Project Documentation: Learning Management System (LMS)

JavaFX Desktop Application

---

1. Project Description

This project is a high-performance desktop-based Learning Management System (LMS) developed using **Java 17**, **JavaFX**, and advanced **Object-Oriented Programming (OOP)** principles.

The application provides a structured and secure environment where students, instructors, and administrators can manage educational workflows. It is built using a **Layered Architecture (MVC + Service + Repository)** and strictly adheres to **SOLID principles**, ensuring the system is scalable, maintainable, and robust.

---

2. Problem Statement

Traditional learning environments often suffer from fragmented, manual processes that lead to data inconsistency, inefficiency, and security vulnerabilities.

Our Goal:

- Develop a centralized digital platform for seamless educational management.
- Implement a real-world application of **OOP** and **SOLID** design patterns.
- Demonstrate professional software architecture using the **JavaFX** framework.

---

3. System Features

A. Authentication & Security

- **Secure Access:** Role-based login and registration.
- **Data Protection:** Sensitive passwords are encrypted using **SHA-256 hashing**.
- **Session Management:** State is maintained via a centralized `SessionManager` to ensure secure user transitions.

B. User Roles

- **Student:** Can browse the catalog, enroll in courses, and track their learning progress.
- **Instructor:** Empowered to create, update, and manage course content.
- **Admin:** Oversees system-level operations, including user management and platform maintenance.

C. Course & Enrollment Management

- **Full CRUD Operations:** Comprehensive management of course data.
- **Smart Enrollment:** Automated checks prevent duplicate enrollments and ensure data integrity.
- **Persistence:** All data is persisted via a file-based storage system using the Repository Pattern.

---

4. System Architecture

The application is divided into four distinct layers to ensure a **clean separation of concerns**:

1. **UI Layer (Presentation):** JavaFX Controllers and FXML files.
2. **Service Layer (Business Logic):** Validates and processes all core operations.
3. **Repository Layer (Persistence):** Manages data storage and retrieval from flat files.
4. **Model Layer (Entities):** Defines the structure of core objects like Users and Courses.

---

5. Object-Oriented Programming (OOP) Concepts

1. Encapsulation

All data fields are kept `private` and accessed only via public `getters` and `setters`. This ensures data integrity by preventing unauthorized external modifications.

2. Abstraction

The system uses an **Abstract User Class** to define common traits, while **Interfaces** (like `UserRepository`) hide complex implementation details. This allows the system to switch storage methods (e.g., from `.txt` to SQL) without affecting the UI or business logic.

3. Inheritance

`Student`, `Instructor`, and `Admin` inherit from the base `User` class. This eliminates code duplication and enforces a consistent structure across all user types.

4. Polymorphism

The system uses method overriding (e.g., `getRole()`) so that the application can treat different objects as a generic `User` while executing role-specific behavior at runtime.

---

6. SOLID Principles Applied

- **Single Responsibility (SRP):** Every class has a single focus (e.g., a Controller only handles UI events, never file saving).
- **Open/Closed (OCP):** The system is open for extension (adding new roles) but closed for modification of existing, tested logic.
- **Liskov Substitution (LSP):** Subclasses like `Student` can replace the `User` parent class without breaking the application logic.
- **Interface Segregation (ISP):** Clients are not forced to depend on methods they do not use; repositories are kept lean and specific.
- **Dependency Inversion (DIP):** High-level services depend on **Interfaces**, not concrete implementations, providing maximum flexibility.

---

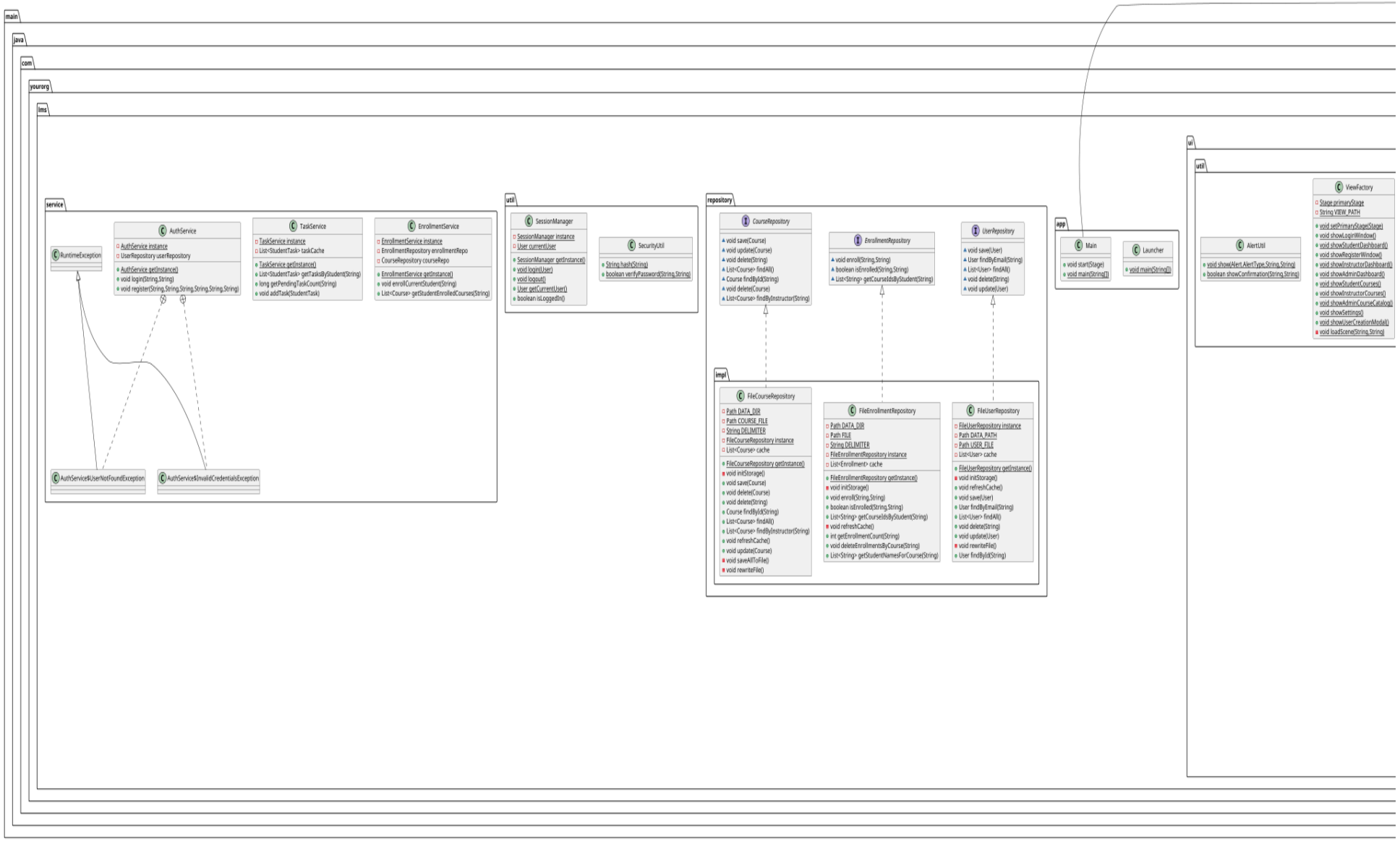
7. Design Patterns Used

1. **Factory Pattern (`UserFactory`):** Centralizes object creation and ensures password hashing is applied consistently.
2. **Singleton Pattern:** Used for `SessionManager` and `Services` to ensure a consistent state across the entire application.
3. **Repository Pattern:** Decouples the business logic from the specific file-handling code.

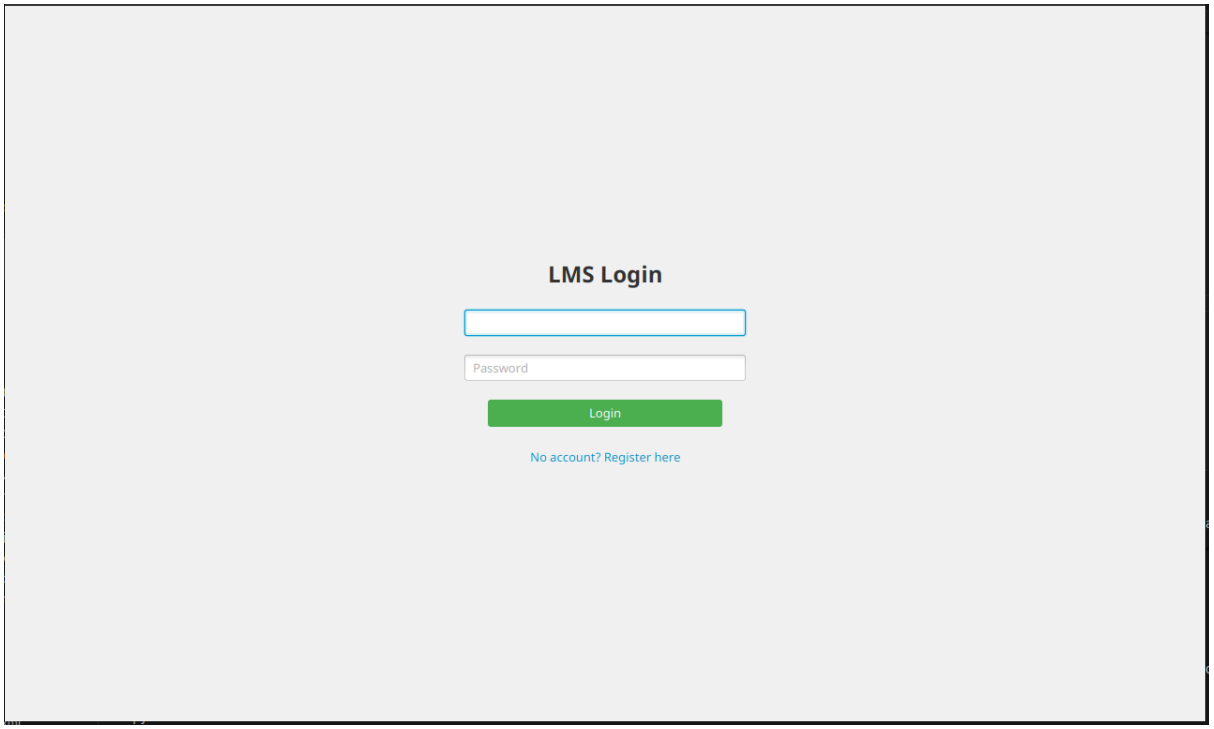
---

8. Conclusion

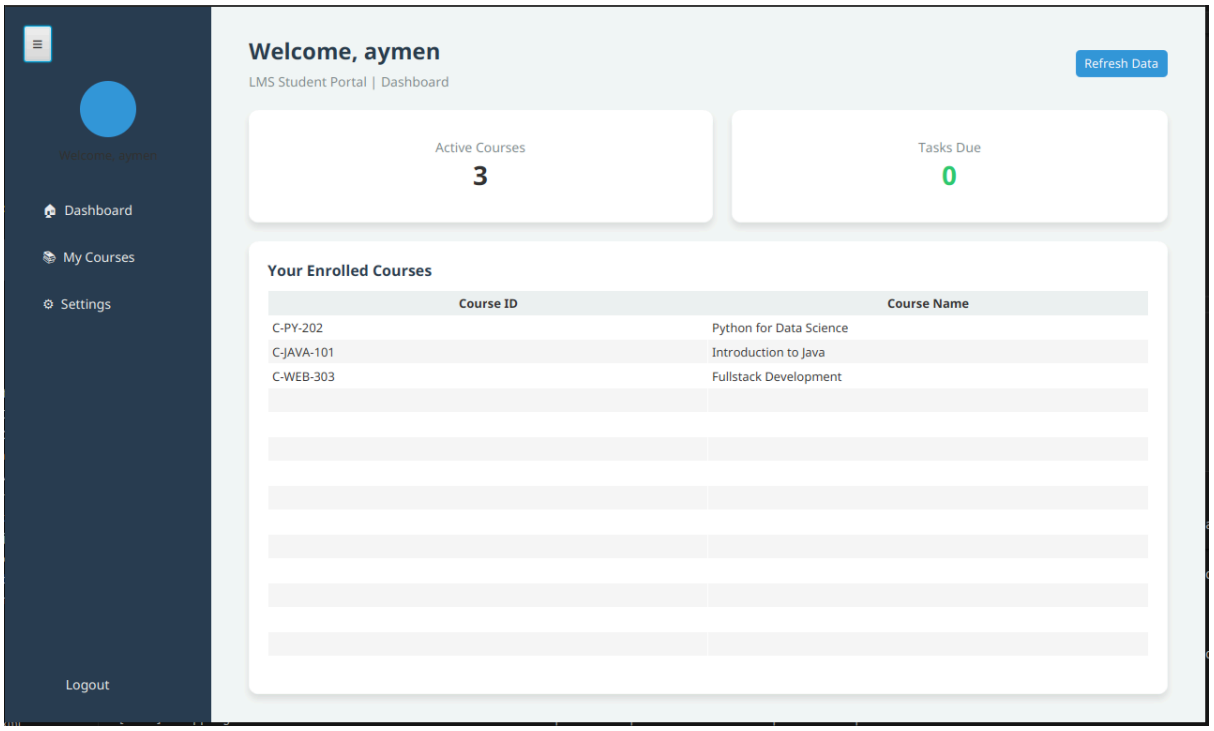
This LMS project serves as a comprehensive demonstration of professional software development. By strictly following **SOLID** principles and a **Layered Architecture**, the resulting application is secure, highly maintainable, and ready for future feature expansion.


main
java
com
yourorg
lms
service
RuntimeException
AuthService
- AuthService instance
- UserRepository userRepository
+ AuthService getInstance()
+ void login(String,String)
+ void register(String,String,String,String,String,String)
TaskService
- TaskService instance
- List<StudentTask> taskCache
+ TaskService getInstance()
+ List<StudentTask> getTasksByStudent(String)
+ long getPendingTaskCount(String)
+ void addTask(StudentTask)
EnrollmentService
- EnrollmentService instance
- EnrollmentRepository enrollmentRepo
- CourseRepository courseRepo
+ EnrollmentService getInstance()
+ void enrollCurrentStudent(String)
+ List<Course> getStudentEnrolledCourses(String)
AuthService$UserNotFoundException
AuthService$InvalidCredentialsException
util
SessionManager
- SessionManager instance
- User currentUser
+ SessionManager getInstance()
+ void login(User)
+ void logout()
+ User getCurrentUser()
+ boolean isLoggedIn()
SecurityUtil
+ String hash(String)
+ boolean verifyPassword(String,String)
repository
CourseRepository
+ void save(Course)
+ void update(Course)
+ void delete(String)
+ List<Course> findAll()
+ Course findById(String)
+ void delete(Course)
+ List<Course> findByInstructor(String)
EnrollmentRepository
+ void enroll(String,String)
+ boolean isEnrolled(String,String)
+ List<String> getCourseIdsByStudent(String)
UserRepository
+ void save(User)
+ User findByEmail(String)
+ List<User> findAll()
+ void delete(String)
+ void update(User)
impl
FileCourseRepository
- Path DATA_DIR
- Path COURSE_FILE
- String DELIMITER
- FileCourseRepository instance
- List<Course> cache
+ FileCourseRepository getInstance()
- void initStorage()
+ void save(Course)
+ void delete(Course)
+ void delete(String)
+ Course findById(String)
+ List<Course> findAll()
+ List<Course> findByInstructor(String)
+ void refreshCache()
+ void update(Course)
- void saveAllToFile()
- void rewriteFile()
FileEnrollmentRepository
- Path DATA_DIR
- Path FILE
- String DELIMITER
- FileEnrollmentRepository instance
- List<Enrollment> cache
+ FileEnrollmentRepository getInstance()
- void initStorage()
+ void enroll(String,String)
+ boolean isEnrolled(String,String)
+ List<String> getCourseIdsByStudent(String)
- void refreshCache()
+ int getEnrollmentCount(String)
+ void deleteEnrollmentsByCourse(String)
+ List<String> getStudentNamesForCourse(String)
FileUserRepository
- FileUserRepository instance
- Path DATA_PATH
- Path USER_FILE
- List<User> cache
+ FileUserRepository getInstance()
- void initStorage()
+ void refreshCache()
+ void save(User)
+ User findByEmail(String)
+ List<User> findAll()
+ void delete(String)
+ void update(User)
- void rewriteFile()
+ User findById(String)
app
Main
+ void start(Stage)
+ void main(String[])
Launcher
+ void main(String[])
ui
util
AlertUtil
+ void show(Alert.AlertType,String,String)
+ boolean showConfirmation(String,String)
ViewFactory
- Stage primaryStage
- String VIEW_PATH
+ void setPrimaryStage(Stage)
+ void showLoginWindow()
+ void showStudentDashboard()
+ void showRegisterWindow()
+ void showInstructorDashboard()
+ void showAdminDashboard()
+ void showStudentCourses()
+ void showInstructorCourses()
+ void showAdminCourseCatalog()
+ void showSettings()
+ void showUserCreationModal()
- void loadScene(String,String)

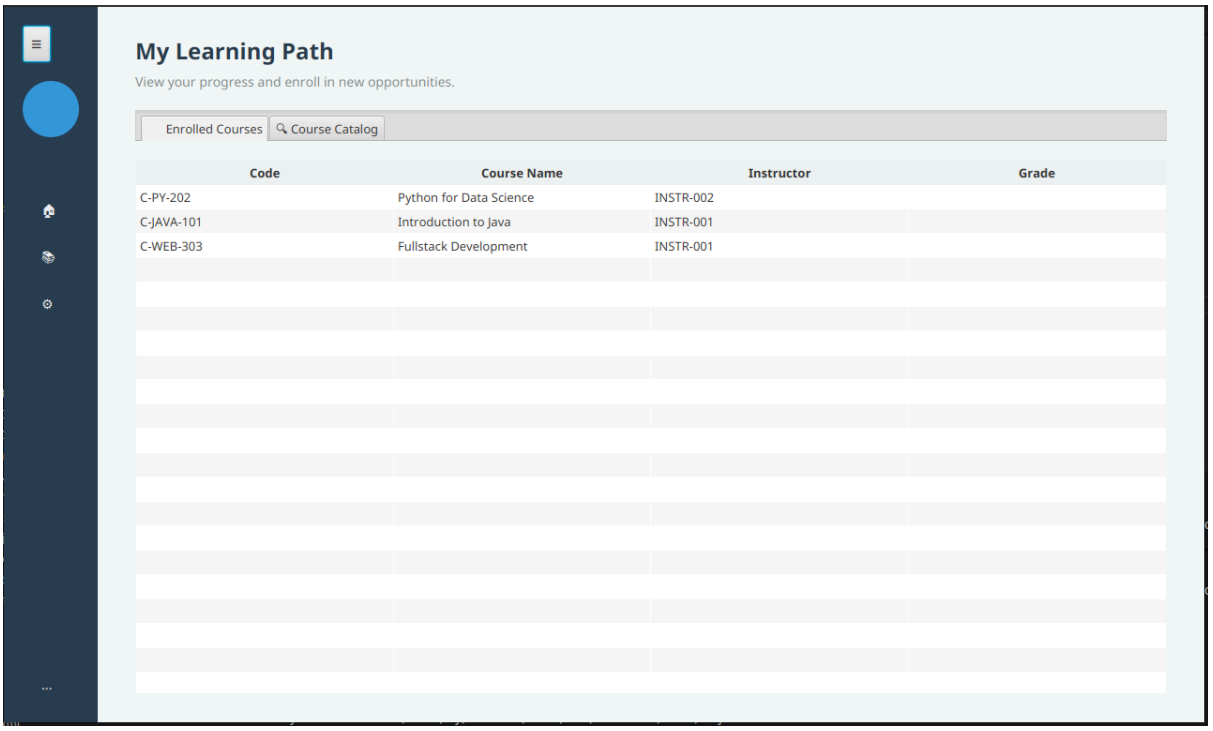
Login page



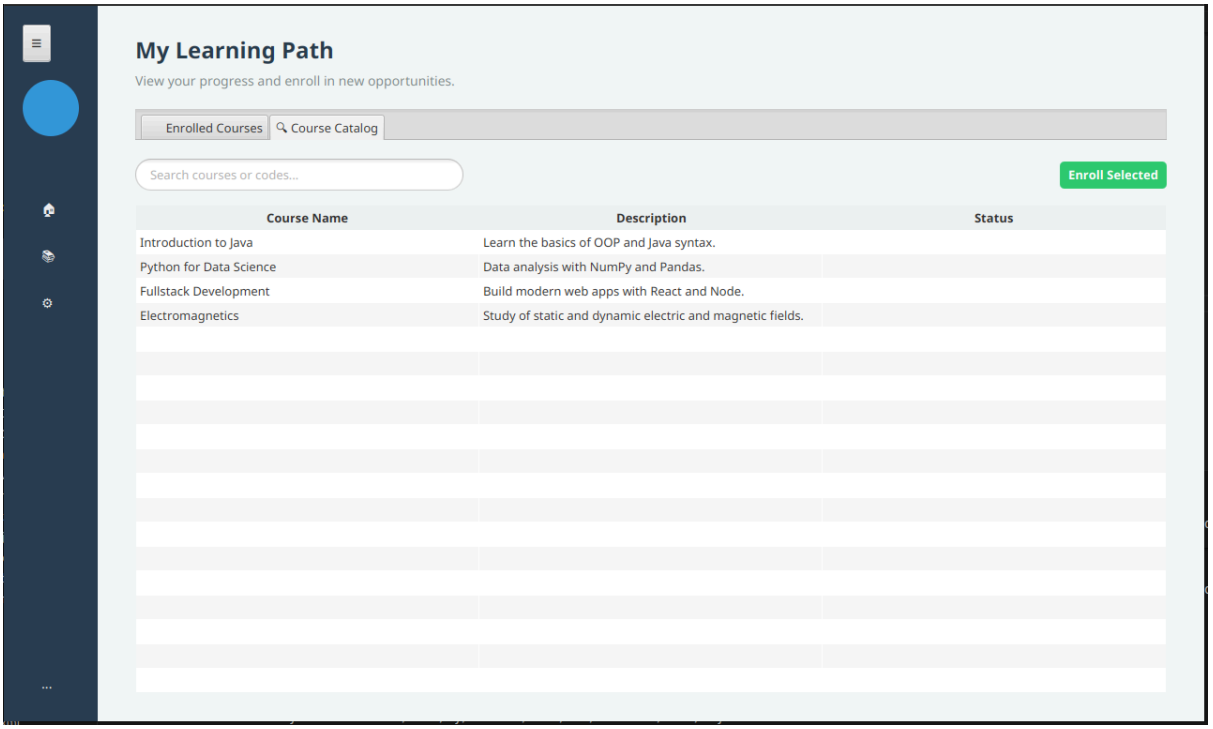
Student page



Learning section



Couse catalog



Settings Section