
Università
degli Studi di
Messina

DIPARTIMENTO DI INGEGNERIA

# TECHNICAL MANUAL

**Jaime Hernández Pérez**

SICUE

Intercambio Estudiantes
Universidades Españolas


https://github.com/JaimeHP05/Software-Engineering-ERASMUS.git

# Table of contents

Introduction ..... 3
Specific Requirements (IEEE 830) ..... 5
User Stories ..... 7
Use Cases ..... 10
Class Diagram ..... 18
Validation Matrices [Requirements vs. Use Cases] ..... 23
Validation Matrices [Classes vs. Use Cases] ..... 24
Testing ..... 25
Data/Repository/DB-oriented Functionalities ..... 26
Third-Party Services ..... 26
Complex Functionalities ..... 27
Sprint Burndown Analysis ..... 31
Sprint 1: Core Architecture & Data Foundation ..... 32
Sprint 2: Workflows & Third-Party Integration ..... 33
Sprint 3: Intelligence & Analytics ..... 34
Sprint 4: Future Updates ..... 35
Conclusion ..... 36

# Introduction

## Project Overview

This website works as a SICUE / Erasmus platform for student interchange with other universities in Spain (for SICUE) or in all of Europe (Erasmus).

The student can login with their university credentials and create their own learning agreement. Then the administrators can accept or deny the agreement and the student will get notified with the resolution.

There are more functionalities that add flavour to the website, but they will be explained further in the report.

The platform is designed to facilitate the entire lifecycle of a Learning Agreement, from the initial course selection by the student to the final validation by the academic administrator.

## Software Development Plan

The SDP chosen has been Scrum. The project required a flexibility to add more libraries from 3rd parties (PDF, Email, AI) as the user needs were discovered. Thus, the project will be divided in three Sprints (in case more modifications and additions are needed, a 4th sprint will be created).

Each sprint will get a section in the presentation, showing its objectives, functionalities added, and their UML artifacts (User Case and Class diagrams). A Trello recreation of the project will also be added to show how effective the Scrum method is at organizing a project with divided sprints.

## Core System Modules

The application is composed of five key functional modules:

1. **Authentication & Role-Based Access Control (RBAC):** Manages secure user access, strictly differentiating between Students (who generate agreements) and Administrators (who validate them and manage system data).
2. **Intelligent Agreement Management:** Allows for the creation and tracking of learning agreements. This module includes an AI-based Recommendation Engine that assists students in finding equivalent courses via text similarity algorithms.
3. **Automated Documentation & Notification:** A service layer that automatically generates official PDF Learning Agreements and dispatches transactional Emails to users upon status changes (Draft → Approved/Rejected).
4. **Administrative & Analytical Dashboard:** Provides administrators with tools to manage academic entities (Universities, Degrees, Courses) and visualizes system performance through interactive statistical charts.
5. **Data Persistence Layer:** A relational database structure that ensures the integrity of user profiles, academic data, and agreement history.

## Technology Stack

The system has been developed using a modern, scalable stack:

- **Backend:** Node.js with Express.js for RESTful API architecture.
- **Database:** SQLite for relational data storage.
- **Frontend:** HTML5, Vanilla JavaScript (ES6+), and Tailwind CSS for a responsive user interface.
- **Third-Party Integrations:**
    - PDFKit for dynamic document generation.
    - Nodemailer for SMTP email communication.
    - String-Similarity for the AI matching algorithm.
    - Chart.js for data visualization.
- **Testing:** Jest and Supertest for automated integration testing.

# REQUIREMENTS ANALYSIS

## Specific Requirements (IEEE 830)

### 1. External Interfaces (IE)

- **IE1:** The system shall generate and provide a PDF document automatically upon agreement creation.
- **IE2:** The system shall integrate with an SMTP email server to send transactional notifications (creation, approval, rejection) with the generated PDF attached.

### 2. Functional Requirements (F)

**Authentication & Roles**

- **F1:** The system shall allow users to authenticate using a unique username and password.
- **F2:** The system shall distinguish between two user roles: Student and Administrator.
- **F3:** The system shall maintain a user session and allow users to securely logout.

**Student Functionalities**

- **F4:** The system shall present a student Dashboard displaying a list of all agreements associated with the user, showing their current status and duration.
- **F5:** The system shall allow students to create new agreements. This involves selecting a destination, the duration, and the course pairs.
- **F6:** The system shall provide an AI-based recommendation feature that suggests the best matching destination course based on the origin course using string similarity algorithms.
- **F7:** The system shall filter available courses based on the degree and university.

**Administrator Functionalities**

- **F8:** The system shall provide an Admin Dashboard with statistical visualization charts showing the most popular destinations, and the distribution of agreements.
- **F9:** The system shall allow administrators to search for agreements by student name or username and view pending agreements by default.
- **F10:** The system shall allow administrators to manage agreements by changing the status to Approved or Rejected. This action sends an email update to the student.
- **F11:** The system shall allow administrators to manage data (CRUD). They must be able to create new users / universities / degrees / courses
- **F12:** The system shall allow administrators to publish or delete announcements. These can be sent to specific universities or degrees.

## 3. Performance Requirements (RR)

- **RR1:** The system shall handle a growing number of universities, students, announcements, degrees, and agreements, as well as multiple simultaneous requests without negatively affecting its performance.
- **RR2:** The system shall always be available, ensuring that students and administrators can access it at any time.

## 4. Design Constraints (RD)

- **RD1:** The user interface shall be web-based, responsive, and compatible with modern browsers.
- **RD2:** The system shall use JavaScript for both frontend (Vanilla JS) and backend (Node.js/Express) to ensure consistency.
- **RD3:** The visual interface shall adhere to a clean, professional design using the Tailwind CSS framework.

## 5. System Attributes (AS)

- **AS1:** The system shall securely manage user credentials.
- **AS2:** The system shall require an active internet connection to perform external operations (Email).
- **AS3:** The code structure shall separate concerns between the API (Backend), Database (SQLite), and Client (Frontend) to facilitate future updates.

# User Stories

## US1: User Authentication & Role-Based Access

- **As a:** User (Student or Administrator)
- **I want to:** Log in to the platform using my credentials and be redirected to my specific dashboard.
- **Description:** The system must validate my username and password. If I am a Student, I should land on the “My Agreements” dashboard to see my applications. If I am an Administrator, I should land on the “Manage Agreements” panel. The sidebar menu must hide or show buttons depending on my role.
- **Priority:** High
- **Duration:** 1 week
- **Sprint:** 1

## US2: Academic Data Management (CRUD)

- **As an:** Administrator
- **I want to:** Create and manage Universities, Degrees, and Courses within the system.
- **Description:** I need forms to add new universities and degrees. When adding a Course, I must be able to specify its ECTS credits and the period so that students have accurate information for their agreements. I also need to register new Users into the database using the data from their university.
- **Priority:** High
- **Duration:** 1 week
- **Sprint:** 1

## US3: Agreement Creation with Course Pairing

- **As a:** Student
- **I want to:** Create a new learning agreement by selecting a destination and validating specific subjects.
- **Description:** When I select a Destination University, the system must load the available courses. I need to be able to define the Duration of my stay (One Semester/Full Year) and add Course Pairs (mapping one origin subject to one destination subject). The system must prevent me from submitting an empty agreement without course pairs.
- **Priority:** High
- **Duration:** 1 week
- **Sprint:** 2

## US4: Admin Dashboard & Decision Making

- **As an:** Administrator
- **I want to:** Search for specific agreements and approve or reject them.
- **Description:** I need a dashboard that lists all pending agreements by default. I must have a Search Bar to find agreements by student name. For each agreement, I need to see the details and have “Approve” and “Reject” buttons to update the status instantly.
- **Priority:** Medium
- **Duration:** 1 week
- **Sprint:** 2

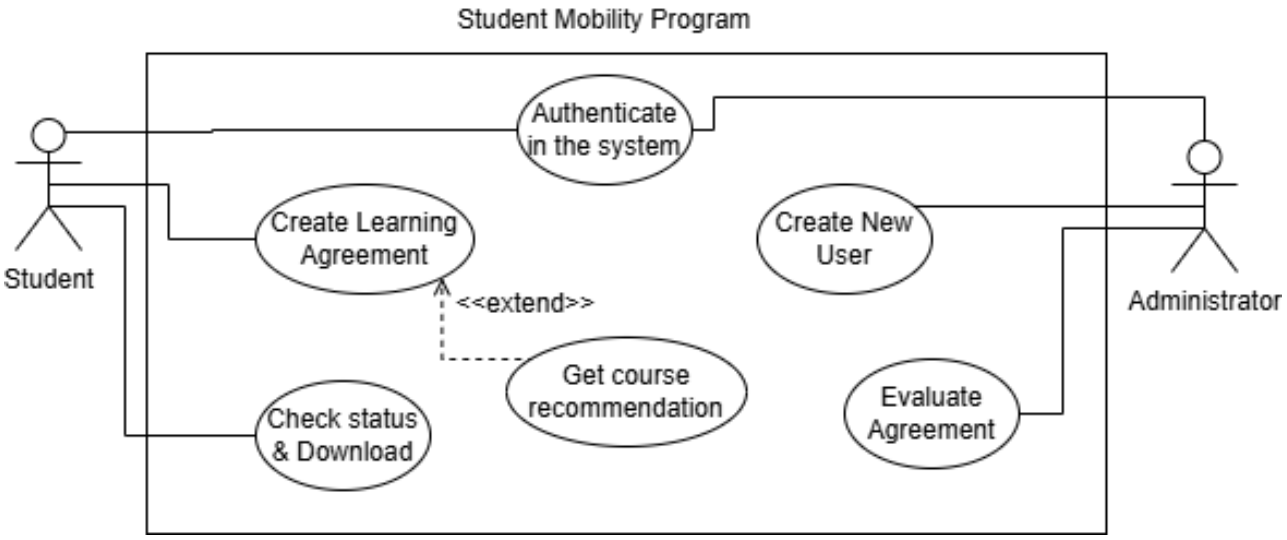
## US5: Automated Documentation & Notification

- **As a:** Student
- **I want to:** Automatically receive an official PDF of my agreement and an email notification when I create or update a request.
- **Description:** When I submit an agreement, the system must dynamically generate a PDF Learning Agreement (with tables calculating credit totals) and download it to my browser. Simultaneously, I expect to receive an Email with the PDF attached confirming the action. If the Admin changes the status, I want to receive a new email with the updated PDF.
- **Priority:** Medium
- **Duration:** 1 week
- **Sprint:** 3

## US6: AI-Powered Course Recommendation

- **As a:** Student
- **I want to:** Receive automatic suggestions for destination courses that match my origin courses.
- **Description:** To save time searching through long lists, when I select an origin course, I want a button ("Auto-Match") that uses an algorithm to analyse text similarity and automatically select the best matching course in the destination.
- **Priority:** Low
- **Duration:** 1 week
- **Sprint:** 3

# Use Cases




*Use Case Diagram*

This section will show each Use Case divided in their specific sections. Along with them, a sequence diagram will be added to show the way each one works.

The titles of the Use Cases are:

UC1 → Authenticate in the System

UC2 → Create New User

UC3 → Create Learning Agreement

UC4 → Evaluate Agreement

UC5 → Get Course Recommendations (AI)

UC6 → Check Status & Download Agreement

## UC1: Authenticate in the System

**Goal:** The system must verify that the user exists in the database and subsequently display their specific dashboard based on their role.

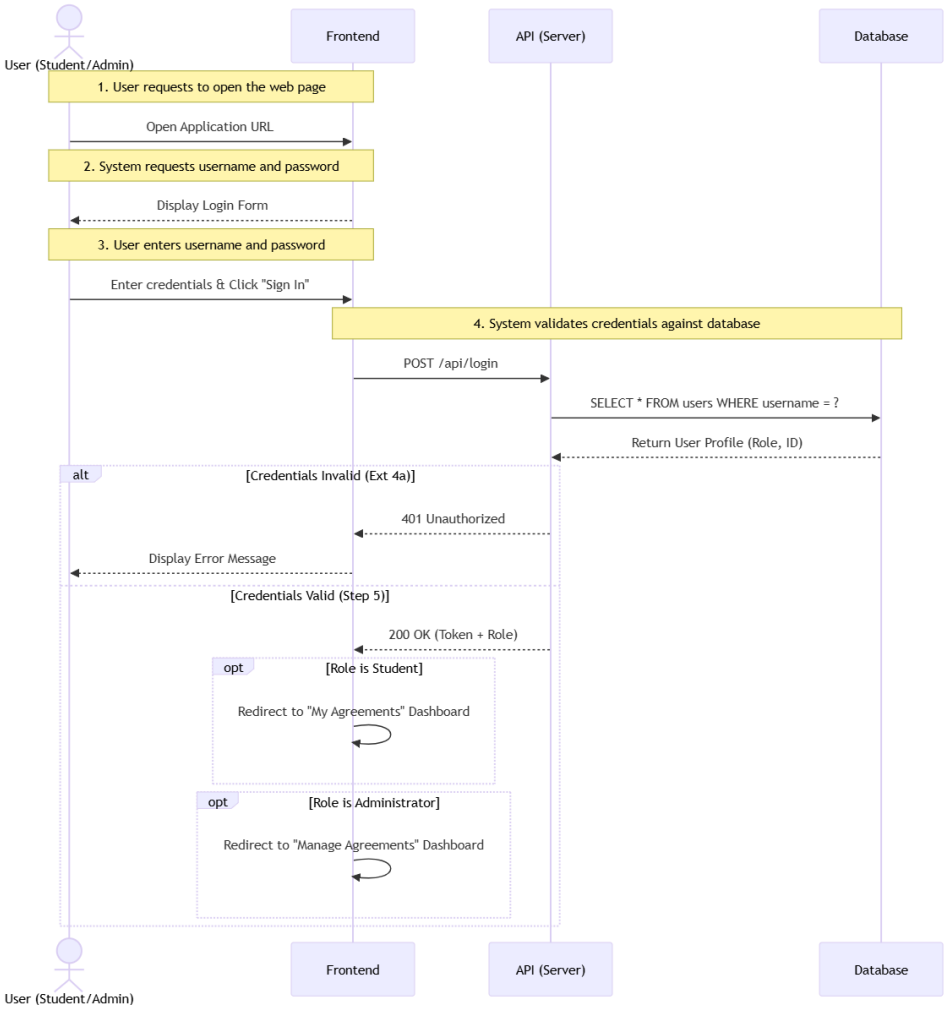
**Primary Actor:** Student, Administrator

**Main Scenario:**

1. The user requests the system to open the web page.
2. The system requests the user to enter their username and password.
3. The user enters their username and password.
4. The system validates the credentials against the database.
5. The user is logged in and shown options based on their role:
    - **Student:** Redirected to the “My Agreements” dashboard.
    - **Administrator:** Redirected to the “Manage Agreements” dashboard.

**Extensions:**

**4a.** The username or password is incorrect. The system displays an error message and returns to instruction 2.



## UC2: Create New User

**Goal:** The system must allow the Administrator to register new users (Students or other Admins) into the database with their academic details.

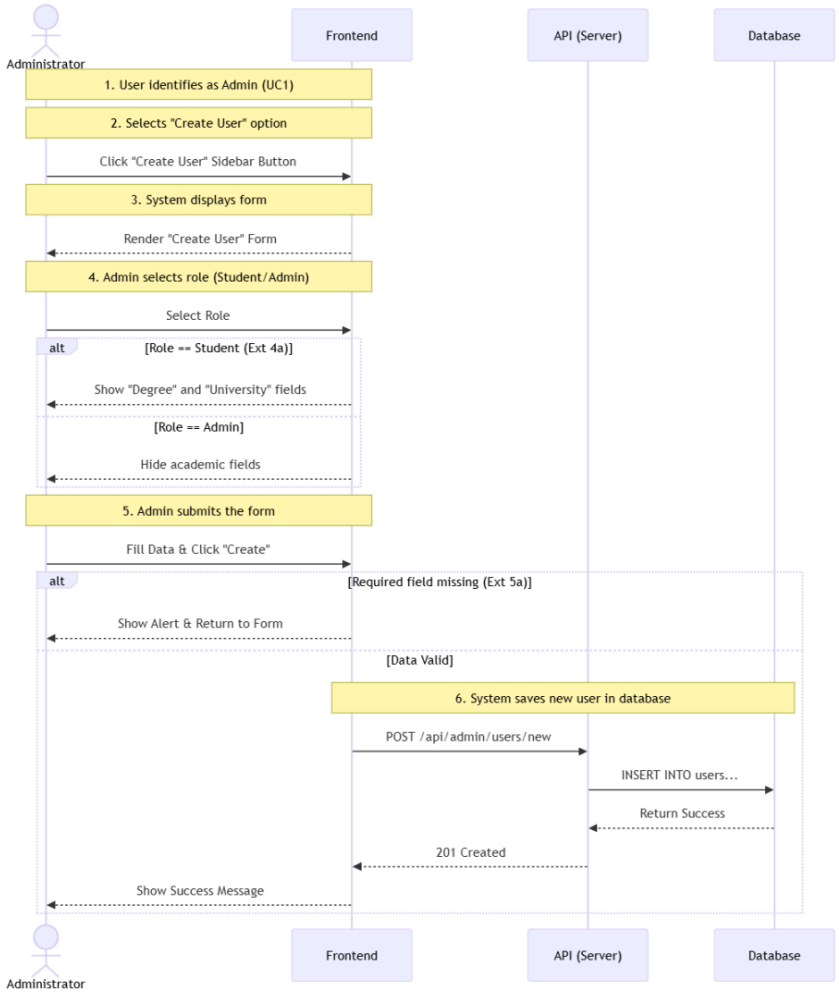
**Primary Actor:** Administrator

**Main Scenario:**

1. The user identifies as an Administrator (UC1).
2. The user selects the “Create User” option from the sidebar.
3. The system displays a form requesting personal and academic data.
4. The Administrator selects the role (Student/Admin).
5. The Administrator submits the form.
6. The system saves the new user in the database.

**Extensions:**

- **4a.** If Student is selected, the options Degree and University are also shown.
- **5a.** A required field is missing. The system shows an alert and returns to instruction 3.



## UC3: Create Learning Agreement

**Goal:** The system must allow the Student to create a new mobility agreement by selecting a destination, duration, and validating course pairs.

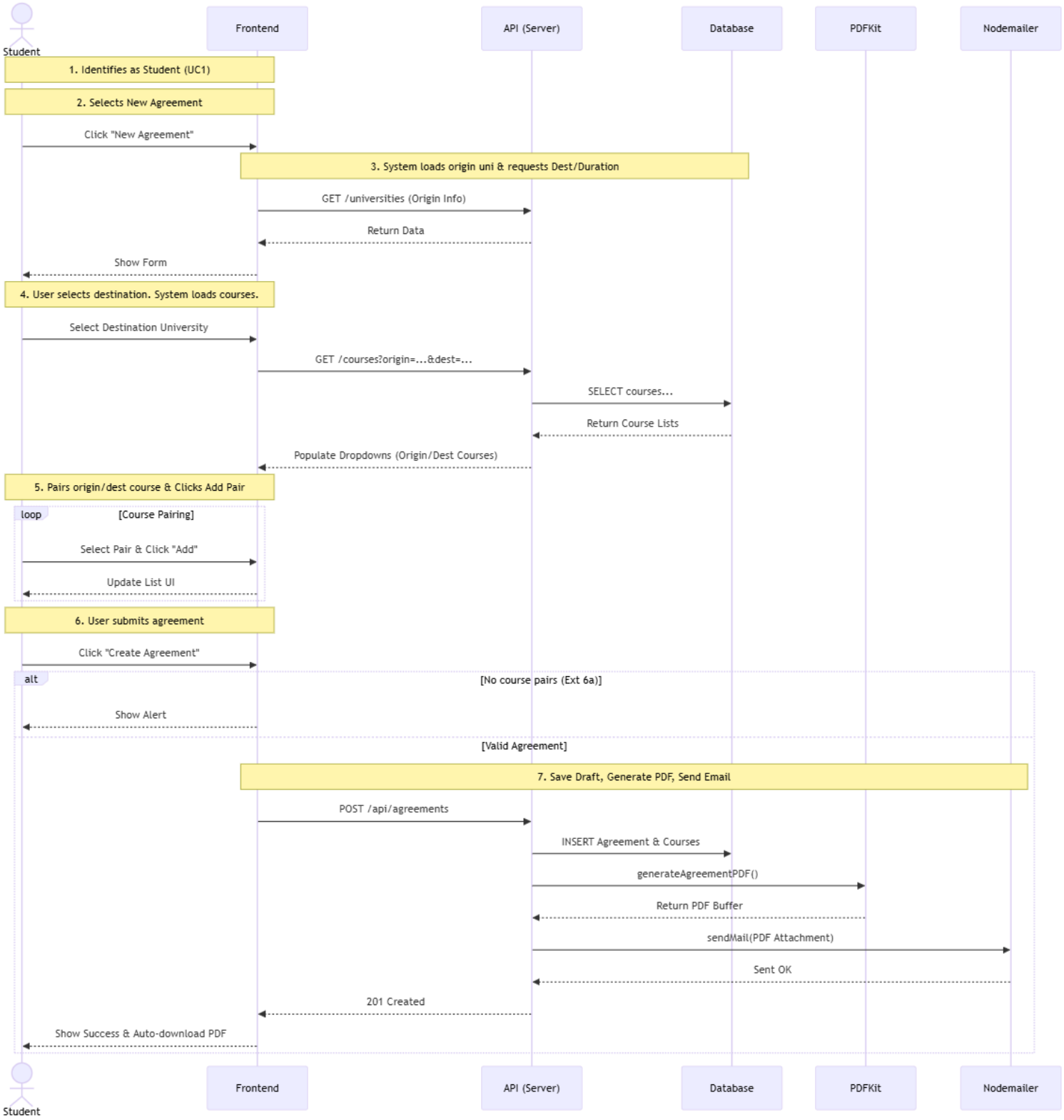
**Primary Actor:** Student

**Main Scenario:**

1. The user identifies as a Student (UC1).
2. The user selects the New Agreement option.
3. The system loads the user's origin university and requests a Destination University and Duration (Semester/Year).
4. The user selects the destination. The system loads the list of available courses for both universities.
5. The user pairs an origin course with a destination course and clicks "Add Pair".
6. The user submits the agreement.
7. The system saves the agreement as "Draft", generates a PDF, and sends a confirmation email to the user.

**Extensions:**

- **6a.** The user attempts to submit without adding any course pairs. The system shows an alert preventing submission.


Student
Frontend
API (Server)
Database
PDFKit
Nodemailer
1. Identifies as Student (UC1)
2. Selects New Agreement
Click "New Agreement"
3. System loads origin uni & requests Dest/Duration
GET /universities (Origin Info)
Return Data
Show Form
4. User selects destination. System loads courses.
Select Destination University
GET /courses?origin=...&dest=...
SELECT courses...
Return Course Lists
Populate Dropdowns (Origin/Dest Courses)
5. Pairs origin/dest course & Clicks Add Pair
loop
[Course Pairing]
Select Pair & Click "Add"
Update List UI
6. User submits agreement
Click "Create Agreement"
alt
[No course pairs (Ext 6a)]
Show Alert
[Valid Agreement]
7. Save Draft, Generate PDF, Send Email
POST /api/agreements
INSERT Agreement & Courses
generateAgreementPDF()
Return PDF Buffer
sendMail(PDF Attachment)
Sent OK
201 Created
Show Success & Auto-download PDF
Student
Frontend
API (Server)
Database
PDFKit
Nodemailer

## UC4: Evaluate Agreement

**Goal:** The system must allow the Administrator to review pending agreements and change their status (Approve/Reject).

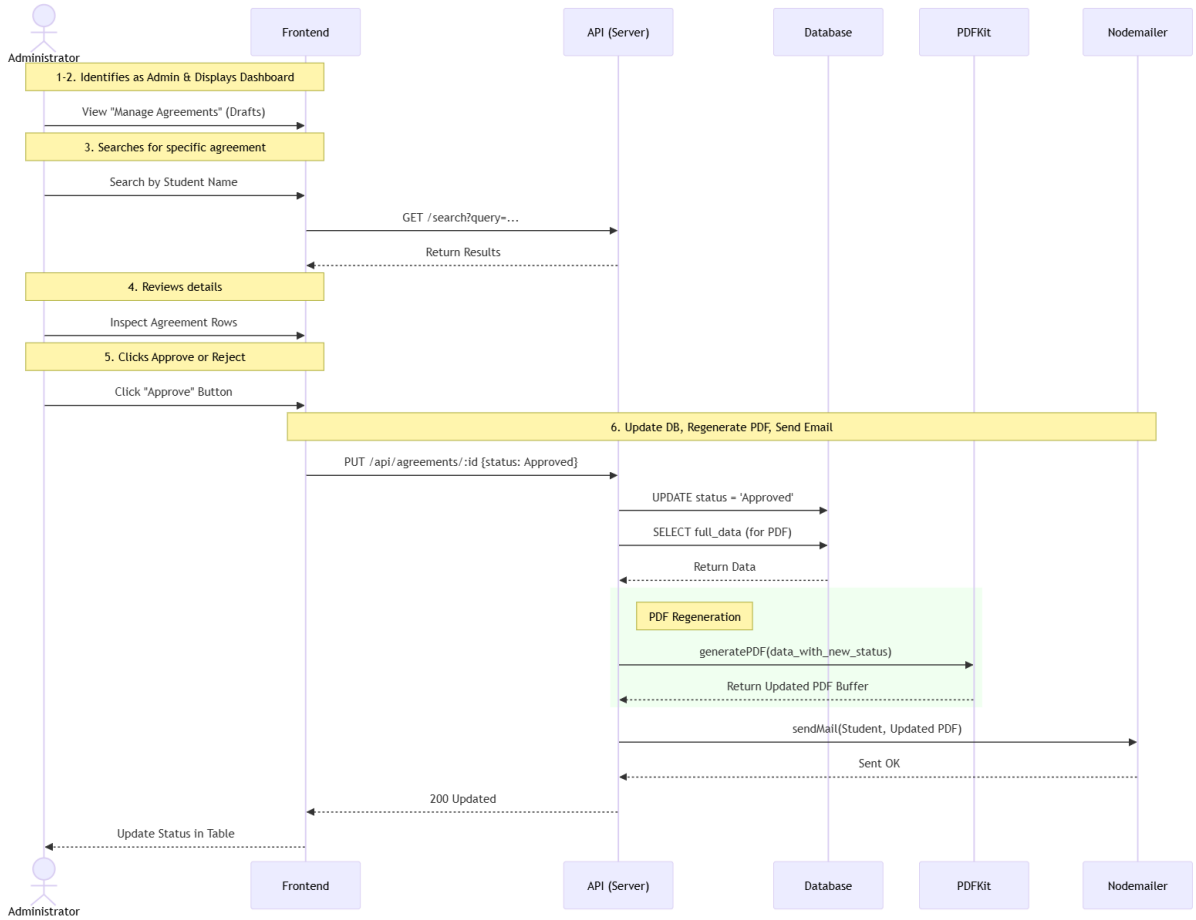
**Primary Actor:** Administrator

**Main Scenario:**

1. The user identifies as an Administrator (UC1).
2. The system displays the “Manage Agreements” dashboard with a list of “Draft” agreements by default.
3. The user searches for a specific agreement either by using the “Draft” list, or searching the students username / name.
4. The user reviews the details (Student Name, Destination, Course Pairs) of a specific agreement.
5. The user clicks the “Approve” or “Reject” button.
6. The system updates the status in the database, regenerates the PDF with the new status stamp, and sends an email notification to the Student.

**Extensions:**

- **2a.** There are no pending agreements. The system displays a “No agreements found” message.



## UC5: Get Course Recommendations (AI)

**Goal:** The system must assist the Student in finding equivalent courses by suggesting matches based on text similarity.

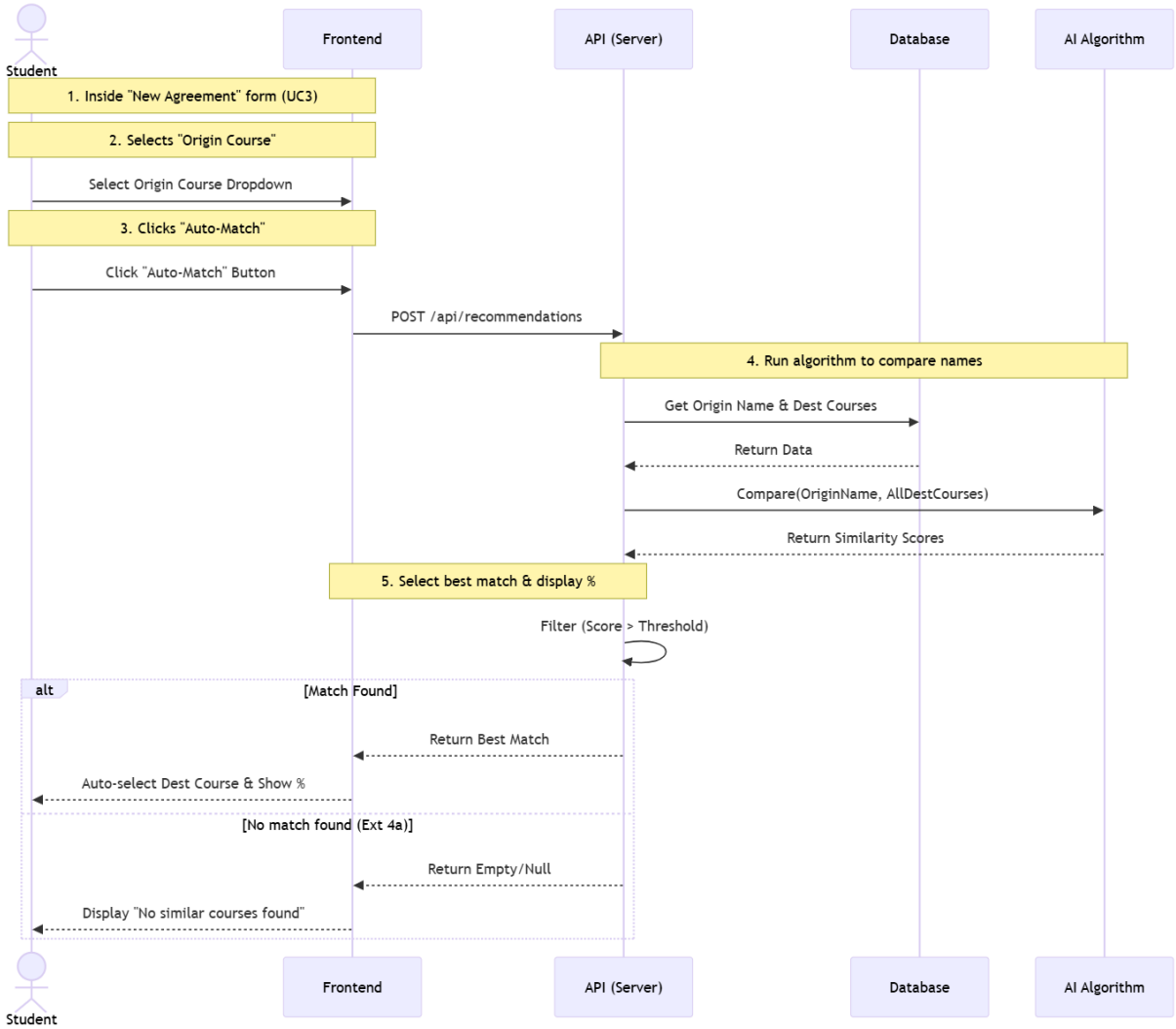
**Primary Actor:** Student

**Main Scenario:**

1. The user is inside the “New Agreement” form (UC3).
2. The user selects an “Origin Course” from the dropdown list.
3. The user clicks the “Auto-Match” button.
4. The system runs an algorithm to compare the selected course name against all available courses at the destination university.
5. The system automatically selects the best match in the “Destination Course” section and displays the similarity percentage.

**Extensions:**

- **4a.** No similar course is found (similarity score is too low). The system displays a message: “No similar courses found.”



## UC6: Check Status and Download Agreement

**Goal:** The system must allow the Student to view the current status of their applications and download the PDF documentation.

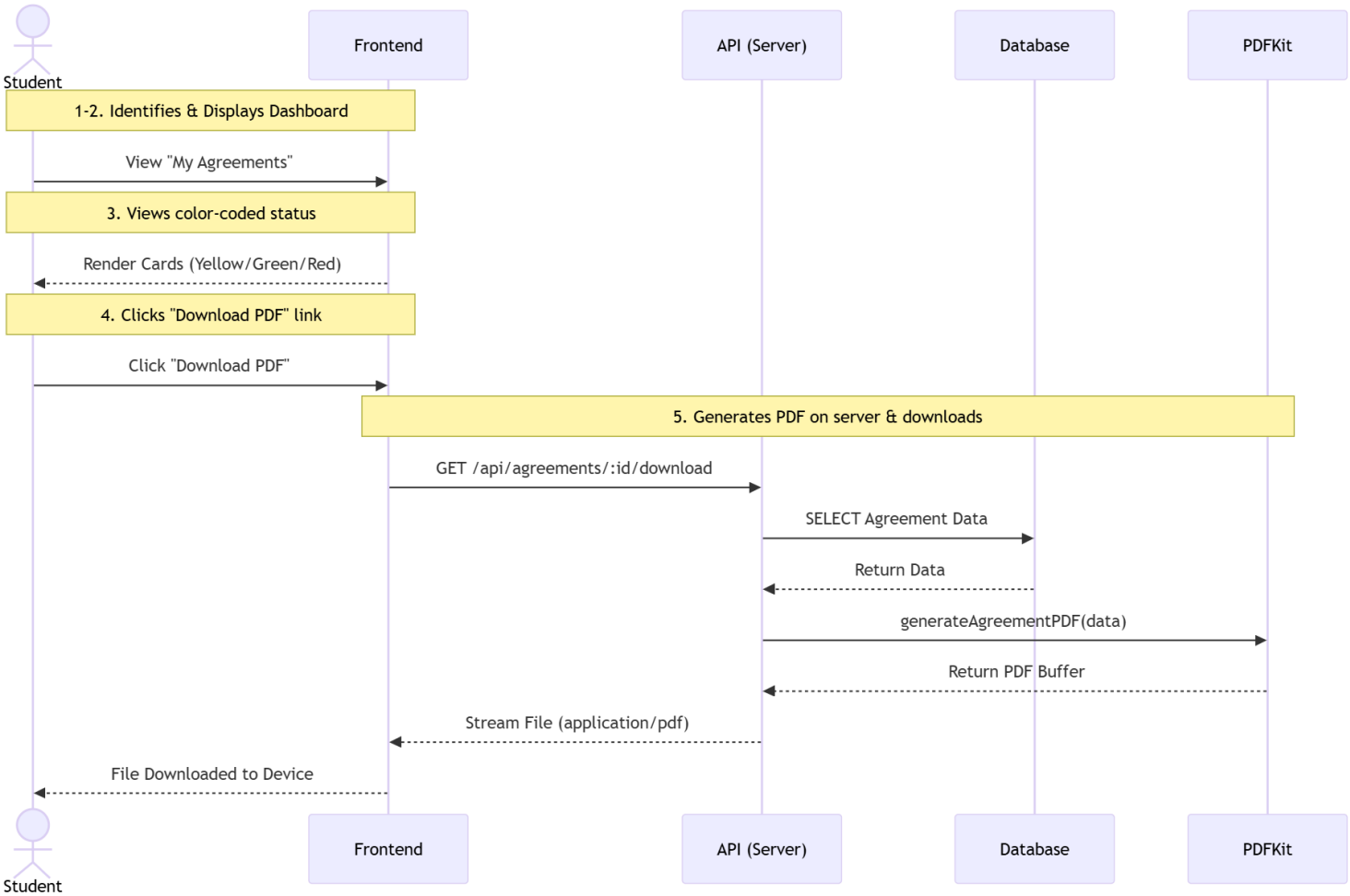
**Primary Actor:** Student

**Main Scenario:**

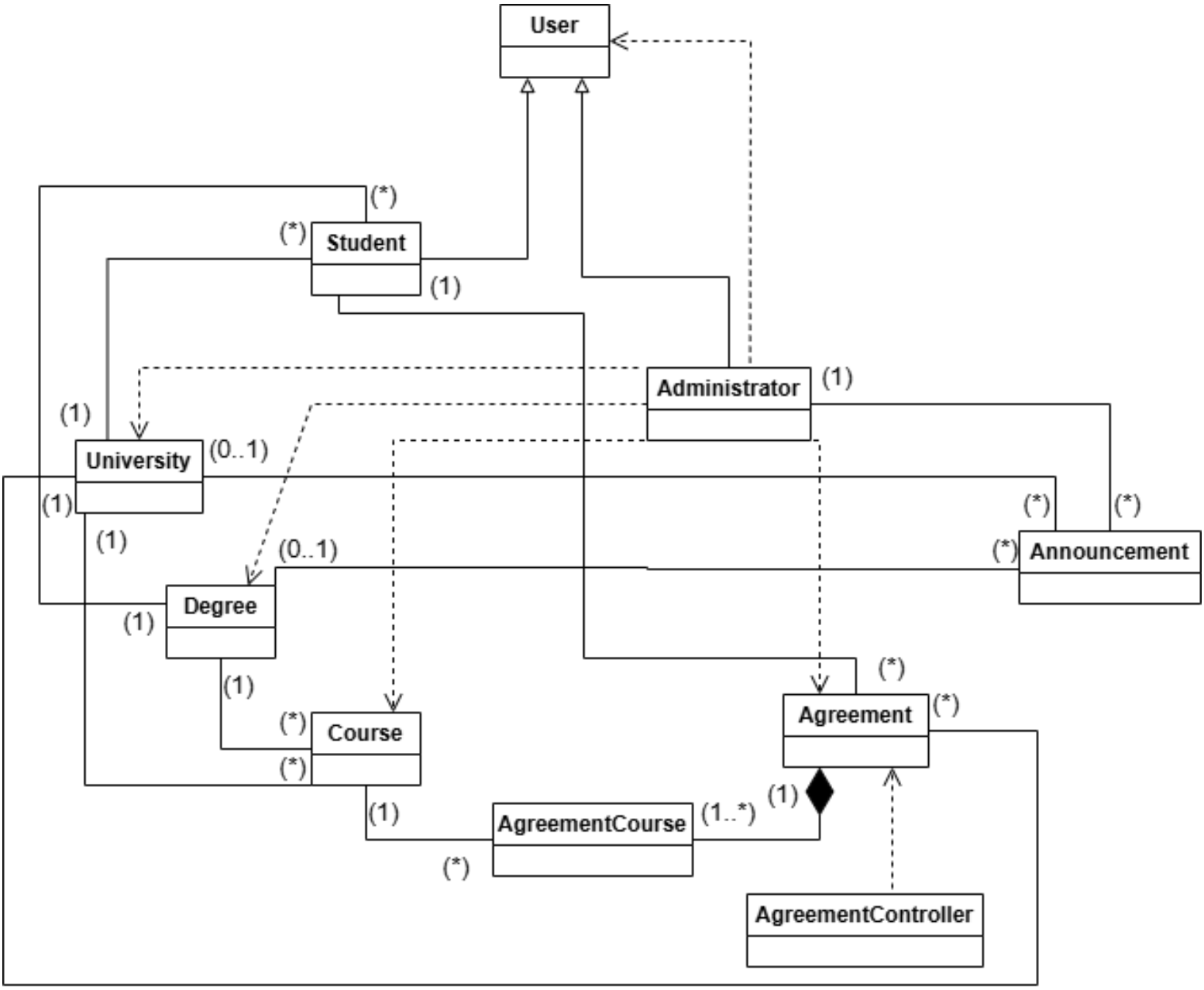
1. The user identifies as a Student (UC1).
2. The system displays the “My Agreements” dashboard with a list of created cards.
3. The user views the color-coded status (Yellow/Draft, Green/Approved, Red/Rejected).
4. The user clicks the “Download PDF” link on a specific agreement card.
5. The system generates the PDF on the server and downloads it to the user’s device.

**Extensions:**

- **2a.** The student has not created any agreements yet. The system displays a “No agreements found” message.
- **3a.** If the status has changed, the new PDF will also be downloadable through the students email.



# Class Diagram




*Class Diagram*

This section will serve as an explanation of the attributes that each of the classes showed in the diagram possess.

These were not included in the diagram itself for it to be able to fit in the report.

## 1. User

This class  stores the personal and academic data of the users within the system.

| Attribute Name | Type | Description |
|---|---|---|
| - id | int | Unique identifier of the user (PK). |
| - username | string | Unique login name for the user. |
| - password | string | Secure password for authentication. |
| - name | string | Full legal name of the user. |
| - email | string | Email address for notifications. |
| - dni | string | National Identity Document number. |
| - role | string | Role of the user (student or admin). |
| - degreeId | int | Identifier of the associated degree (FK). |
| - universityId | int | Identifier of the home university (FK). |

| Operation Name | Return Type | Description |
|---|---|---|
| + getId() | int | Gets the user identifier. |
| + setUsername(username) | void | Sets the login username. |
| + getRole() | string | Gets the user role. |
| + setDegreeId(id) | void | Links the user to a specific degree. |
| + getUniversityId() | int | Gets the ID of the user's university. |

## 2. University

This class represents the academic institutions involved in the exchange agreements.

| Attribute Name | Type | Description |
|---|---|---|
| - id | int | Unique identifier of the university. |
| - name | string | Official name of the university. |

| Operation Name | Return Type | Description |
|---|---|---|
| + getId() | int | Gets the university identifier. |
| + getName() | string | Gets the name of the university. |
| + setName(name) | void | Sets the name of the university. |

## 3. Degree

This class represents the specific academic degrees available at the universities.

| Attribute Name | Type | Description |
|---|---|---|
| - id | int | Unique identifier of the degree. |
| - name | string | Name of the degree. |

| Operation Name | Return Type | Description |
|---|---|---|
| + getId() | int | Gets the degree identifier. |
| + getName() | string | Gets the degree name. |

## 4. Course

This class stores the detailed information of an academic subject, including its credit value and temporal period.

| Attribute Name | Type | Description |
|---|---|---|
| - id | int | Unique identifier of the course. |
| - name | string | Name of the subject. |
| - credits | float | ECTS credit value of the course. |
| - period | string | Academic period. |
| - degreeId | int | Identifier of the degree this course belongs to. |
| - universityId | int | Identifier of the university offering this course. |

| Operation Name | Return Type | Description |
|---|---|---|
| + getName() | string | Gets the course name. |
| + getCredits() | float | Gets the ECTS credits. |
| + setPeriod(period) | void | Sets the academic period. |
| + getUniversityId() | int | Gets the associated university ID. |

## 5. Agreement

This class represents the mobility contract made by a student, tracking status & duration.

| Attribute Name | Type | Description |
|---|---|---|
| - id | int | Unique identifier of the agreement. |
| - userId | int | Identifier of the student owner. |
| - originUniId | int | Identifier of the home university. |
| - destUniId | int | Identifier of the destination university. |
| - duration | string | Duration of the stay. |
| - status | string | Current state (Draft, Approved, Rejected). |
| - createdAt | date | Timestamp of creation. |

| Operation Name | Return Type | Description |
|---|---|---|
| + setStatus(status) | void | Updates the agreement status (Admin only). |
| + getDuration() | string | Gets the duration of the stay. |
| + addCoursePair(pair) | void | Adds a validated course pair to the agreement. |
| + getOwner() | User | Gets the student associated with the agreement. |

## 6. AgreementCourse

This class represents the specific validation link between two courses in an agreement.

| Attribute Name | Type | Description |
|---|---|---|
| - id | int | Unique identifier of the relation. |
| - agreementId | int | Identifier of the parent agreement. |
| - originCourseId | int | Identifier of the home university course. |
| - destCourseId | int | Identifier of the destination university course. |

| Operation Name | Return Type | Description |
|---|---|---|
| + getOriginCourse() | Course | Retrieves the full origin course object. |
| + getDestCourse() | Course | Retrieves the full destination course object. |

## 7. Announcement

This class stores news and updates posted by administrators, along segmentation logic.

| Attribute Name | Type | Description |
|---|---|---|
| - id | int | Unique identifier of the announcement. |
| - title | string | Headline of the announcement. |
| - content | string | Body text of the message. |
| - imageUrl | string | URL to an attached image (optional). |
| - authorId | int | Identifier of the admin who created it. |
| - targetDegreeId | int | ID for filtering by degree (optional). |
| - targetUniId | int | ID for filtering by university (optional). |

| Operation Name | Return Type | Description |
|---|---|---|
| + isGlobal() | boolean | Returns true if no targets are set. |
| + getTargetDegree() | int | Gets the specific target audience degree. |

## 8. AgreementController (Backend Logic)

This class encapsulates the complex business logic and third-party integrations handled by the server application.

| Attribute Name | Type | Description |
|---|---|---|
| - mailer | Nodemailer | Instance of the email transport service. |
| - pdfEngine | PDFKit | Instance of the document generation engine. |

| Operation Name | Return Type | Description |
|---|---|---|
| + createAgreement(data) | json | Handles creation transaction and triggers alerts. |
| + generatePDF(data) | buffer | Complex: Renders the official PDF in memory. |
| + sendEmail(to, pdf) | void | 3rd Party: Sends transactional email via SMTP. |
| + getRecommendations() | list | Complex: Runs string-similarity algorithm for matching. |
| + calculateStats() | json | Aggregates data for admin dashboard charts. |

# Validation Matrices [Requirements vs. Use Cases]

The following matrix maps the Specific Requirements (Functional Requirements and External Interfaces) against the Use Cases. This demonstrates that every requirement listed in the IEEE 830 analysis is operationalized by at least one specific user interaction or system process.

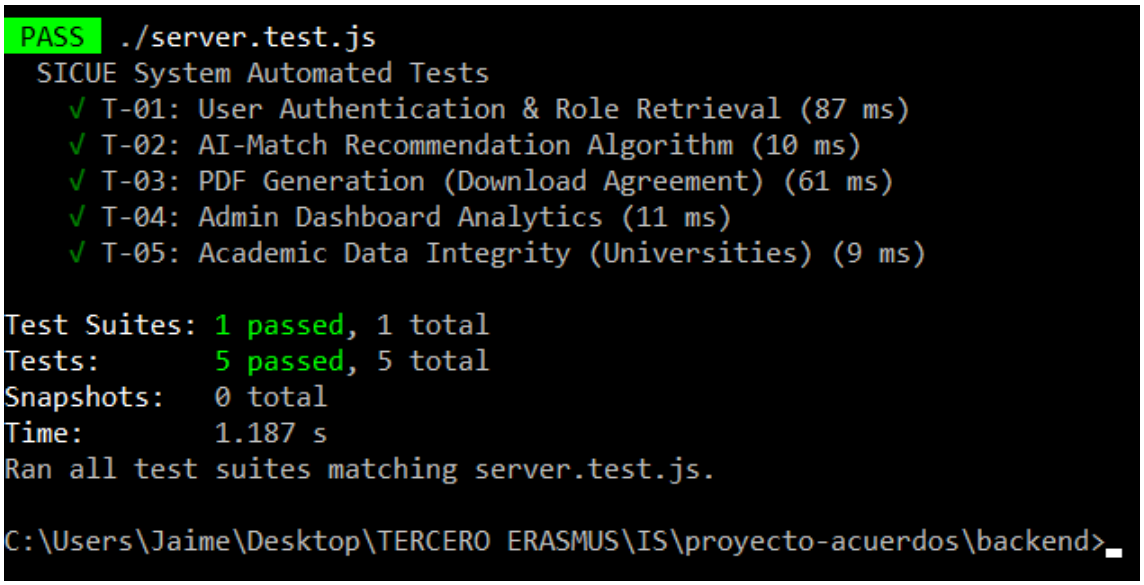
| **Requirement / Use Case** | **UC1 (Auth)** | **UC2 (User Mgmt)** | **UC3 (Create Agmt)** | **UC4 (Eval Agmt)** | **UC5 (AI Match)** | **UC6 (Status/DL)** |
|---|---|---|---|---|---|---|
| **F1** (Auth Creds) | **X** | | | | | |
| **F2** (Roles) | **X** | **X** | | | | |
| **F3** (Session) | **X** | | | | | |
| **F4** (Student Dash) | | | | | | **X** |
| **F5** (New Agmt/Pairs) | | | **X** | | | |
| **F6** (AI Recomm.) | | | **X** | | **X** | |
| **F7** (Degree Filter) | | | **X** | | **X** | |
| **F8** (Admin Dash) | | | | **X** | | |
| **F9** (Search/Filter) | | | | **X** | | |
| **F10** (Approve/Reject) | | | | **X** | | **X** |
| **F11** (CRUD Data) | | **X** | | | | |
| **F12** (Announcements) | | **X** | | | | |
| **IE1** (PDF Gen) | | | **X** | **X** | | **X** |
| **IE2** (Email SMTP) | | | **X** | **X** | | |

# Validation Matrices [Classes vs. Use Cases]

This matrix illustrates the relationship between the Static Structure (Classes/Entities) and the Dynamic Behavior (Use Cases) of the system. It identifies which data entities are required, manipulated, or queried during the execution of each use case, validating that the database schema and class design are sufficient to support the system's functionality.

| Class / Use Case | UC1 (Auth) | UC2 (User Mgmt) | UC3 (Create Agmt) | UC4 (Eval Agmt) | UC5 (AI Match) | UC6 (Status/DL) |
|---|---|---|---|---|---|---|
| **User** | **X** | **X** | **X** | **X** | **X** | **X** |
| **University** | | **X** | **X** | **X** | **X** | |
| **Degree** | | **X** | **X** | | **X** | |
| **Course** | | | **X** | | **X** | |
| **Agreement** | | | **X** | **X** | | **X** |
| **AgreementCourse** | | | **X** | **X** | | |
| **Announcement** | | **X** | | | | |
| **AgreementController** | **X** | **X** | **X** | **X** | **X** | **X** |

# Testing




| Test ID | Functionality Tested | Category | Description & Validation Logic |
|---|---|---|---|
| **T-01** | **User Authentication** | Data / CRUD | **Goal:** Verify secure login and role assignment. **Logic:** Sends a POST request with valid credentials (student/pass123). **Validation:** Checks if the API returns 200 OK, a session token, and correctly identifies the role: "student". |
| **T-02** | **AI Recommendation Algorithm** | Complex Logic | **Goal:** Validate the string-similarity heuristic engine. **Logic:** Sends an origin course ("Introduction to AI") and requests matches from a destination university. **Validation:** Checks if the algorithm returns a sorted list and if the top result has a similarity score > 0.2. |
| **T-03** | **PDF Document Generation** | 3rd Party Service | **Goal:** Ensure the PDF engine generates valid binary files. **Logic:** Requests the download endpoint for an existing agreement. **Validation:** Inspects the response headers to ensure Content-Type is application/pdf and the buffer is not empty. |
| **T-04** | **Admin Analytics Aggregation** | Complex / Data | **Goal:** Verify SQL aggregation queries for the dashboard. **Logic:** Requests the /api/admin/stats endpoint. **Validation:** Checks if the JSON response contains the specific structures for statusCounts (Draft/Approved) and topDestinations required by Chart.js. |
| **T-05** | **Database Integrity** | Data / CRUD | **Goal:** Confirm the persistence layer is active. **Logic:** Requests the list of Universities. **Validation:** Verifies that the database connection is open and the seed data (University of Madrid, etc.) is correctly retrieved. |

# LIST OF FUNCTIONALITIES

## Data/Repository/DB-oriented Functionalities

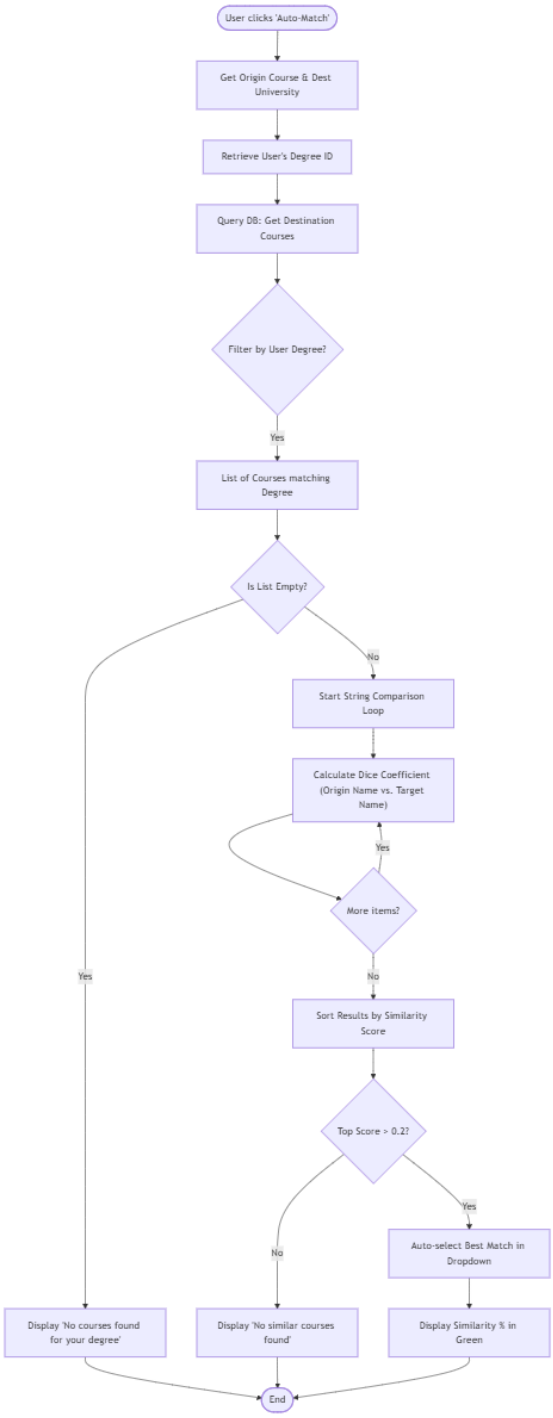
- **Hierarchical Academic Entity Management:** Full CRUD (Create, Read, Update, Delete) capabilities for the administrator to model the university structure: Universities → Degrees → Courses. It ensures referential integrity between these entities.
- **Role-Based User Management:** A system to create and manage user accounts, distinguishing between "Student" (linked to specific academic data like Degree and University) and "Administrator" (global access).
- **Agreement Lifecycle Persistence:** Storage and tracking of the mobility request status (*Draft*, *Approved*, *Rejected*) and its associated data over time.
- **Relational User Profile:** Dynamic retrieval of aggregated user data using SQL JOIN operations to combine personal, academic, and institutional information into a single view.

## Third-Party Services

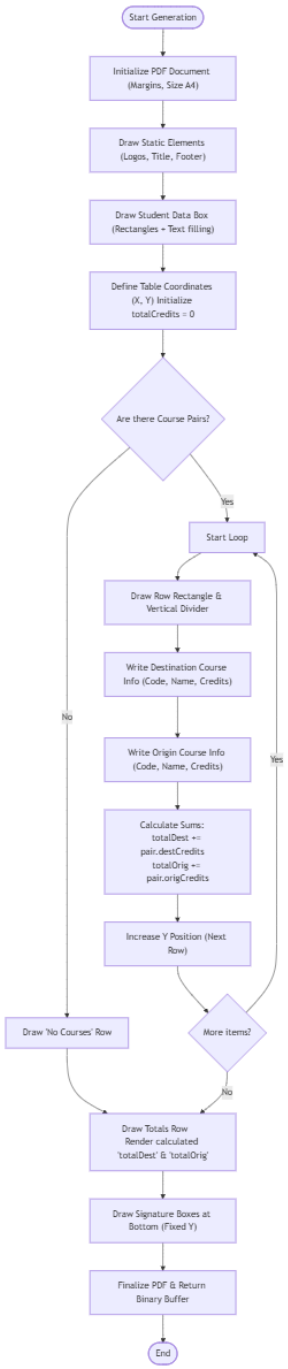
- **PDFKit (Document Generation Engine):** Used to generate binary PDF files on the server side. This service allows the creation of the official "Learning Agreement" document dynamically without relying on static templates.
- **Nodemailer (SMTP Communication):** Integration with real SMTP servers (Gmail) to handle transactional email delivery. It manages the transmission of the generated PDF buffers as email attachments directly from memory.
- **String-Similarity (Mathematical Comparison):** A specialized library that implements the Sørensen-Dice coefficient algorithm. It is used to compute the edit distance and similarity between strings for the recommendation system when choosing a course from the destination university.
- **Chart.js (Data Visualization):** A frontend library used to render the raw statistical data provided by the backend into interactive visual components (Donut and Bar charts) for the administrator dashboard.

# Complex Functionalities

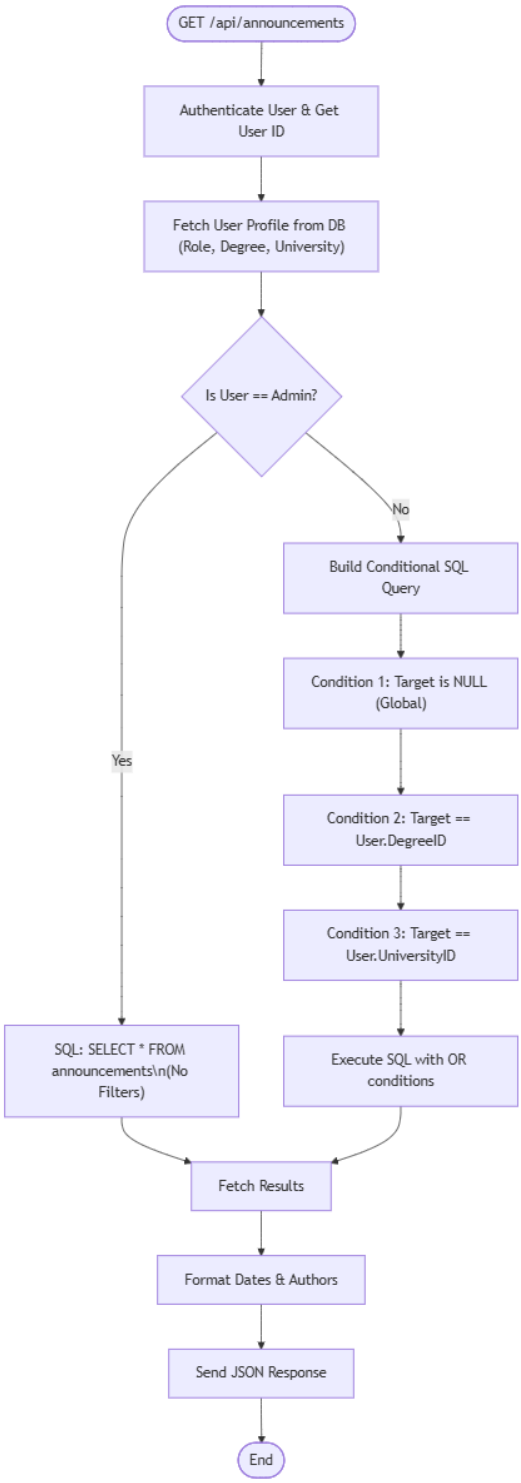
- **AI-Match Recommendation Algorithm:**
    - *Motivation:* This is not a simple database query. It is a heuristic algorithm that assists students in finding course equivalences, making the process easier and faster.
    - *Logic:* The system receives an origin course, retrieves the student's degree ID, filters the destination courses by that degree, runs the string similarity algorithm against the dataset, applies a confidence threshold (score > 0.2), and ranks the results to suggest the "Best Match" automatically.



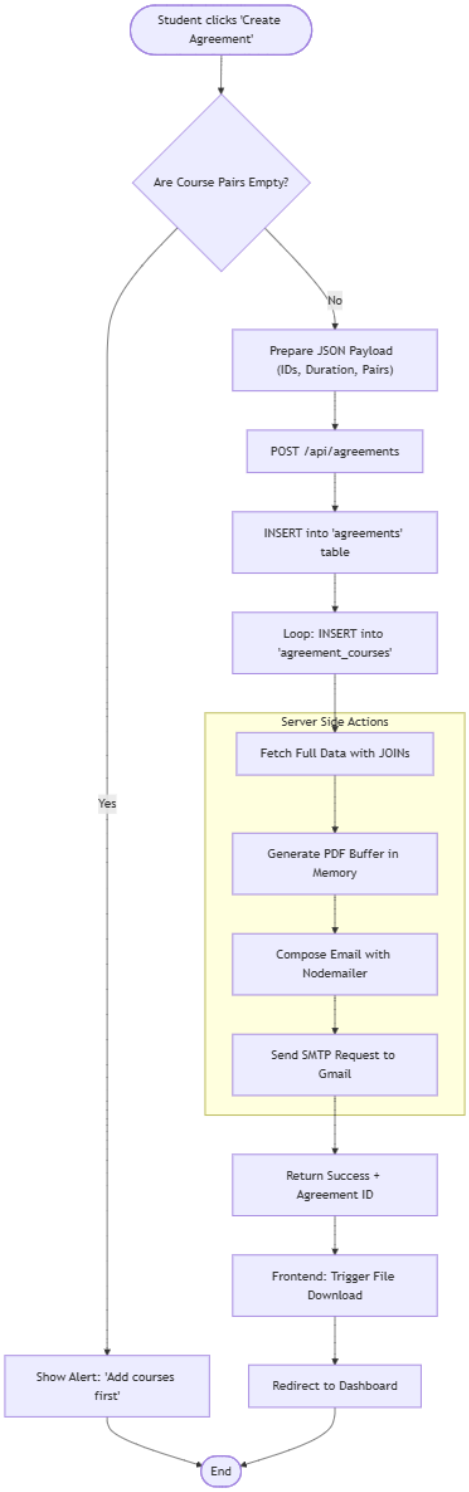
- **Dynamic PDF Rendering Engine:**
    - *Motivation:* Generating a professional document requires complex layout calculations that go beyond simple data dumping. This requires changing layout according to the number of courses, calculating the number of credits, and more information.
    - *Logic:* The backend implements a drawing algorithm that manually calculates (x, y) coordinates to draw grids and tables. It iterates over complex data structures (course pairs) and performs arithmetic aggregation in runtime (summing ECTS credits for origin and destination separately) to populate the document footer.



- **Context-Aware Announcement Feed:**
    - *Motivation:* The system needs to deliver personalized content based on user attributes rather than a static feed.
    - *Logic:* A filtering algorithm in the backend evaluates three levels of visibility (Global, University-specific, Degree-specific). It constructs a dynamic conditional SQL query (WHERE ... OR ...) based on the requester's profile to ensure students only see relevant news.

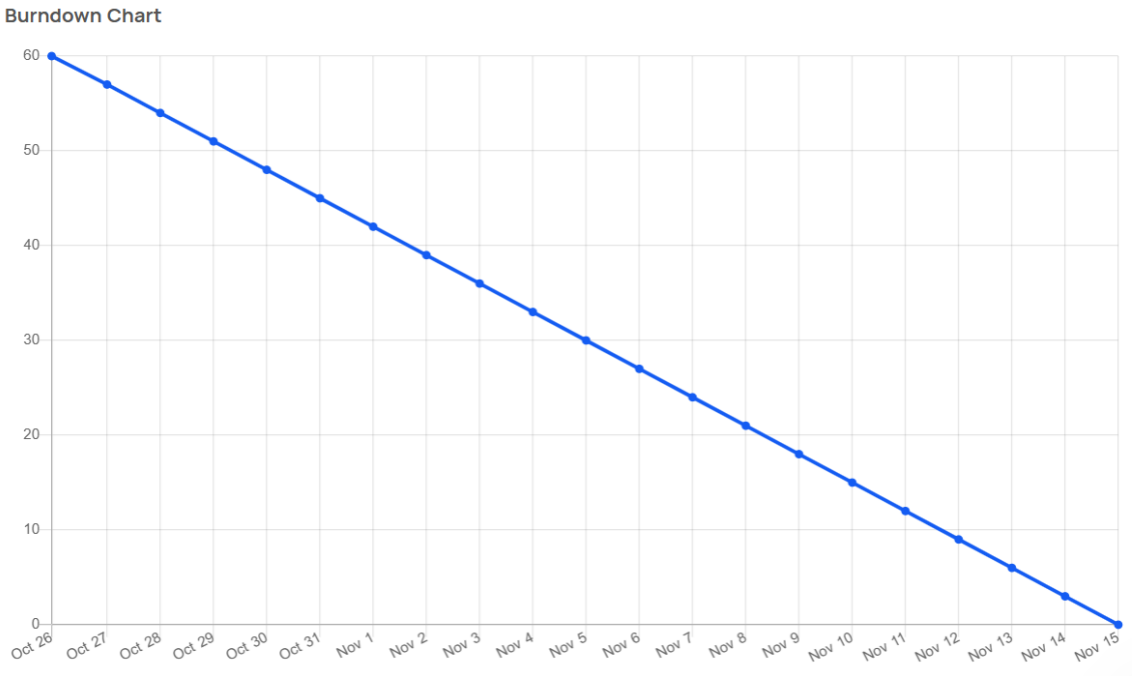


- **Agreement Validation Workflow:**
    - *Motivation:* The creation of an agreement involves synchronizing multiple asynchronous processes.
    - *Logic:* An orchestrated process that handles dependent data loading (Destination University selection triggers Course loading), validates the integrity of the course pairs, persists the transaction in the database, and triggers asynchronous events (PDF generation and Email notification) upon success.



# SCRUM METHODOLOGY

## Sprint Burndown Analysis




This is a representation of how the project would look if each sprint card from Trello had 4 points, and everything was done in time. This is a perfect, and optimal Burndown Chart for this project. In other situations, each Trello card would have a different amount of points depending on their importance, but this is a representation.

# Sprint 1: Core Architecture & Data Foundation


The image shows a screenshot with a representation in Trello of how the sprint would work, dividing it in: To do | Work in Progress | Quality Check | Tasks that have been completed

Each task has a number of points, we will suppose that each of them has 4 points, for easier representation in future graphics.

**Objective:**

The primary goal of the first increment was to establish a robust system architecture and ensure data integrity for the people involved in the mobility process.

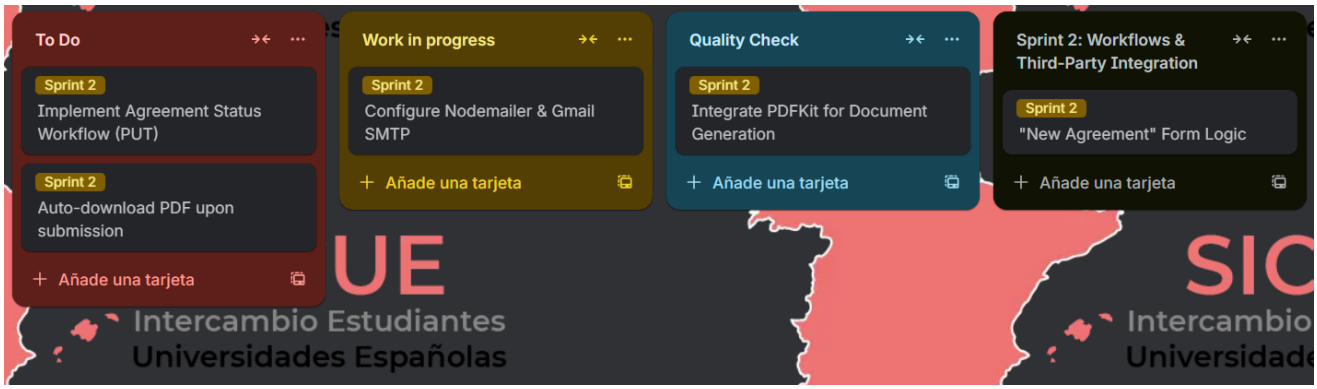
**Development Narrative:**

It began by initializing the Node.js environment and configuring the SQLite relational database schema. The priority was to implement a secure authentication system and Role-Based Access Control (RBAC). The distinction between "Student" and "Administrator" roles at the database level was created, ensuring that API endpoints could differentiate permissions. Once authentication was stable, the full CRUD (Create, Read, Update, Delete) REST API endpoints were developed for the fundamental academic entities: Universities, Degrees, and Courses.

**Key Functionalities Delivered:**

- **Category 1 (Data):** Relational Database Schema implementation.
- **Category 1 (Data):** User Authentication and Profile management.
- **Category 1 (Data):** Administrative tools for creating Universities and Degrees.

# Sprint 2: Workflows & Third-Party Integration




The image shows a screenshot with a representation in Trello of how the sprint would work, dividing it in: To do | Work in Progress | Quality Check | Tasks that have been completed

Each task has a number of points, we will suppose that each of them has 4 points, for easier representation in future graphics.

**Objective:**

> This sprint focused on the core business logic: allowing students to create mobility agreements and generating the required official documentation using external services.

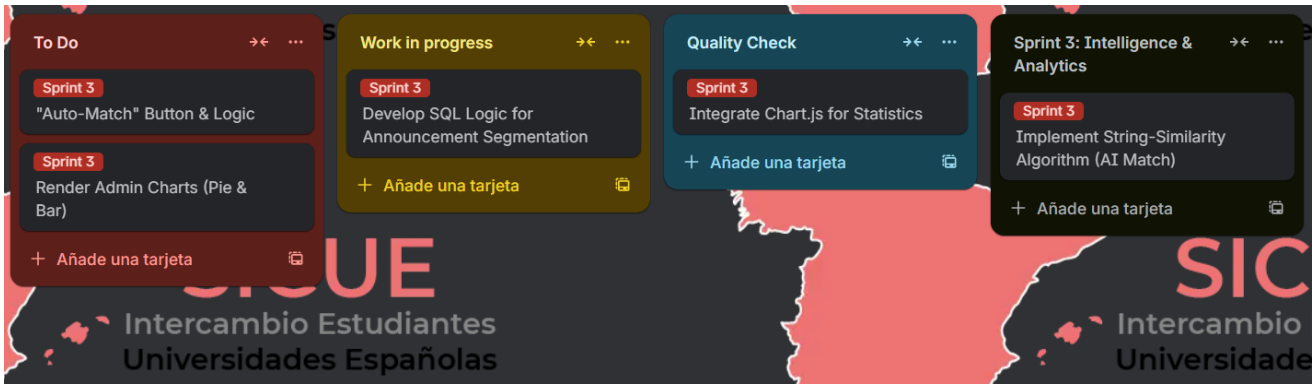
**Development Narrative:**

> With the core data in place, I implemented the complex Agreement Creation Workflow. This involved designing a frontend logic that dynamically loads courses based on the selected destination. To fulfill the documentation requirements, I integrated PDFKit (Category 2). Instead of using a static template, I developed a rendering engine that programmatically draws the "Learning Agreement," calculating credit totals in real-time. Simultaneously, I integrated Nodemailer to handle transactional emails, ensuring students receive immediate confirmation and a copy of their PDF upon submission or status change.

**Key Functionalities Delivered:**

- **Category 3 (Complex):** Agreement Creation Workflow (Course Pairing logic).
- **Category 2 (3rd Party):** Dynamic PDF Generation engine (PDFKit).
- **Category 2 (3rd Party):** SMTP Email Notification system (Nodemailer).
- **Category 1 (Data):** Agreement state persistence (Draft $\rightarrow$ Approved/Rejected).

# Sprint 3: Intelligence & Analytics




The image shows a screenshot with a representation in Trello of how the sprint would work, dividing it in: To do | Work in Progress | Quality Check | Tasks that have been completed

Each task has a number of points, we will suppose that each of them has 4 points, for easier representation in future graphics.

**Objective:**

The final increment aimed to enhance user experience through intelligent assistance and provide decision-support tools for administrators.

**Development Narrative:**

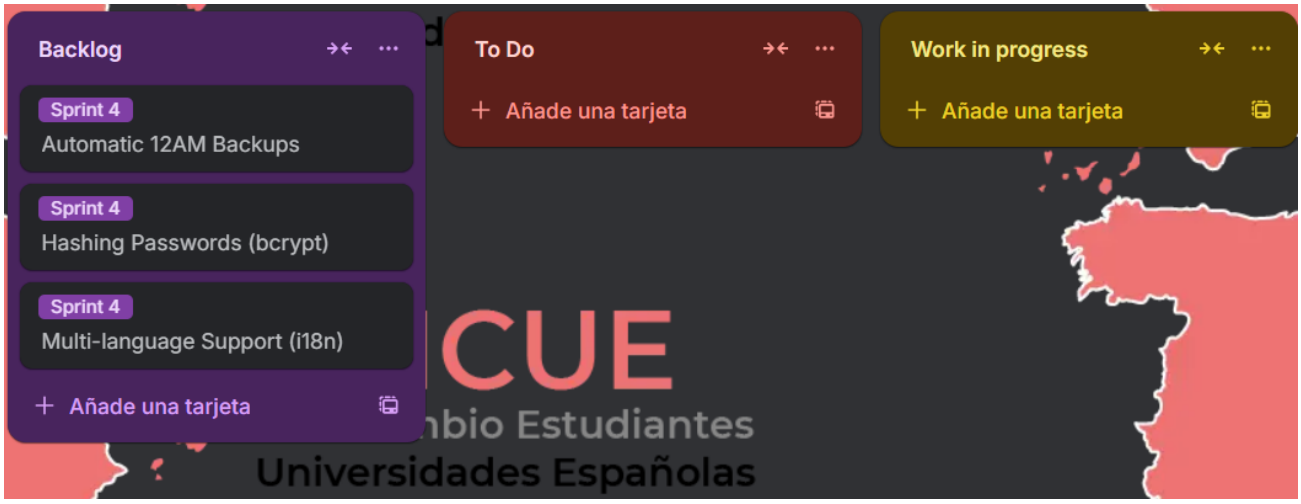
To reduce errors in course selection, I implemented an AI-Match Algorithm using the *Sørensen-Dice coefficient* (via the string-similarity library). This feature analyses the semantic similarity between origin and destination courses to suggest validations automatically. For the administration side, I developed an analytical dashboard. I integrated Chart.js on the frontend and wrote complex SQL aggregation queries on the backend to visualize KPIs such as "Top Destinations" and "Agreement Status Distribution." Finally, I implemented a Context-Aware Announcement system that filters news based on the user's specific degree or university.

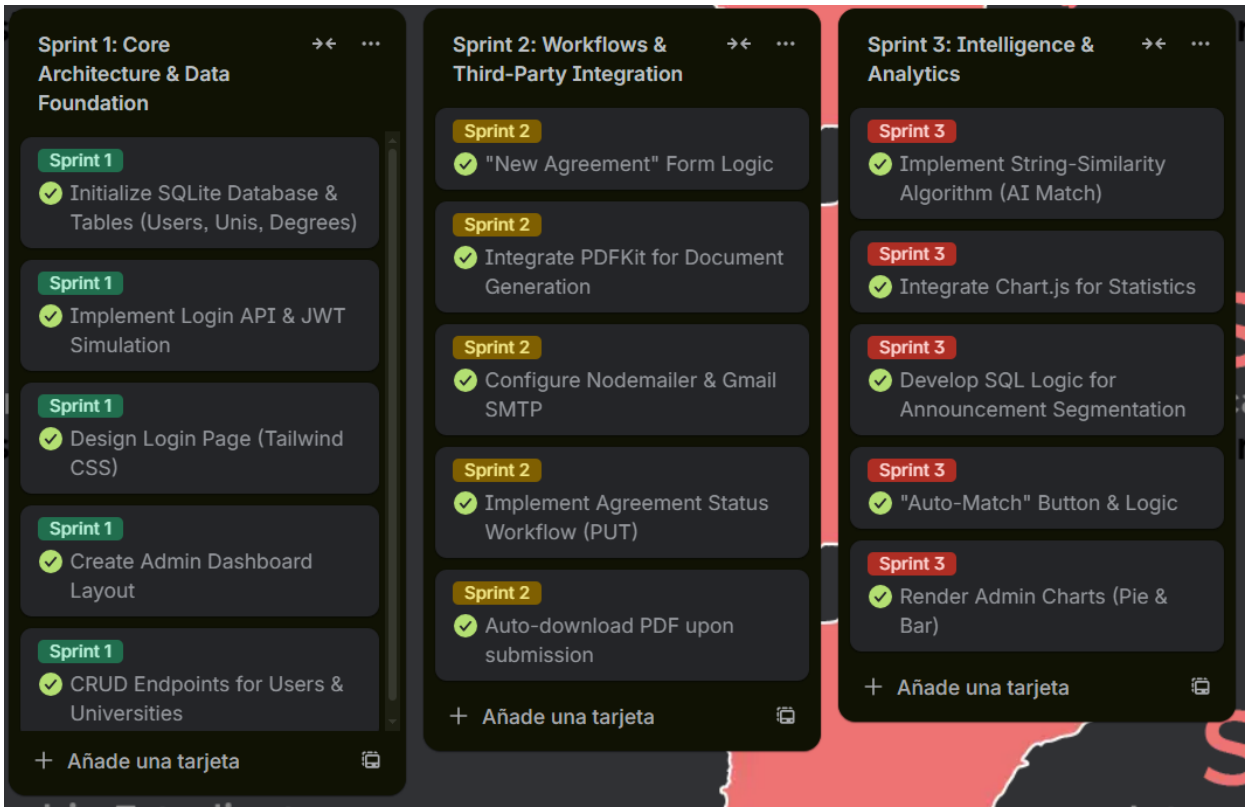
**Key Functionalities Delivered:**

- **Category 3 (Complex):** AI-based Course Recommendation Algorithm.
- **Category 3 (Complex):** Context-Aware News Feed (SQL Filtering logic).
- **Category 2 (3rd Party):** Data Visualization (Chart.js).
- **Category 2 (3rd Party):** String Similarity Mathematical Library.

# Sprint 4: Future Updates

This sprint does not exist yet, but in case more functionalities need to be implemented, they would be added to this Sprint.




These screenshots show all the completed sprint tasks, and also the tasks that would be done in a possible fourth sprint.

# Conclusion

## Final Thoughts

The development of the University Mobility Agreements Portal has been successfully completed, fulfilling the functional and non-functional requirements defined in the IEEE 830 analysis.

By integrating Data-Oriented functionalities (CRUD for users and academic entities), Third-Party Services (PDFKit, Nodemailer, Chart.js), and Complex Logic (AI-Match Algorithm, Context-Aware segmentation), the project demonstrates a comprehensive application of Software Engineering principles.

## Future Improvements

While the current system is fully functional for a demonstration environment, a transition to a production environment would require addressing specific security and scalability aspects:

1. **Security:** Implementation of Bcrypt for password hashing and JSON Web Tokens (JWT) for stateless API authentication to replace the current session simulation.
2. **Database Migration:** Migration from SQLite to a more robust RDBMS like PostgreSQL or MySQL to handle high concurrency in a real university setting.
3. **Internationalization (i18n):** Implementation of a multi-language support system to accommodate Erasmus students from different countries.
4. **Daily Backups:** Every day at 12:00AM the server would do a backup and save in GitHub the information in the database, stored in a file with the date as its name.