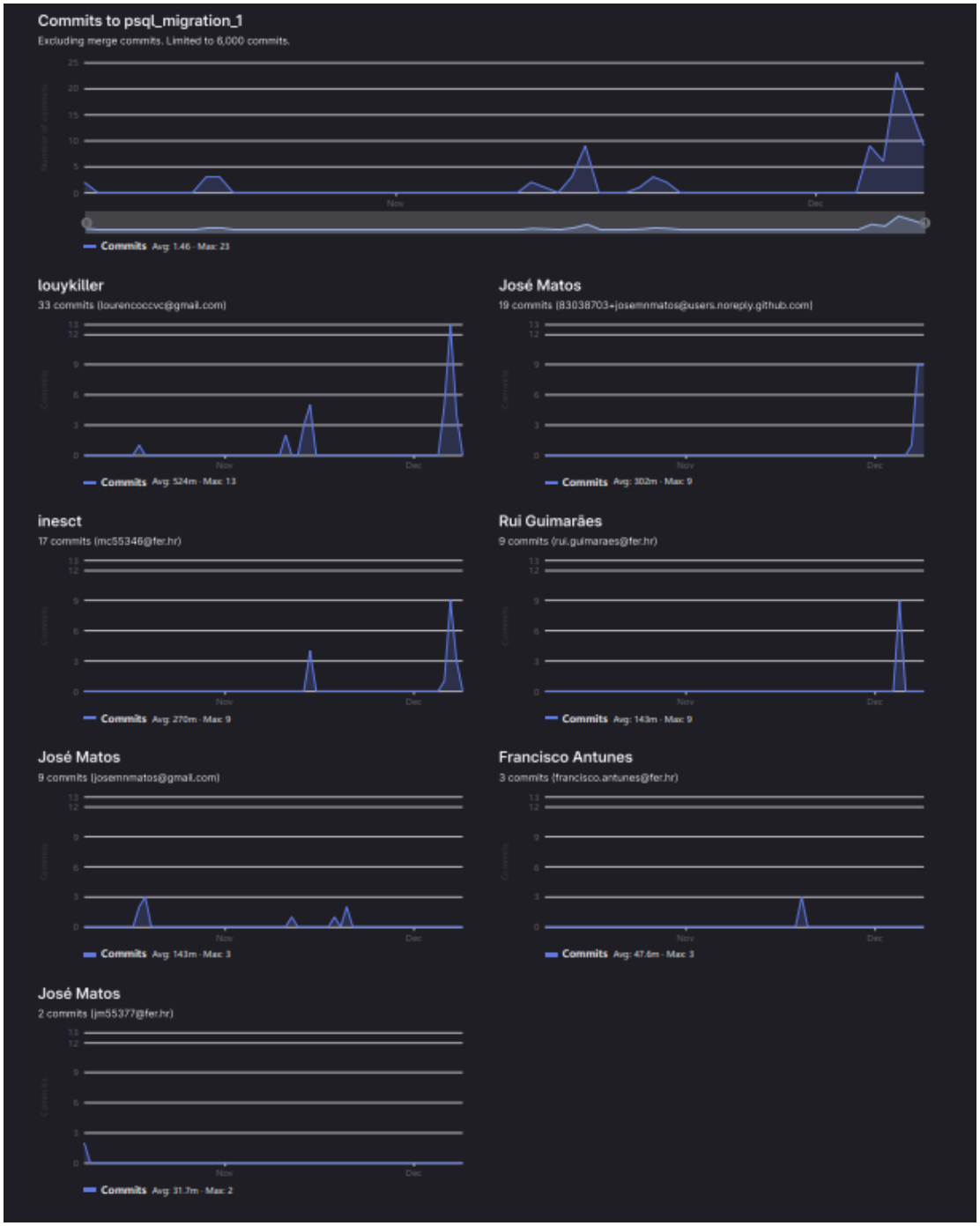

Figure 6.2: Secondary test branch changelog