Final Documentation for Coursepedia

Yuxuan Xia, Yongyu Liu, Haochen Yang

May 13, 2024

Contents

1 Introduction

1.1 Project Description

Our Educational Course Management Software offers a suite of tools designed to enhance the academic journey for students. It features secure login capabilities for students to access their course history, enabling them to review past academic achievements and plan future courses. Additionally, the platform includes a sophisticated course browsing function that allows students to apply various filters such as time, location, major, and semester availability, ensuring they find courses that match their academic goals. Furthermore, the software provides an automatic course plan recommendation feature, which generates personalized academic plans based on the student's major and completed courses, considering prerequisites and graduation requirements. This feature-rich platform is crafted to support students in navigating their educational paths efficiently, making informed decisions, and achieving academic success.

2 Goals of the Documentation

The primary goal of this documentation is to provide a comprehensive guide that outlines the functionality and usability of the Educational Course Management Software. This document is intended to serve multiple purposes:

1. **Facilitate Understanding:** To clearly explain the features and processes of the software to all stakeholders, including developers, administrators, and end-users, ensuring that all parties have a thorough understanding of the system's capabilities and interface.

2. **Support Training:** To serve as a foundational training resource for new users and administrators, enabling them to efficiently navigate and utilize the software to its full potential. This includes detailed explanations of how to log in, view course histories, browse available courses with filters, and receive personalized course recommendations.

3. **Promote Efficient Use:** To provide step-by-step instructions and troubleshooting advice, ensuring users can operate the software effectively and resolve any issues that may arise during its use.

4. **Guide Future Enhancements:** To document the current state of the software for future reference, which will be invaluable for ongoing maintenance and enhancement. This documentation will help developers understand existing features and functionalities to better innovate and improve the software.

5. **Ensure Consistency:** To maintain consistency in the use and application of the software across different user groups and educational institutions. This documentation aims to standardize operations to avoid discrepancies and ensure a uniform user experience.

Through this documentation, we aim to enhance user engagement, reduce learning curves, and foster an environment that maximizes educational efficiency and effectiveness.

3 Deliverables

3.1 Project Overview

The Coursepedia is a comprehensive educational platform designed to streamline the management and accessibility of academic resources for students and faculty. With a focus on ease of use and integration, the Coursepedia encompasses various functionalities including academic planning, course management, and user authentication, all accessible via a unified user interface.

3.2 Development Process

We used the agile and eXtreme Programming methods for the development for two reasons: 1. All of our team members are capable of revisiting a methodology to an unprecedented new situation 2. This is a rather small project and we could not foresee the obstacles, planning everything beforehand is unrealistic.

Below is a more detailed description of our iterative process.

3.2.1 Iterative Development

Iterative development has been a crucial approach in our project, allowing us to progressively refine and enhance our application.

3.2.2 Iteration 1: Initial Setup and Data Collection (March 20)

- Focused on overcoming technical challenges such as bypassing reCAPTCHA.
- Established foundational functionalities including login and signup processes.
- Encountered significant data collection challenges impacting testing of recommendation functions.

3.2.3 Iteration 3: Refinement and Scope Definition (April 10)

- Limited the recommendation system scope to CS and DS courses to manage complexity.
- Continued development of the recommendation function and adjustments in the database structure.

3.2.4 Refactoring

Refactoring was applied to simplify the project structure and enhance code quality.

- Simplified the database by reducing the feature set to essential elements for CS and DS majors.
- Standardized data handling processes to accommodate inconsistencies in course data.

3.2.5 Collaborative Development

Our project benefited significantly from collaborative efforts, which were structured around regular meetings and clear role distributions.

- Each team member specialized in different aspects of the project, enhancing efficiency and focus.
- Regular meetings provided a platform for problem-solving and decision-making.

3.3 Requirements and Specifications

This section details the user stories and use cases implemented in our project, focusing on the functional requirements that were retained throughout the development process.

3.3.1 Use Cases

- **Use Case Name: Student Login and View Past Courses**
  - **Actor:** Student
  - **Preconditions:** The student has an account with the school.
  - **Basic Flow:**
    1. Student navigates to the web application.
    2. Student enters login credentials.
    3. System validates credentials and logs the student in.
    4. System retrieves student's past course information from the official school database.
    5. System analyzes and displays past taken courses on the website for the student to browse.
  - **Alternate Flows:** If login credentials are incorrect, the system displays an error message and prompts for re-entry.
  - **Postconditions:** The student is logged in and can view their past courses.
- **Use Case Name: Browse Courses with Filters**
  - **Actor:** Student
  - **Preconditions:** The student is logged in.
  - **Basic Flow:**
    1. Student selects to browse courses.
    2. Student applies filters for time, location, major, and available semester.
    3. System retrieves and displays courses matching the selected filters.
  - **Alternate Flows:** No courses match the selected filters; the system displays a message indicating no results and suggests modifying the filters.
  - **Postconditions:** The student views courses matching their selected filters.
- **Use Case Name: Generate Recommended Course Plan**

- **Actor:** Student
- **Preconditions:** The student is logged in.
- **Basic Flow:**
  1. Student chooses their major from a list.
  2. System retrieves the student's taken courses.
  3. System generates a recommended course plan based on the student's major and taken courses, considering prerequisites and graduation requirements.
  4. System displays the recommended course plan to the student.
- **Alternate Flows:** If the system cannot generate a plan (e.g., due to missing prerequisites), it alerts the student and suggests alternative actions.
- **Postconditions:** The student receives a personalized course plan recommendation.

3.4 Architecture and Design

Include multiple UML diagrams to show the important parts of your system (you must have UML diagrams). Describe in a top-down manner the architecture of your system. Include enough details about the design of your system such that anyone who refers to your documentation can understand the major components of your system and how they are related. Describe how the choice of the framework influence the design of your system.

3.4.1 Backend (Server-Side)

API Layer (Controllers)

- AuthenticationController: Manages user login and session states.
- CourseController: Retrieves and filters course data.
- PlanController: Handles course plan generation.

Service Layer

- AuthService: Provides user authentication services.
- CourseService: Manages course information logic.
- PlanService: Develops course planning algorithms.

Data Access Layer (Models)

- User: Represents the student's user model.
- Course: Contains course details.
- Plan: Links course plans with student profiles.

Utility Classes

- DatabaseConnection: Ensures database interaction.
- Logger: Facilitates application logging.

Configuration

- SecurityConfig: Sets up application security protocols.
- ApplicationConfig: Manages general application settings.

3.4.2 Frontend (Client-Side)

Views

- LoginView: The student login interface.
- DashboardView: Main control panel for course browsing.
- CourseListView: Lists courses based on selected filters.
- PlanView: Displays the student's course plan.

ViewModels

- UserViewModel: Connects user data to the UI.
- CourseViewModel: Manages course-related UI data.
- PlanViewModel: Updates and displays course plans.

Utilities

- API Client: Handles requests to backend services.
- Validator: Validates client-side user input.

Middleware

- AuthenticationMiddleware: Ensures user sessions are valid.
- ErrorHandlingMiddleware: Manages API response errors.

3.4.3 Diagrams

Here is the UML Diagram which illustrates the structure of the program.

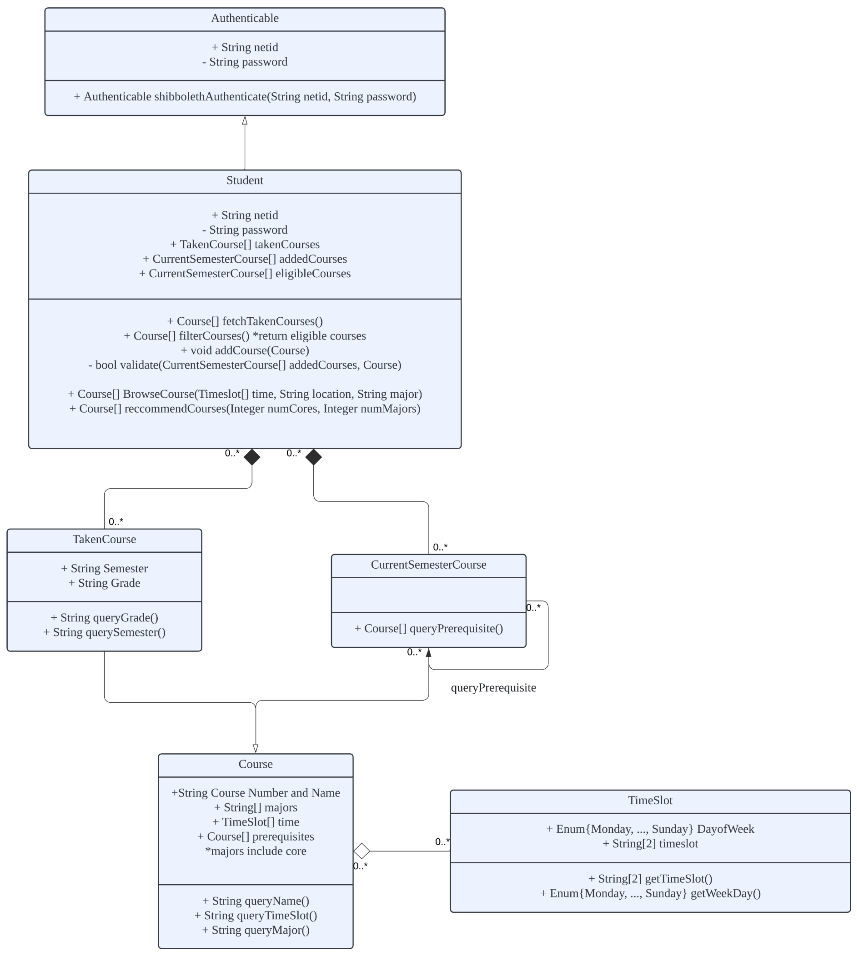
Authenticable
+ String netid
- String password
+ Authenticable shibbolethAuthenticate(String netid, String password)
Student
+ String netid
- String password
+ TakenCourse[] takenCourses
+ CurrentSemesterCourse[] addedCourses
+ CurrentSemesterCourse[] eligibleCourses
+ Course[] fetchTakenCourses()
+ Course[] filterCourses() *return eligible courses
+ void addCourse(Course)
- bool validate(CurrentSemesterCourse[] addedCourses, Course)
+ Course[] BrowseCourse(Timeslot[] time, String location, String major)
+ Course[] reccommendCourses(Integer numCores, Integer numMajors)
0..*
0..*
0..*
0..*
TakenCourse
+ String Semester
+ String Grade
+ String queryGrade()
+ String querySemester()
CurrentSemesterCourse
+ Course[] queryPrerequisite()
0..*
0..*
queryPrerequisite
Course
+String Course Number and Name
+ String[] majors
+ TimeSlot[] time
+ Course[] prerequisites
*majors include core
+ String queryName()
+ String queryTimeSlot()
+ String queryMajor()
0..*
0..*
TimeSlot
+ Enum{Monday, ..., Sunday} DayofWeek
+ String[2] timeslot
+ String[2] getTimeSlot()
+ Enum{Monday, ..., Sunday} getWeekDay()

Figure 1: Class Diagram

This is the entity relationship diagram.

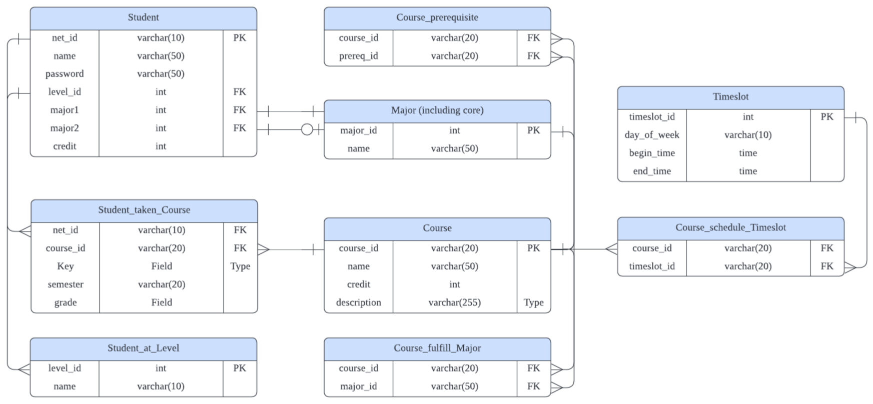

Figure 2: Entity Relationship Diagram

4 Testing

4.1 Parse Course History Function Testing

The `parse_course_history()` function is a crucial component of our system, responsible for parsing the HTML source code to extract important student course data. This function is tested using white-box testing strategies to ensure robustness and functionality across various scenarios.

4.1.1 Test Cases for Parse Course History

The function is located in `SEproject/backend/courses/parse_course_history.py`. The testing involves two primary strategies:

- Testing the semester loop with different numbers of semesters present in the HTML.
- Testing the parsing branches to ensure that partial data can still be parsed correctly.

Here are the specific cases:

1. **Case 1:** Code contains 1 semester, full information.
2. **Case 2:** Code contains 2 semesters, full information.
3. **Case 3:** Code contains 3 semesters, full information.
4. **Case 4:** Code contains 1 semester, missing credit for one course.
5. **Case 5:** Code contains 1 semester, missing grade for one course.
6. **Case 6:** Code contains 1 semester, missing course name for one course.

To run these tests, execute the following command: `python -m backend.test.parse_test`.

4.2 Course API View Testing

The `CourseAPIViewTestCase` class tests the API functionality responsible for processing POST requests containing HTML content of student courses.

4.2.1 Detailed Test Cases for Course API View

- `test_successful_student_and_course_creation()`: Tests successful parsing of HTML and creation of student and course instances. This test ensures that valid HTML content results in correct database entries and an HTTP 200 OK response.
- `test_student_serializer_with_invalid_data()`: Focuses on the StudentSerializer to verify its capability to handle and report invalid data effectively. This test is crucial for maintaining data integrity.
- `test_unauthorized_access()`: Confirms that the CourseAPIView restricts access for unauthenticated requests, returning an HTTP 401 Unauthorized status code. This test checks the enforcement of authentication requirements.

These tests collectively ensure that the system accurately handles both valid inputs and error conditions, maintaining system integrity and user data security.

5 Conclusion

This project documentation for the Educational Course Management Software, titled "Coursepedia," encapsulates a detailed exploration of the development and operational aspects of the system. Throughout the development process, the project has successfully adhered to its primary goals, providing a robust platform that significantly enhances the educational experience for students by simplifying course management, planning, and academic tracking.

5.1 Achievements

Coursepedia has achieved several significant milestones:

- **User-Centric Design:** The platform has been meticulously designed with the end-user in mind, ensuring ease of use and accessibility. Features like student login, course browsing, and automated course plan recommendations are tailored to enhance user engagement and academic planning.
- **Technical Robustness:** Through iterative development and rigorous testing, the software has proven to be reliable and secure, managing user data with high standards of integrity and security.
- **Adaptive Features:** The system's ability to adapt to various academic requirements and its flexibility in handling course data demonstrate the technical prowess and forward-thinking approach of the development team.

5.2 Reflections

- **Yongyu's Reflection** I've gained valuable insights into balancing expectations and workload, as our initial goals were over-ambitious.Having a realistic understanding of the scope and complexity of the tasks involved is crucial. Data collection proved challenging and time-consuming, and implementing a sophisticated recommendation algorithm within our timeframe was daunting.Fortunately, I had supportive teammates who contributed strong algorithm designs and helped with data collection. I also recognized the importance of establishing a standardized team development workflow. Effective communication, clear task delegation, and regular progress updates are essential for ensuring project success and maintaining team cohesion.
- **Haochen's Reflection**: What I learned from the project is that sometimes you need to do some trade offs to meet deadline. Like Yuxuan said, our project initially want to use the automatic script for every majors. What's more, we also want to use the shibboleth api to validate the student's nyu account. Although these are cool ideas, the request for accessing shibboleth api was rejected, while we have no time managed to finish all the recommend script for the all majors. In order to meet the deadline, we have to give up these fancy but time-consuming ideas and apply the manually sign-up and parse course history instead. All in all, all of us learned a lot from the project and the lesson will be useful in our future career.

- **Yuxuan's Reflection**: I learned that don't be too optimistic when developing software. We were initially trying to implement course recommendations for all majors but ended up finding it was impossible to do this because each major has its requirements and there are 19 majors. It's not feasible for our three-member team. In addition, I learned not to try to automate everything. When I was building the dataset, I spent two weeks writing an automated parse script that can be hand-filling in less than an hour. The automation obsession wasted a lot of time and I learned to evaluate the cost of manual operation and script-writing to save time in my future projects.

5.3 Future Directions

Looking ahead, several enhancements are envisioned:

- **Expansion of Course Offerings:** Integrating additional majors and specializations to cater to a broader student population.
- **Increased Customization:** Developing more personalized user interfaces and learning paths based on individual academic histories and preferences.
- **Advanced Data Analytics:** Utilizing machine learning to refine course recommendations and predict student academic outcomes.

In conclusion, Coursepedia stands as a testament to the power of innovative software solutions in transforming educational environments. The project not only fulfills its initial mandates but sets a foundation for continual enhancement and adaptation to meet the evolving needs of the educational sector.