UniProgrammes

Design Description

Sprint #1 – 15/10/2024

Date	Sprint #	Version #	Written by	Reviewed by
15/10/2024	1	0	Development Team	Ali Hajizadeh Idloo, Giuseppe Vitello

The rest of this page was intentionally left blank.

Summary

1. Introduction

1.1 Purpose

This document describes the design of UniProgrammes which is an interface to plan and analyze university programmes.

1.2 Intended Audience and Reading Suggestions

This document is for anyone who would like to understand the design aspects of the UniProgrammes project. The key audience are:

- Product and project management teams
- Development team

1.3 Scope

The scope of the document covers the architecture and design aspects required to build the UniProgrammes project. The following aspects will be covered in the document:

- System Architecture
- Database Description
- Graphical User Interface (GUI) Design
- List of APIs Required
- Technology Stack Selection
- Detailed Software Design

2. Overall Description

2.1 Product Perspective

This project focuses on developing a user-friendly interface that collects course and programme data from university systems and creates a visual representation of the program structure. The key purpose of this tool is to simplify and enhance the planning, organization, and analysis of university programs. The system will help users ensure that course prerequisites, learning outcomes, and degree requirements are met, while also providing the ability to modify the course sequence dynamically.

The system supports academic planning by:

- Visualizing Course Progression:

  The graphical tool will create a tree hierarchy of courses, allowing users to visualize the flow and dependencies of each course within the program. This will help in ensuring a logical and consistent progression across courses. The visualization will be based on period and/or by years.

- Ensuring Course Requirements:

  The tool will ensure that any modifications to the program maintain the integrity of the requirements for the intended degree by automatically updating and displaying course-specific requirements (such as prerequisites, main area, and learning outcomes).

- Learning Outcome Analysis:

  The tool will allow users to filter and analyze courses based on specific learning outcomes (e.g., group work, report writing), facilitating a more detailed understanding of the program.

- Programme Customization:

  The tool will enable the addition, removal, or replacement of courses, with real-time updates indicating whether course and program requirements remain satisfied.

- Educational Alignment:

  The tool's capability to automatically highlight whether course changes maintain educational goals and degree criteria ensures that any reorganization of courses still aligns with the broader objectives of the program.

This system, when implemented, will become an essential tool for university students and counselors, enabling them to effectively plan, restructure, and ensure that students' learning paths align with the required academic standards.

2.2 Process Flow

The process flow for the development of the UniProgrammes can be broken down into several key stages:

- Data Collection:

  The system receives data from the university, which includes course information (credits, educational level, prerequisites, main area, and learning outcomes).

  This data is parsed and stored in a central database, ensuring that all relevant information is accessible for further operations.

- Program Structure Creation:

  Using the data, the system generates a graphical program tree that visually represents the sequence of courses.

  Courses are arranged based on their prerequisites and progression, allowing users to see the hierarchical structure and flow of the program.

  The system also highlights the main area (e.g., specific subjects like ELA or DVA) and categorizes courses based on their learning outcomes (e.g., group work, report writing).

- Course Customization:

  Users can interact with the program structure by modifying courses. The system allows:

  - Adding new courses
  - Removing existing courses
  - Reordering courses to adjust the flow of the program

  As modifications are made, the system checks the course requirements to ensure that the prerequisites, credits, and learning outcomes are still valid.

- Real-time Validation:

  After any changes in the course structure, the system performs an automatic validation to verify:

  - Course prerequisites are satisfied.
  - The sequence of courses maintains a logical progression.
  - The program still meets the overall requirements for the intended degree (bachelor, master, etc.).

  If issues arise (e.g., missing prerequisites or unmet learning outcomes), the system highlights those courses for the user to adjust.

- Graphical Representation Update:

  The system updates the graphical representation of the program dynamically as changes are made.

  Users can toggle between different visualizations, such as:

  - Viewing the full program tree hierarchy.
  - Highlighting specific courses based on their main area (e.g., ELA or DVA courses).
  - Highlighting specific courses based on specific learning outcomes (e.g., all courses that emphasize group work).
  - Filtering courses by specific learning outcomes (e.g., all courses that emphasize group work)
- Final Output:

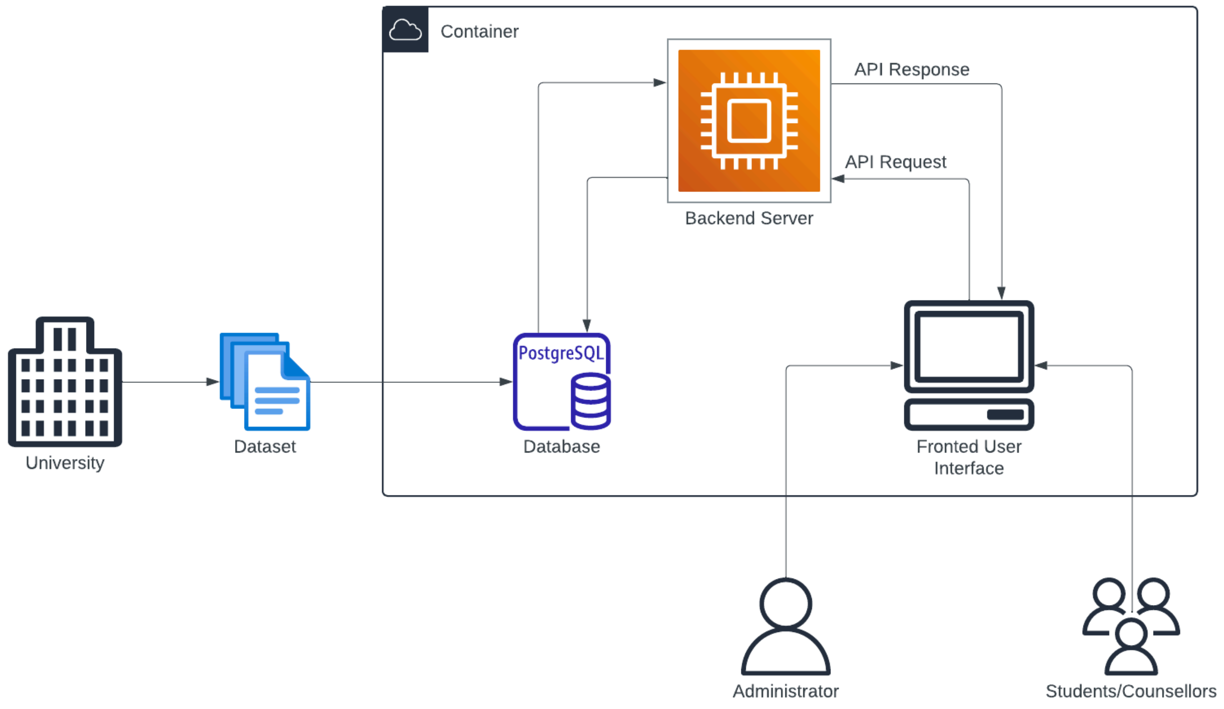
  Once users are satisfied with the program structure, the system produces a final graphical description of the entire program.

  This output can be used for further analysis or documentation purposes, ensuring that the program meets academic standards and provides a well-rounded learning experience for students.

2.3 System Architecture

2.3.1 Baseline

The architecture for the UniProgrammes is composed of several core components that work together to provide efficient program visualization and management. The system operates within a containerized environment that houses the backend server and a PostgreSQL database. The university dataset containing course information is loaded into the database, which serves as the primary data storage layer. The backend server handles all API requests and responses, interfacing with the frontend to provide real-time updates and interactions. The frontend user interface allows administrators, students, and counselors to visualize program structures, modify course sequences, and perform real-time validation of course prerequisites and outcomes. This architecture ensures smooth communication between the user interface and the underlying database, allowing seamless program planning and adjustments.



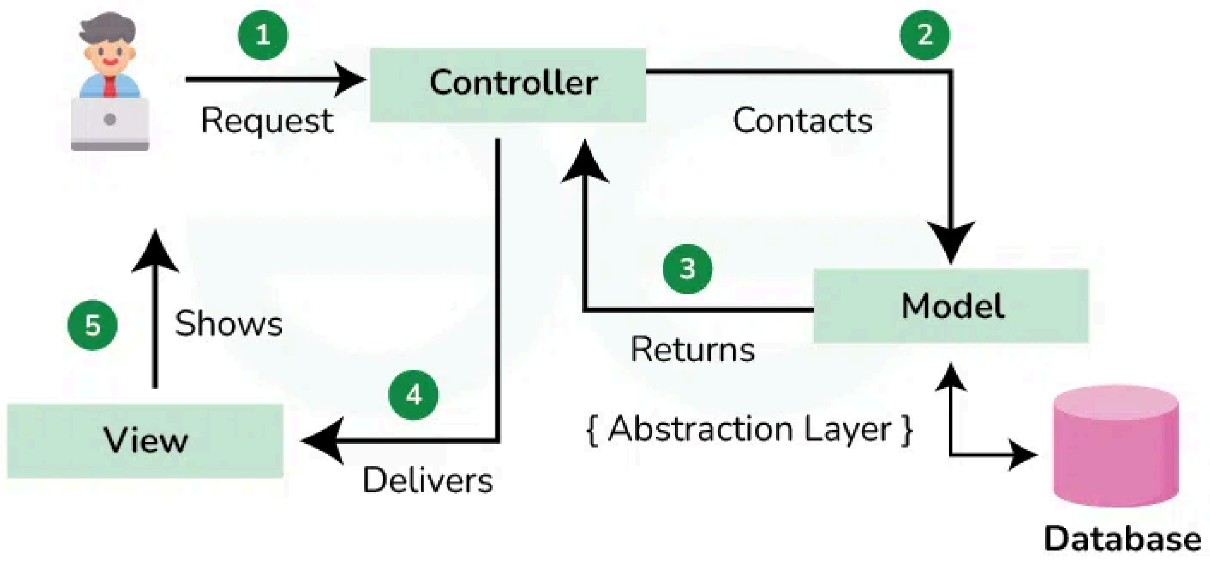
2.3.2 Design Pattern

MVC pattern on Backend. Using the **MVC (Model-View-Controller)** pattern provides a clear separation between business logic (Model), presentation (View), and application flow control (Controller). This separation:

1. Promotes a modular, maintainable architecture.
2. Allows isolated testing of each component, improving code quality.

3. Facilitates collaboration between developers and designers, as Views can change without affecting business logic.
4. Simplifies the implementation of dynamic, responsive user interfaces.

Given the existence of several versions of this pattern, we will decide the best during implementation. The idea of the schema:



Since the system will receive input data from different universities and each one of them will send it differently (for example JSON, XML, etc) we will use the **Pattern Adapter.**

Since there will be an exchange of messages we will use the **Visitor pattern.** The **Visitor Pattern** allows adding new operations to message types without altering their classes, making it easier to manage different message formats. It centralizes message-processing logic, improving code maintainability and scalability.

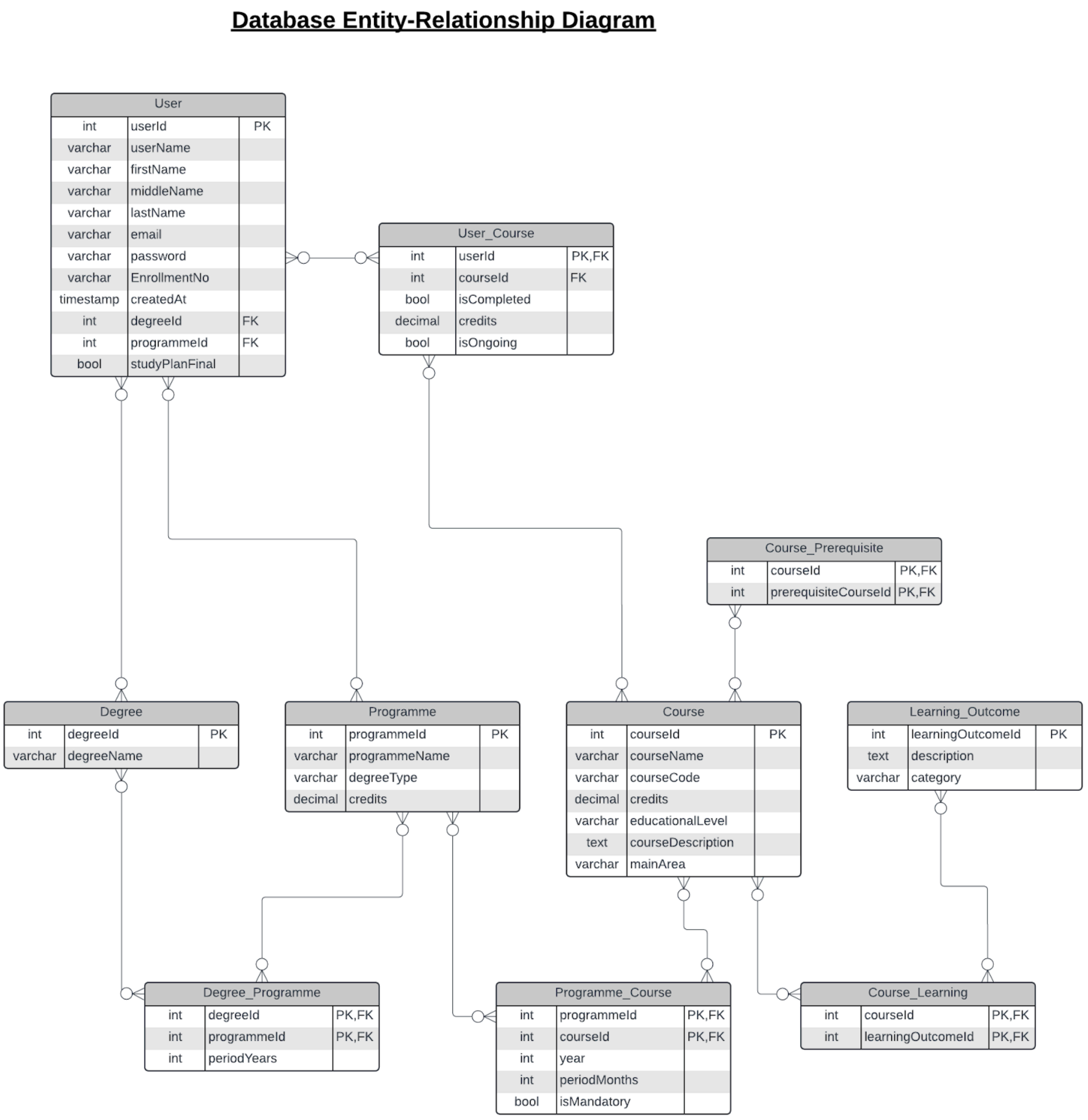
If during the implementation we will notice the existence of message hierarchy we will use the **Composite Pattern**. The **Composite Pattern** enables handling individual messages and message threads uniformly, simplifying hierarchy management.

3. Database Description

3.1 Introduction

The database system presented is designed to manage a university's academic programs and courses, providing an efficient way to track user interactions, course prerequisites, degree structures, and learning outcomes. This database is central to ensuring that students and administrators can effectively plan, analyze, and manage academic progress. It includes a range of features, from user authentication to tracking course completion and integrating degree program requirements. The database supports both students and administrators in managing course enrollments and validating learning outcomes. The database's relational structure ensures the integrity of the data and seamless access to all necessary academic information.

3.2 Database Schema



The database schema is composed of multiple interconnected tables that manage users, courses, programs, degree structures, and learning outcomes. The main entities in this schema include Users, Courses, Programmes, and Learning Outcomes. Each table is connected through foreign key relationships, ensuring referential integrity and allowing for the smooth retrieval of data across various entities.

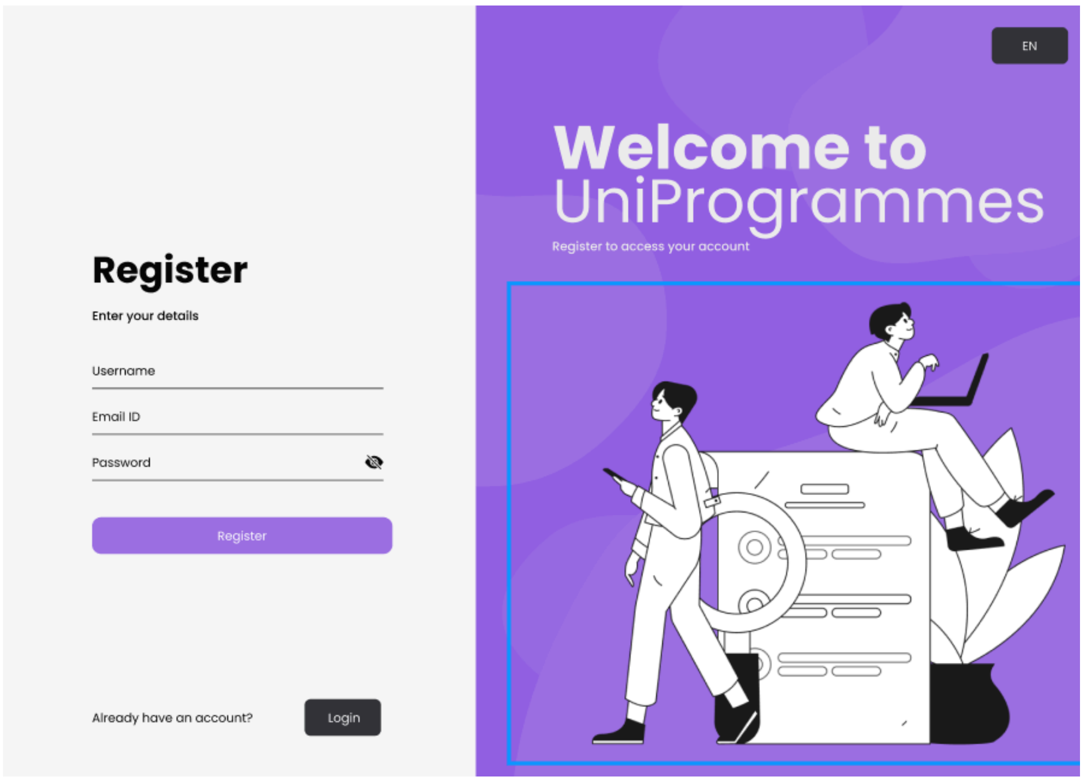
The schema also integrates Prerequisites and Course Learning, which are crucial to ensuring students follow the proper progression and meet the intended learning outcomes before moving forward.

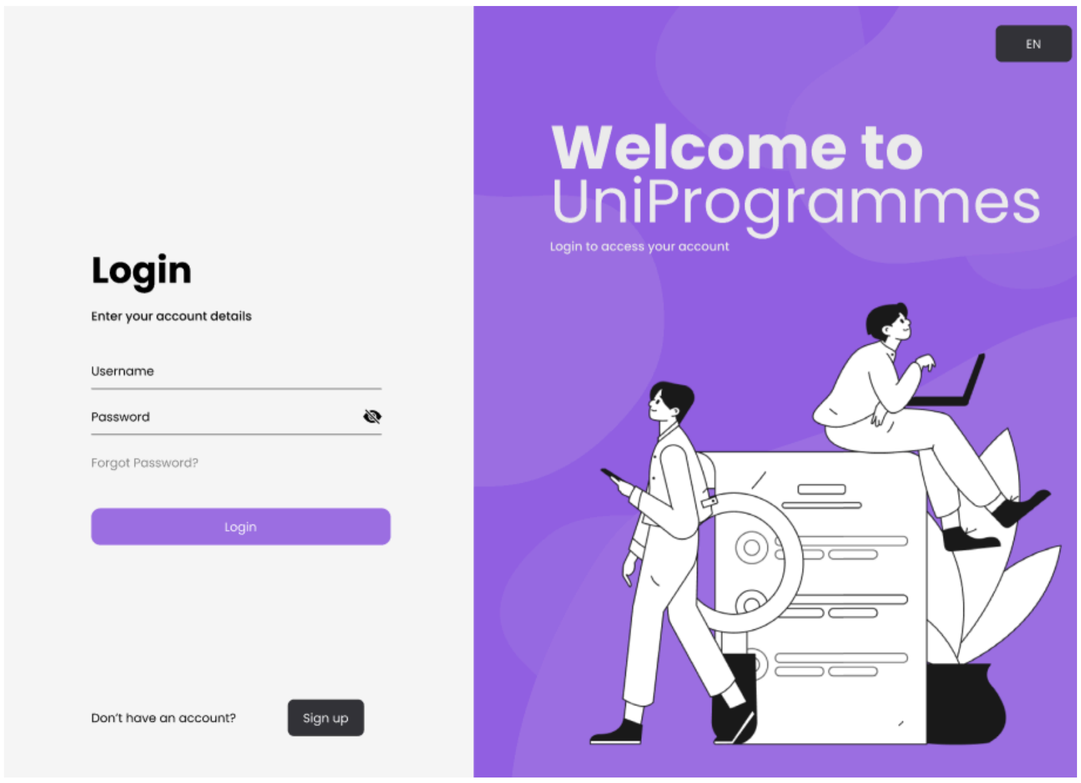
3.3 Database Tables Description

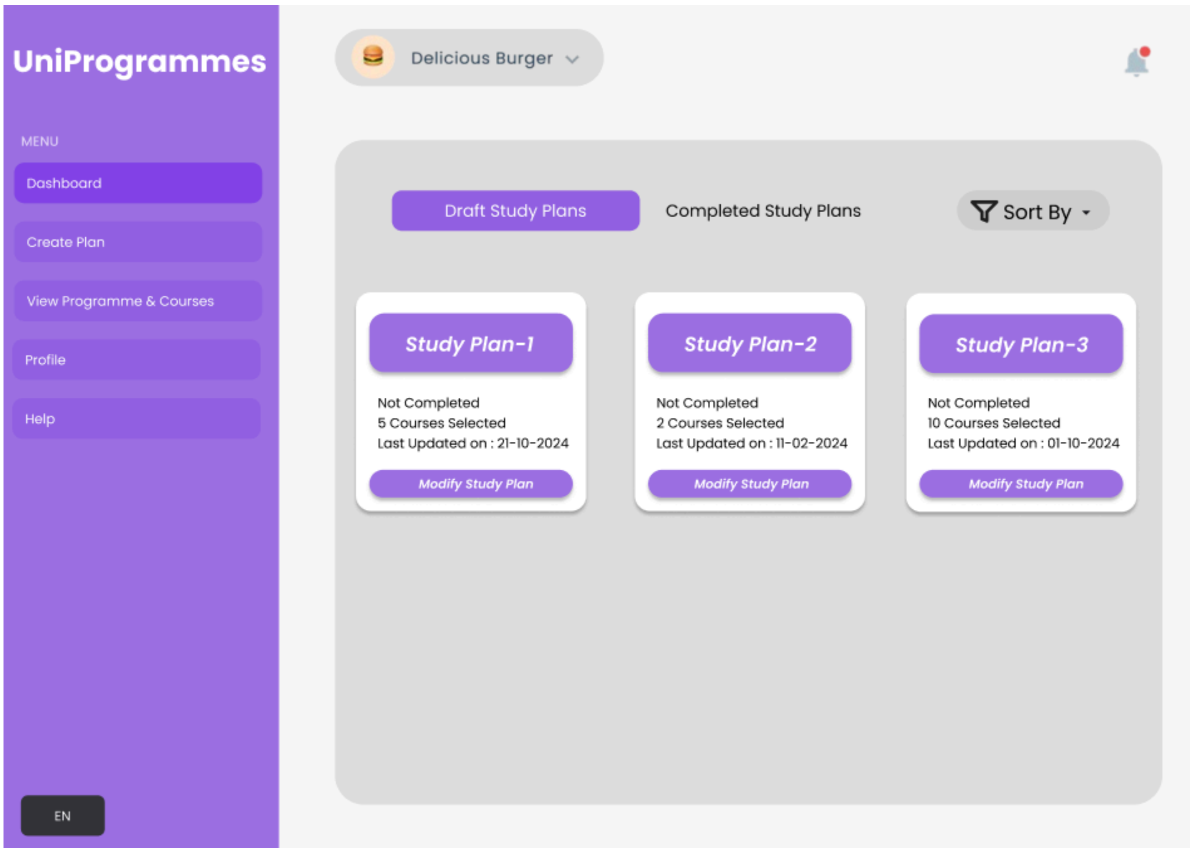
Here is a detailed description of the database schema:

- User: Stores basic user information, including usernames, email addresses, and enrollment details. This table differentiates between students and administrators within the system.
- User_Course: Tracks the relationship between users and courses. It captures whether a user has completed a course, how many credits were earned, and whether the course is ongoing.
- Degree: Defines the various academic degrees offered by the university, such as Bachelor’s or Master’s, along with the associated degree types.
- Programme: Represents each academic program, including its name, type, and credits required for completion. This table organizes programs by degree type and links to the courses available.
- Course: Contains core information about each course, such as course code, name, credits, and the main area of study. It includes a description and details about the educational level of the course (e.g., undergraduate or postgraduate).
- Course_Prerequisite: Manages the prerequisite requirements for each course, ensuring students complete necessary coursework before enrolling in advanced subjects.
- Learning_Outcome: Stores the specific learning objectives or outcomes for each course, categorized for easy tracking.
- Programme_Course: Defines the structure of academic programs by linking them to courses. This table specifies mandatory courses and their scheduling within the program.
- Course_Learning: Maps courses to their associated learning outcomes, ensuring that each course aligns with the program's educational goals.

4. Graphical User Interface (GUI) Design



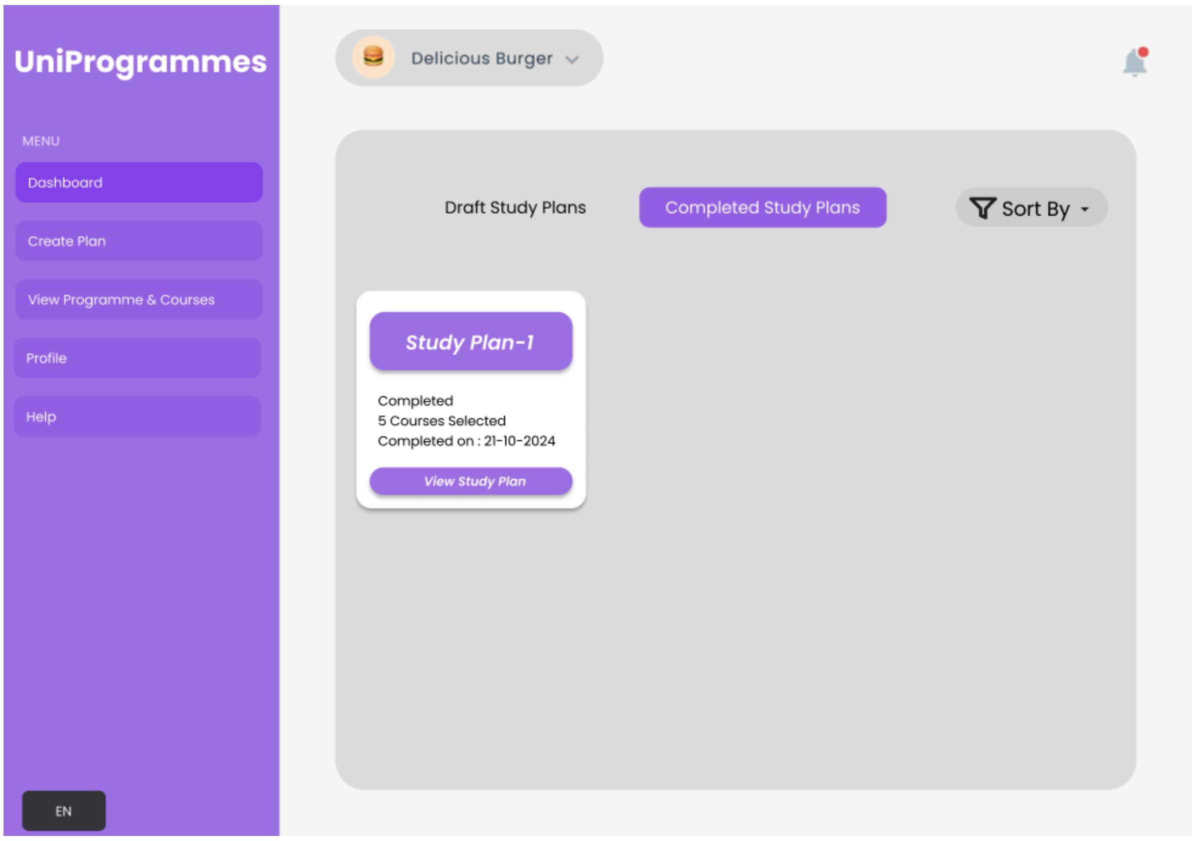




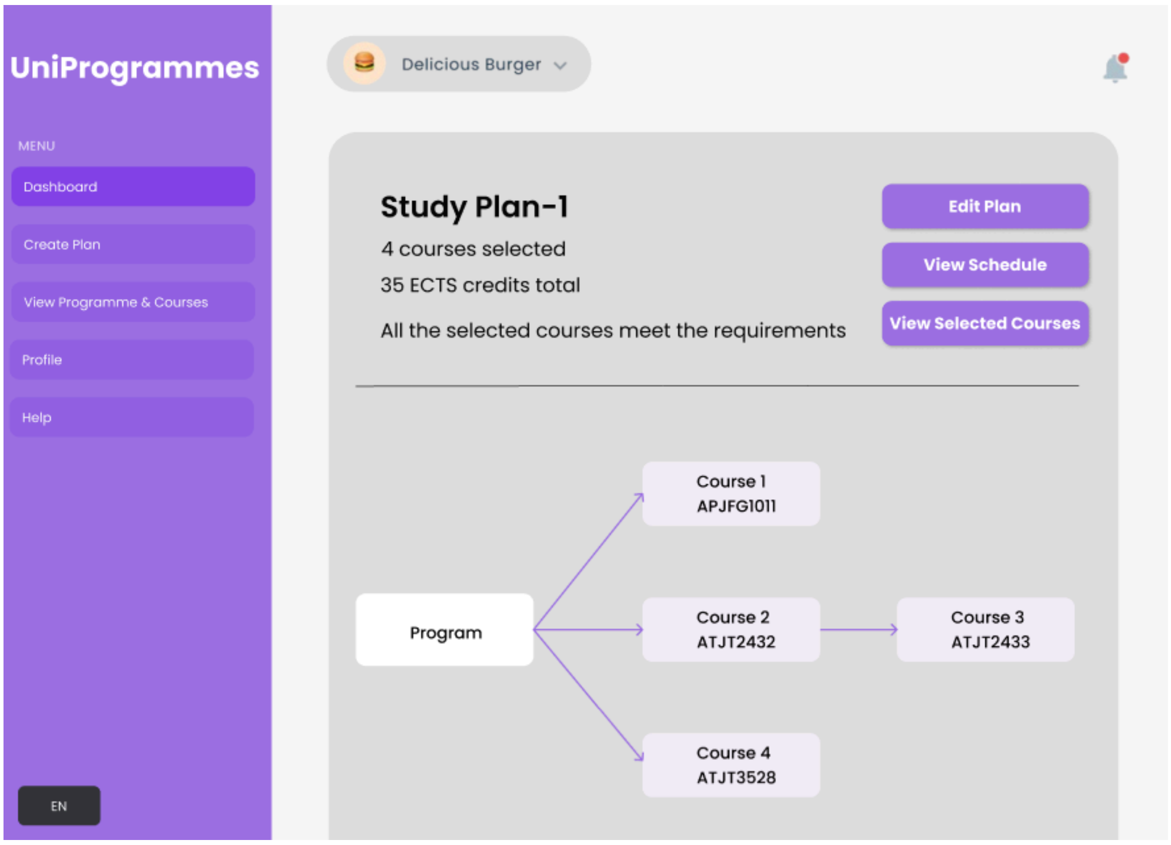
Study Plan-1
Not Completed
5 Courses Selected
Last Updated on : 21-10-2024

Study Plan-2
Not Completed
2 Courses Selected
Last Updated on : 11-02-2024

Study Plan-3
Not Completed
10 Courses Selected
Last Updated on : 01-10-2024



Study Plan-1
Completed
5 Courses Selected
Completed on : 21-10-2024

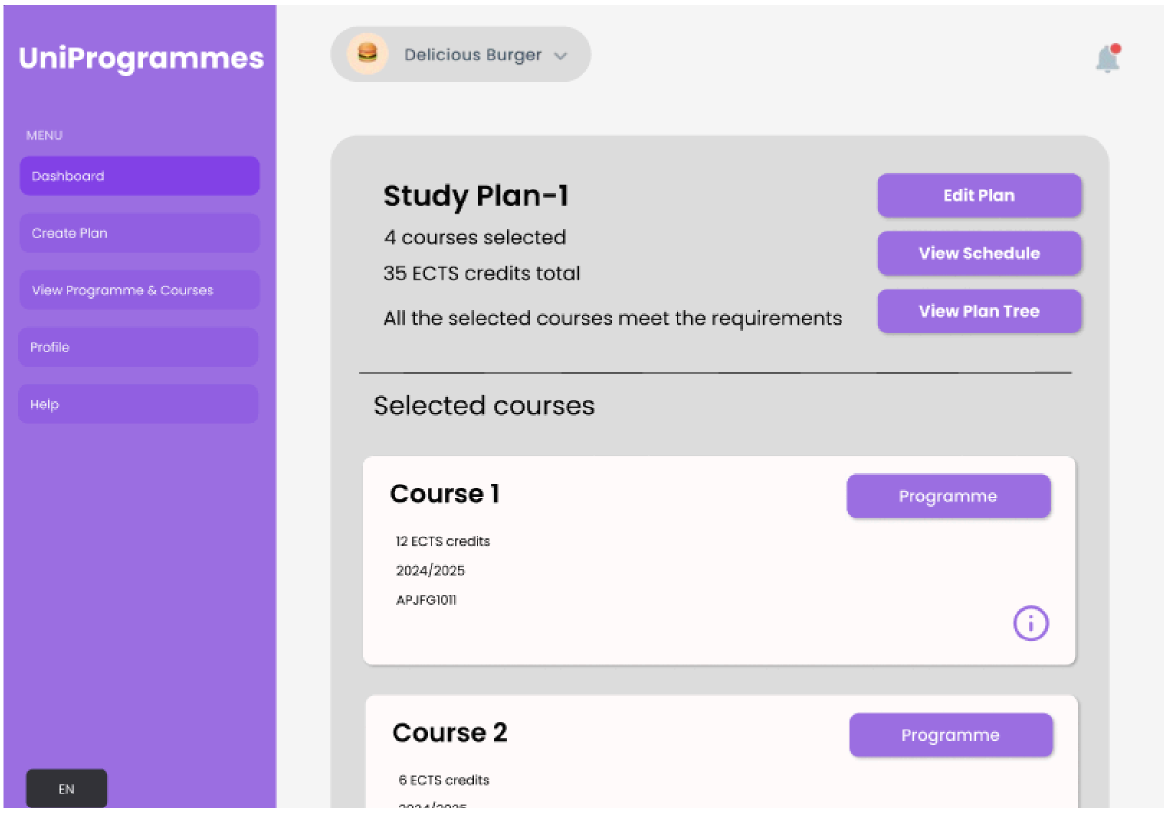


Study Plan-1

4 courses selected

35 ECTS credits total

All the selected courses meet the requirements



Study Plan-1

4 courses selected

35 ECTS credits total

All the selected courses meet the requirements

Selected courses

Course 1

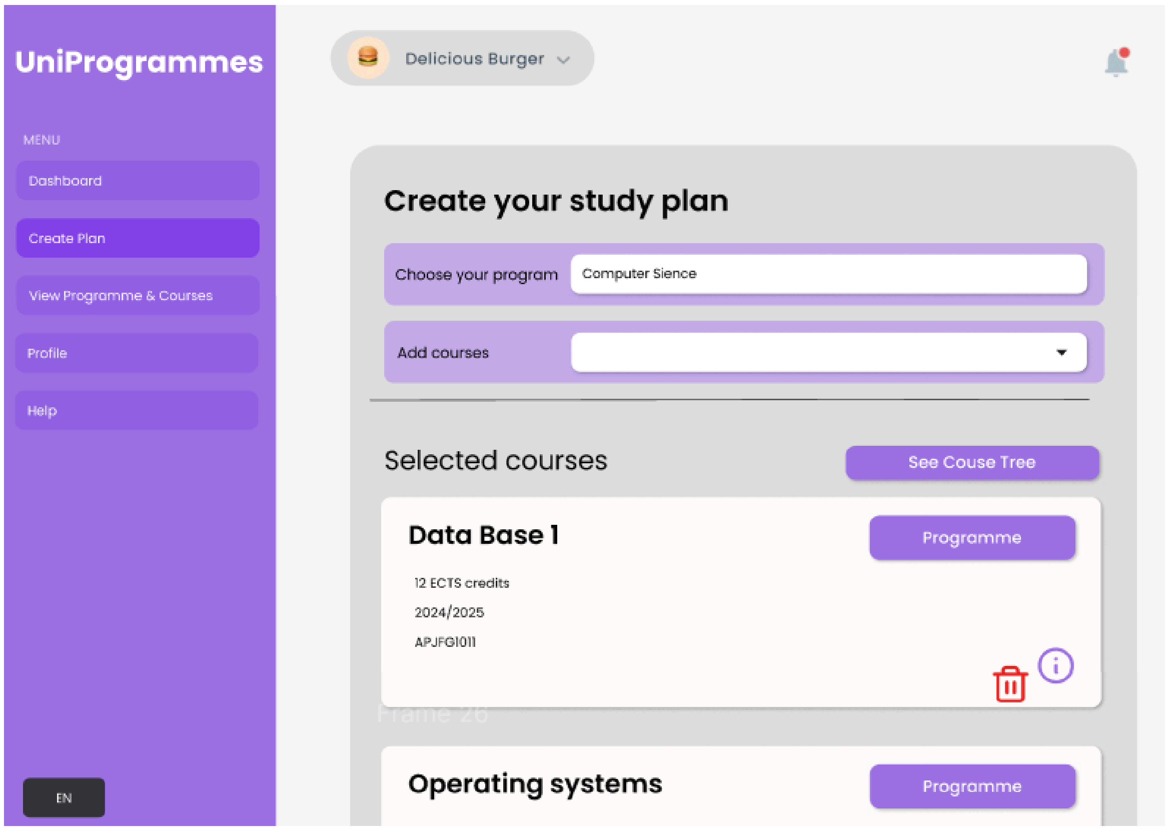
12 ECTS credits

2024/2025

APJFG1011

Course 2

6 ECTS credits



Create your study plan

Choose your program: Computer Sience

Add courses

Selected courses

See Couse Tree

Data Base 1

Programme

12 ECTS credits

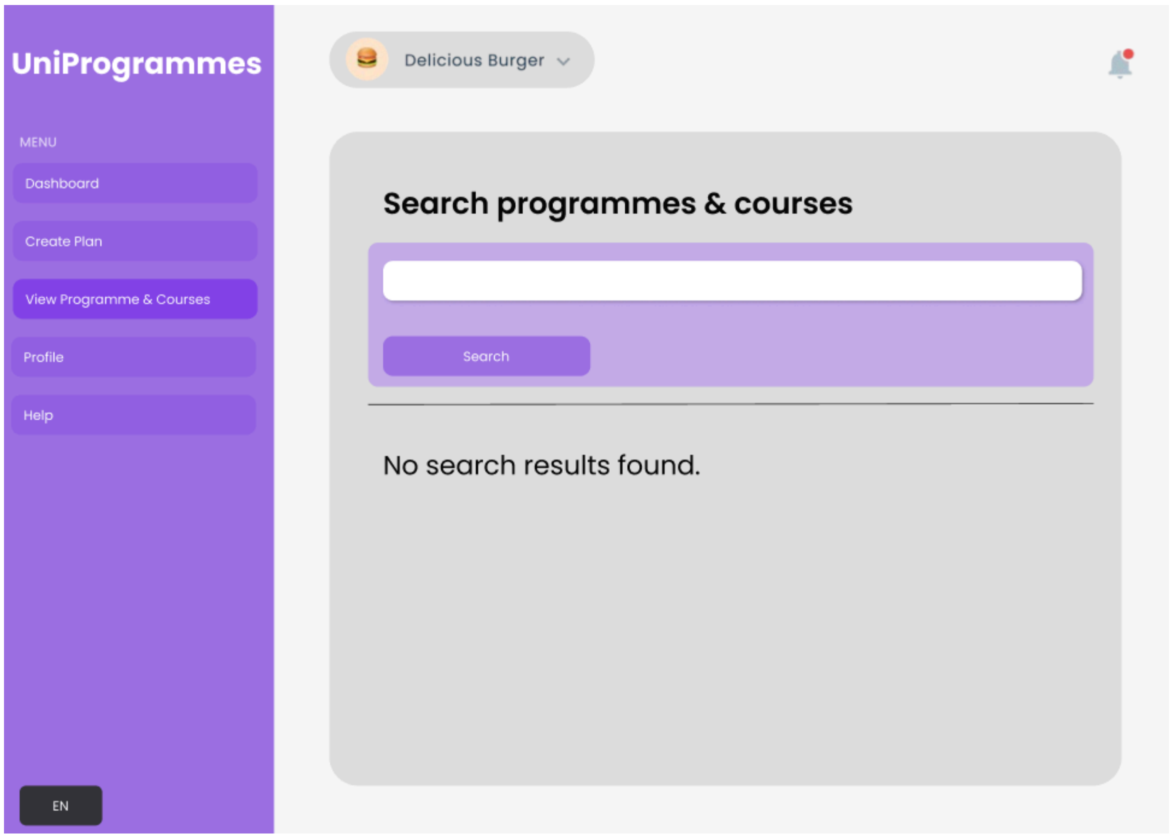
2024/2025

APJFGI011

Frame 26

Operating systems

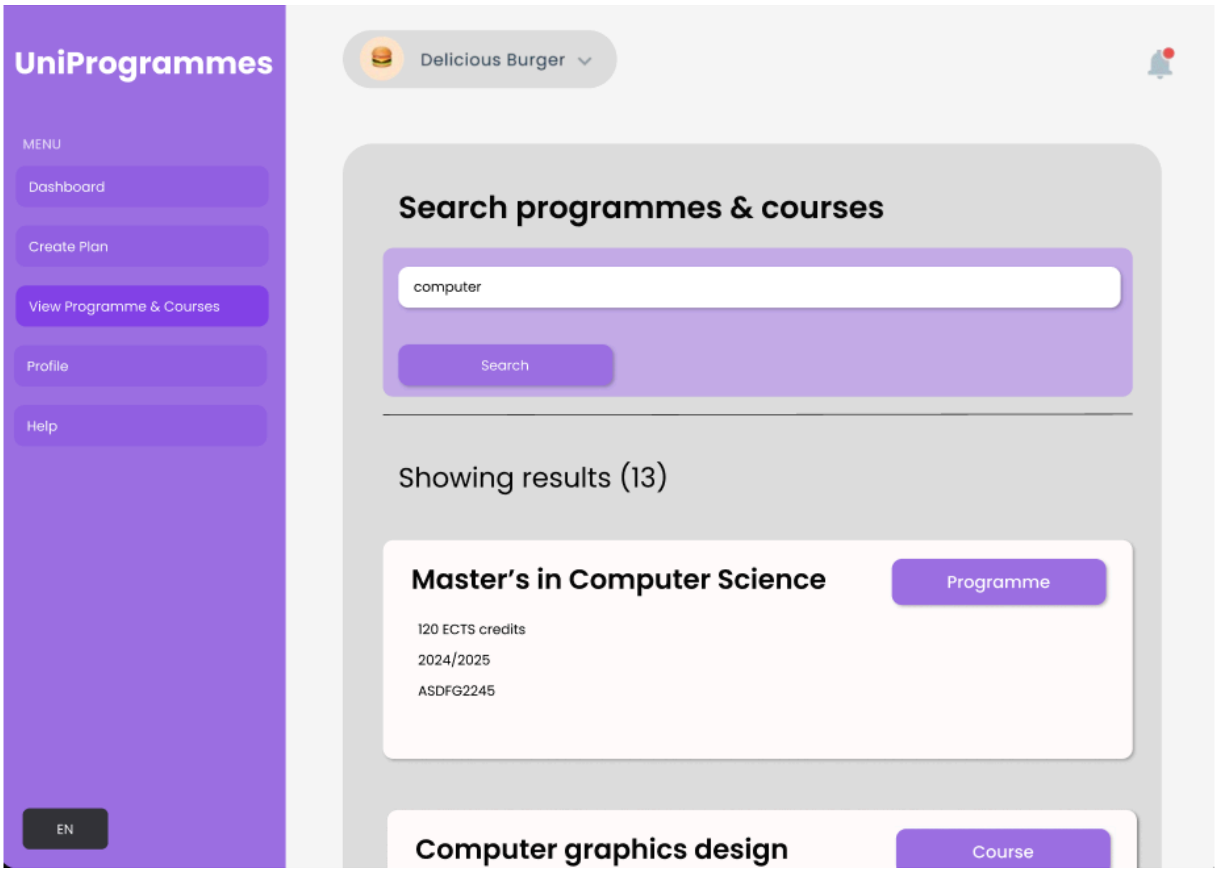
Programme



Search programmes & courses

Search

No search results found.



Search programmes & courses

computer

Search

Showing results (13)

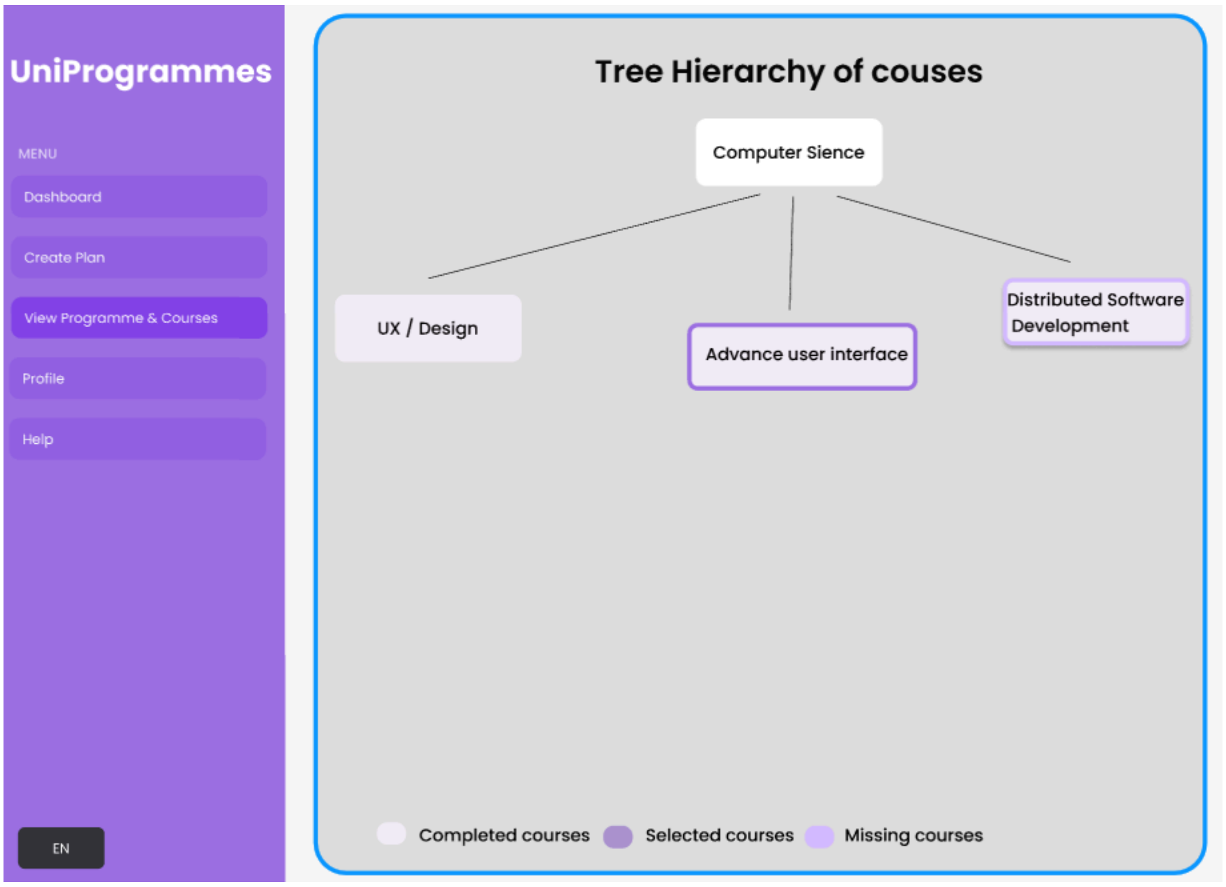
Master's in Computer Science — Programme

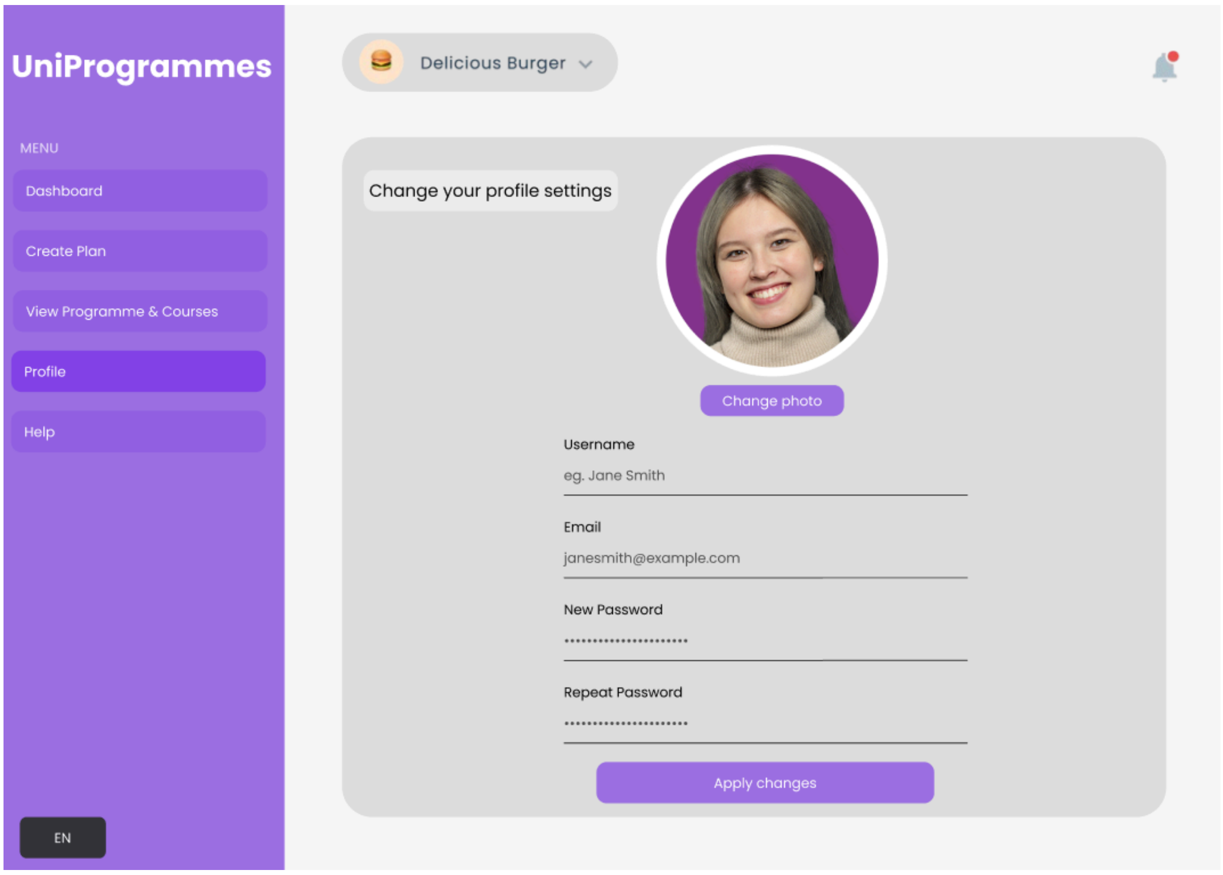
120 ECTS credits

2024/2025

ASDFG2245

Computer graphics design — Course





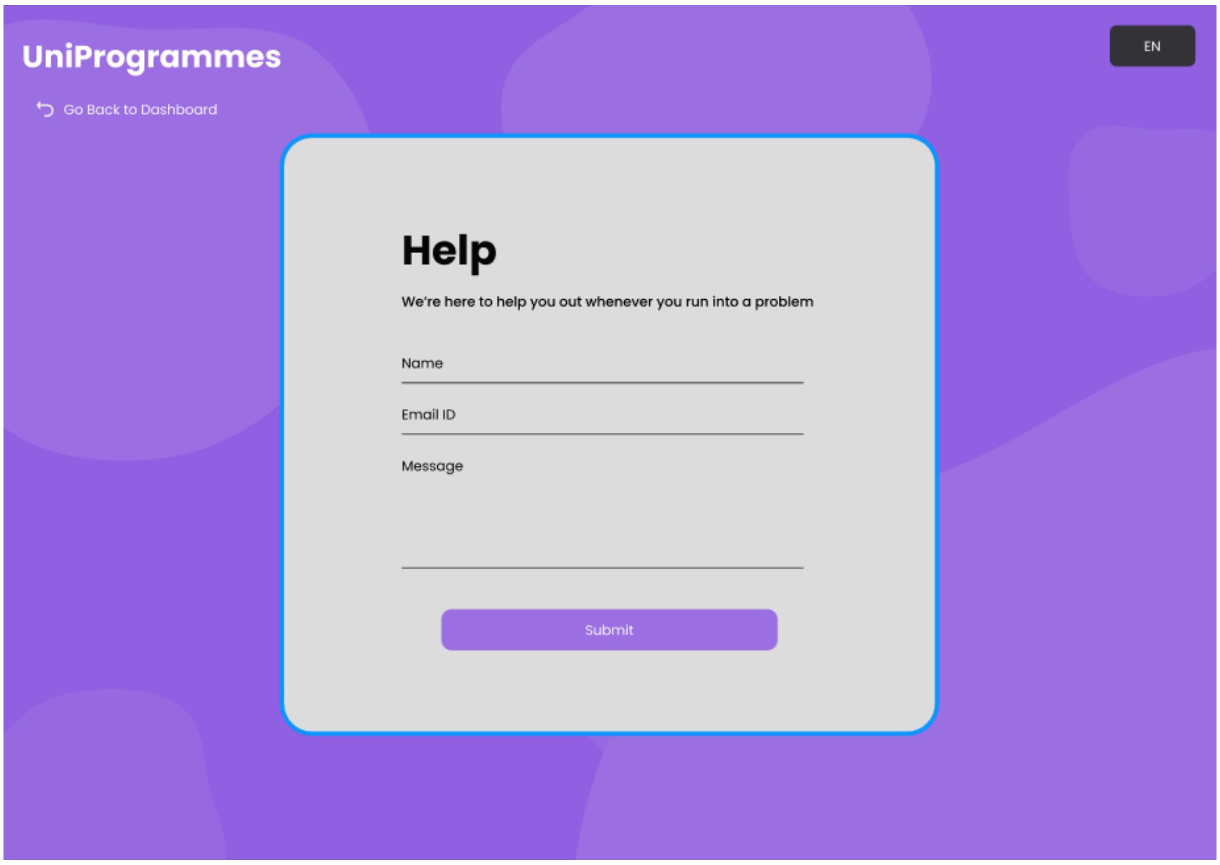
Change your profile settings

Username
eg. Jane Smith

Email
janesmith@example.com

New Password
•••••••••••••••••••••

Repeat Password
•••••••••••••••••••••



Help

We're here to help you out whenever you run into a problem

Name

Email ID

Message

5. List of APIs Required

Register user

Endpoint: POST /v1/users/register

Authentication: None

Payload:

```
{
  "email": string,
  "username": string,
  "password": string,
}
```

Responses:

- 201: User registered correctly.
- 400: Error
    - email_already_registered: The email already has an account
    - username_already_taken: The username has already been selected by another user.

Login

Endpoint: POST /v1/users/login

Authentication: None

Payload:

```
{
  "email": string,
  "password": string,
}
```

Responses:

- 201: Login successful. Returns user data serialized.

```
{
  "id": string,
  "username": string,
  "first_name": string,
  "middle_name": string,
  "last_name": string,
  "email": string,
  "enrollment_number": string,
  "created_at": datetime
}
```

- 400: Error
    - incorrect_password: The password is incorrect

Profile info

Endpoint: GET /v1/users/me

Authentication: User account

Payload: None

Responses:

- 200: Returns user data serialized.

```
{
  "id": string,
  "username": string,
  "first_name": string,
  "middle_name": string,
  "last_name": string,
  "email": string,
  "enrollment_number": string,
  "created_at": datetime
}
```

- 403: Forbidden. Incorrect authentication token.

Update profile info

Endpoint: PATCH /v1/users/me

Authentication: User account

Payload: Partial user info needed to be updated

Responses:

- 200: Returns updated user data serialized.

```
{
  "id": string,
  "username": string,

  "first_name": string,

  "middle_name": string,

  "last_name": string,

  "email": string,

  "enrollment_number": string,

  "created_at": datetime

}
```

- 403: Forbidden. Incorrect authentication token.

Retrieve my study plans

Endpoint: GET /v1/users/me/study-plans

Query params:

- status: Status of the study plan. One of [“draft”, “completed”]

Authentication: User account

Payload: None

Responses:

- 200: Returns saved study plans from the user.

```
[

  {

    "name": string,

    "status": string,

    "created_at": string,

    "updated_at": string,

  },

  ...

]
```

- 403: Forbidden. Incorrect authentication token.

View study plan

Endpoint: GET /v1/study-plans/{id}

Authentication: User account of the user associated to the plan

Payload: None

Responses:

- 200: Returns serialized study plan information.

```
{
  "id": int
  "name": string,
  "status": string,
  "created_at": string,
  "updated_at": string,
  "courses": [...]
}
```

- 403: Forbidden. Incorrect authentication token.

Create study plan

Endpoint: POST /v1/study-plans/

Authentication: User account

Payload: Study plan data

```
{
  "name": string,
  "status": string
}
```

Responses:

- 200: Successful. Returns created study plan data.

```
{
  "id": string,
  "name": string,
  "status": string,
  "created_at": string,
  "updated_at": string,
  "courses": [...]
}
```

- 403: Forbidden. Incorrect authentication token.

Modify study plan

Endpoint: PATCH /v1/study-plans/{id}

Authentication: User account of the user associated to the plan

Payload: Partial study plan info to be updated.

Responses:

- 200: Successful. Returns updated study plan data.

```
{
  "id": string,
  "name": string,
  "status": string,
  "created_at": string,
  "updated_at": string,
  "courses": [...]
}
```

- 403: Forbidden. Incorrect authentication token.

Delete study plan

Endpoint: DELETE /v1/study-plans/{id}

Authentication: User account of the user associated with the plan

Payload: None

Responses:

- 204: No Content. Successfully deleted the study plan.
- 403: Forbidden. Incorrect authentication token.

List/Search courses

Endpoint: GET /v1/courses/

Query params:

- name: Name of the course to be searched.
- semester: Semester in which the course is held. For instance: “1; 2”

Authentication: User account

Payload: None

Responses:

- 200: List of the courses that comply with the given filters.

```
[
  {
    "id": int,
    "name": string,
    "code": string,
    "credits": int,
    "educational_level": string,
    "course_description": string,
    "main_area": string
  },
  ...
]
```

- 403: Forbidden. Incorrect authentication token.

Add course to study plan

Endpoint: POST /v1/study-plan/{id}/courses/

Authentication: User account associated with the study plan

Payload:

```
{
  "course_id": string,
}
```

Responses:

- 204: No content. Course added successfully to the study plan
- 403: Forbidden. Incorrect authentication token

Remove course from study plan

Endpoint: DELETE /v1/study-plan/{id}/courses/{course_id}

Authentication: User account associated with the study plan

Payload: None

Responses:

- 204: No content. Course removed successfully from the study plan
- 403: Forbidden. Incorrect authentication token

Create programme

Endpoint: POST /v1/programmes/

Authentication: Admin

Payload:

```
{
  "name": string,

  "degree_type": string,

  "credits": int,

}
```

Responses:

- 201: Created. Returns the created programme.

```
{

  "id": int,

  "name": string,

  "degree_type": string,

  "credits": int,

}
```

- 403: Forbidden. Incorrect authentication token

List programmes

Endpoint: GET /v1/programmes/

Authentication: User

Payload: None

Responses:

- 201: Returns list of all programmes.

```
[

  {

    "id": int,

    "name": string,

    "degree_type": string,

    "credits": int,

  }

  ...

]
```

- 403: Forbidden. Incorrect authentication token

Delete programme

Endpoint: DELETE /v1/programmes/{id}

Authentication: Admin

Payload: None

Responses:

- 204: No content. Programme deleted successfully
- 403: Forbidden. Incorrect authentication token

Create course

Endpoint: POST /v1/courses/

Authentication: Admin

Payload:

```
{
  "name": string,
  "code": string,
  "credits": int,
  "educational_level": string,
  "description": string,
  "main_area": string
}
```

Responses:

- 201: Created. Returns the created course.

  ```
  {
    "id": int,
    "name": string,
    "code": string,
    "credits": int,
    "educational_level": string,
    "description": string,
    "main_area": string
  }
  ```

- 403: Forbidden. Incorrect authentication token

Delete course

Endpoint: DELETE /v1/courses/{id}

Authentication: Admin

Payload: None

Responses:

- 204: No Content. Course deleted successfully
- 403: Forbidden. Incorrect authentication token

Add course to programme

Endpoint: POST /v1/programme/{id}/courses

Authentication: Admin

Payload:

```
{
  "course_id": string
}
```

Responses:

- 204: No Content. Course associated to programme correctly
- 403: Forbidden. Incorrect authentication token

Remove course from programme

Endpoint: DELETE /v1/programme/{id}/courses/{id}

Authentication: Admin

Payload: None

Responses:

- 204: No Content. Course removed from programme correctly
- 403: Forbidden. Incorrect authentication token

6. Technology Stack Selection

Based on some first team-based decisions, this is the technology stack that we are gonna use to develop the product. Later changes can be made if the team requires them.

Frontend	Vite, ReactJS
Backend	Django
Database	PostgreSQL
Testing	Jest, pytest
Version control	Git
CI/CD	GitHub Actions
Containerization	Docker
Linters	ESLint, Black, flake8
Scrapper	Selenium

7. Detailed Software Design

Comprehensive explanations for each element of the design will be added in subsequent updates and refreshed at the start of every sprint to align with the latest implementation. Detailed descriptions will capture both static and dynamic aspects of the architecture, including class structures, component interactions, and key algorithms. By iterating on these details, the design documentation will serve as a precise and reliable reference for each component’s role and behavior within the system.

The design documentation will be regularly updated to ensure it accurately reflects the system’s present configuration. This living document approach provides ongoing alignment with development progress, enabling the team to address evolving requirements and maintain a high level of transparency and coherence across all phases of the project.