1-Refine the Class Diagram into a Design Class Diagram:

Classes:

1. User (abstract)

Attributes

- userID : int
- name : String
- email : String
- password : String

Methods

- login(email:String, password:String) : boolean
- logout() : void

2. Student (extends User)

Attributes

- level : int
- major : String

Methods

- browseCourses() : List<CourseOffering>
- registerCourse(offeringID:int) : boolean
- dropCourse(offeringID:int) : boolean
- viewSchedule() : Schedule
- submitSpecialRequest(offeringID:int, reason:String) : boolean

3. Instructor (extends User)

Attributes

- department : String

Methods

- viewCourses() : List<CourseOffering>
- viewStudents(offeringID:int) : List<Student>
- approveRequest(requestID:int) : void
- rejectRequest(requestID:int) : void

4. Administrator (extends User)

Attributes

- adminRole : String

Methods

- manageCourses() : void
- manageUsers() : void
- setRegistrationRules(rules:RegistrationRules) : void
- openRegistration(term:String) : void
- closeRegistration(term:String) : void

5. Course

Attributes

- courseCode : String
- title : String
- credits : int
- description : String

Methods

- getCourseInfo() : String

6. CourseOffering

Attributes

- offeringID : int
- capacity : int
- currentEnrollment : int

Methods

- getOfferingInfo() : String
- increaseEnrollment() : void
- decreaseEnrollment() : void
- checkAvailability() : boolean

7. Registration

Attributes

- registrationID : int
- status : String (registered / rejected / pending)

- timestamp : DateTime

Methods

- getRegistrationInfo() : String

8. Semester

Attributes

- termName : String
- startDate : Date
- endDate : Date
- registrationStatus : String (Open/Closed)

Methods

- getSemesterInfo() : String

9. Schedule

Attributes

- days : String
- startTime : Time
- endTime : Time

Methods

- getSchedule() : String

10. SpecialRequest

Attributes

- requestID : int
- reason : String
- status : String (pending/approved/rejected)

Methods

- getRequestInfo() : String

11. Notification

Attributes

- notificationID : int
- message : String
- date : DateTime

Methods

- showNotification() : void

12. RegistrationRules

Attributes

- maxCredits : int
- allowConflicts : boolean
- allowPrereqOverride : boolean

Methods

- getRules() : String

Relationships:

1. User Inheritance Hierarchy

The classes *Student*, *Instructor*, and *Administrator* all inherit from the abstract class *User*.

This allows all three user types to share common attributes and operations such as login and logout
Relationship: Inheritance

2. Student and Registration

A student can have multiple registration records.

Each registration is linked to exactly one student.

Relationship: 1-to-many (Student → Registration)

3. Student and SpecialRequest

A student may submit several special requests throughout the semester.

Each special request is created by one specific student.

Relationship: 1-to-many (Student → SpecialRequest)

4. Course and CourseOffering

A course may have multiple offerings across different semesters or sections.

Each course offering is tied to exactly one course.

Relationship: 1-to-many (Course → CourseOffering)

5. Instructor and CourseOffering

An instructor may teach multiple course offerings.

Each offering is assigned to exactly one instructor.

Relationship: 1-to-many (Instructor → CourseOffering)

6. Instructor and SpecialRequest

An instructor may receive and process multiple special requests.

Each special request is handled by one instructor.

Relationship: 1-to-many (Instructor → SpecialRequest)

7. Semester and CourseOffering

A semester contains multiple course offerings.

Each course offering belongs to exactly one semester.

Relationship: 1-to-many (Semester → CourseOffering)

8. CourseOffering and Schedule (Composition)

Each course offering has exactly one schedule.

A schedule cannot exist independently without its course offering.

Relationship: 1-to-1 (Composition)

9. CourseOffering and Registration

A course offering can have multiple registrations from students.

Each registration refers to exactly one course offering.

Relationship: 1-to-many (CourseOffering → Registration)

10. User and Notification

Any user may receive multiple notifications.

Each notification is associated with one user.

Relationship: 1-to-many (User → Notification)

11. Administrator and RegistrationRules

There is exactly one set of registration rules in the system at any time.

This rule set is managed by an administrator.

Relationship: 1-to-1 (Administrator → RegistrationRules)

Digram

2-Package Diagram

Package: UI

Description: Contains all user-facing interfaces such as the Student Dashboard, Course List, and Registration Form.

Dependencies: Uses the API Layer to request and display data.

Package: API

Description: Handles incoming HTTP requests from UI, routes them to the correct service, performs validation and security checks.

Dependencies: Depends on Auth, User Management, Course Management, and Registration packages.

Package: Auth

Description: Manages login, logout, token validation, and role-based access control.

Dependencies: Uses Persistence package to retrieve user credentials.

Package: User Management

Description: Handles data and operations related to students, instructors, and administrators.

Dependencies: Communicates with Persistence package to access user information.

Package: Course Management

Description: Provides operations to manage courses, sections, schedules, and instructor assignments.

Dependencies: Uses Persistence; accessed by Registration package.

Package: Registration

Description: Contains the core logic for course enrollment including prerequisites validation, capacity checking, and conflict detection.

Dependencies: Uses Course Management, Schedule, Persistence, and Notification packages.

Package: Schedule

Description: Handles time slot management and schedule conflict computation.

Dependencies: Used by Registration package.

Package: Notification

Description: Sends confirmation or failure messages to students (email, SMS, or in-app notifications).

Dependencies: Triggered by Registration package.

Package: Persistence

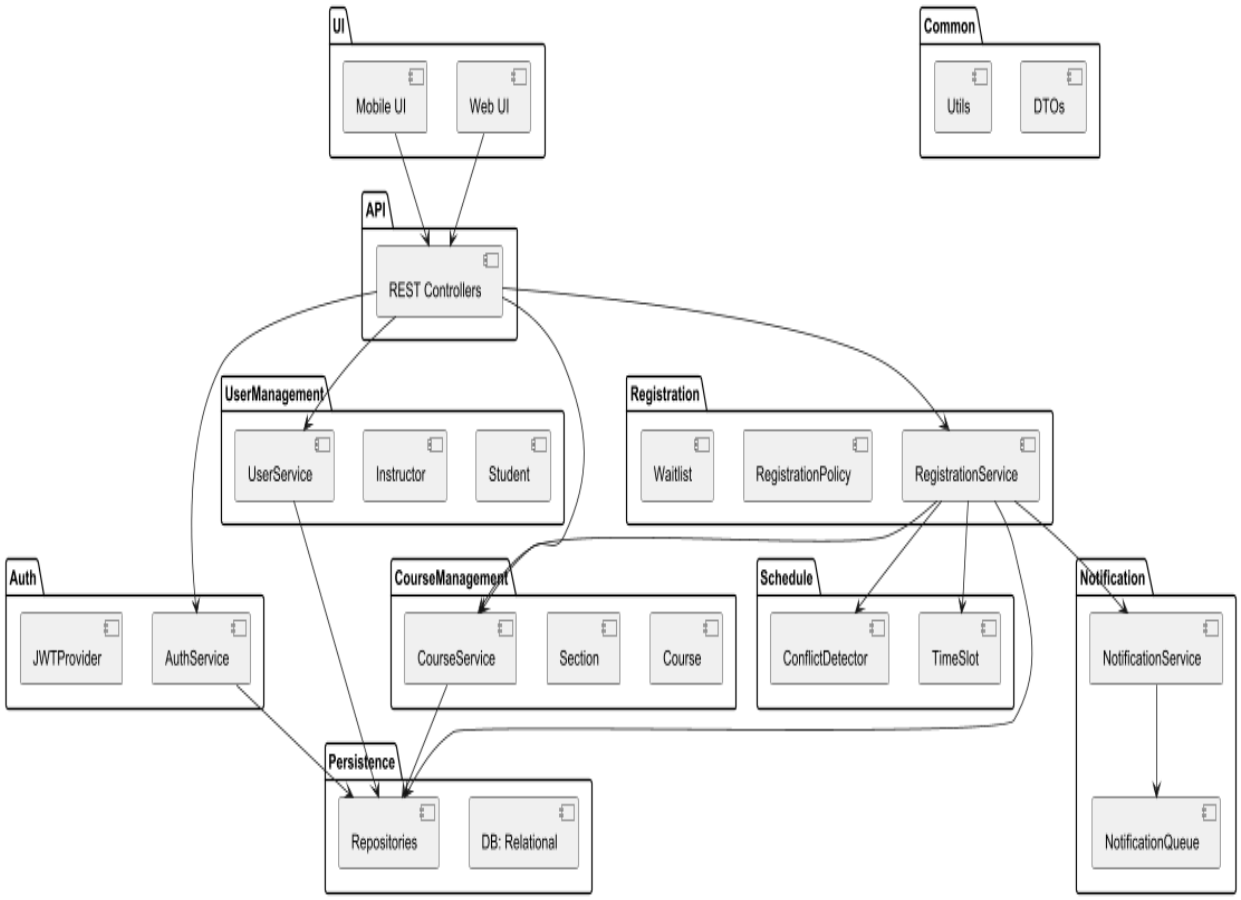
Description: Responsible for database communication, repositories, and data storage access.

Dependencies: Used by Auth, User Management, Course Management, Registration.

Package: Common

Description: Contains shared utilities, DTOs, validators, and exceptions used across multiple packages.

Dependencies: Shared by most system packages.



3-component Diagram

Component: Web UI

Description: Front-end pages that allow students, instructors, and admins to interact with the system.

Interfaces: Requires REST API endpoints to display and update system data.

Component: API Gateway / Controllers

Description: Receives HTTP requests and delegates them to the appropriate service component.

Interfaces: Provides all REST API endpoints. Requires Auth Service to validate authentication.
Requires Service Layer components.

Component: Auth Service

Description: Handles authentication, token creation, user verification, and authorization policies.

Interfaces: Provides /login, /validateToken. Requires database access via Persistence.

Component: User Management Service

Description: Manages student, instructor, and admin profiles.

Interfaces: Provides CRUD operations for user entities. Requires Persistence for data retrieval.

Component: Course Service

Description: Manages course creation, modification, section schedules, and instructor assignments.

Interfaces: Provides courses API functions.
Requires Persistence.

Component: Registration Service

Description: Contains the workflow of course registration:
– prerequisites check
– capacity verification
– schedule conflict detection
– final approval or rejection

Interfaces: Provides registrations.Requires Course Service, Schedule Service, Notification Service, Persistence.

Component: Schedule Service

Description: Manages time slots and detects schedule clashes.

Interfaces: Used directly by Registration Service.

Component: Notification Service

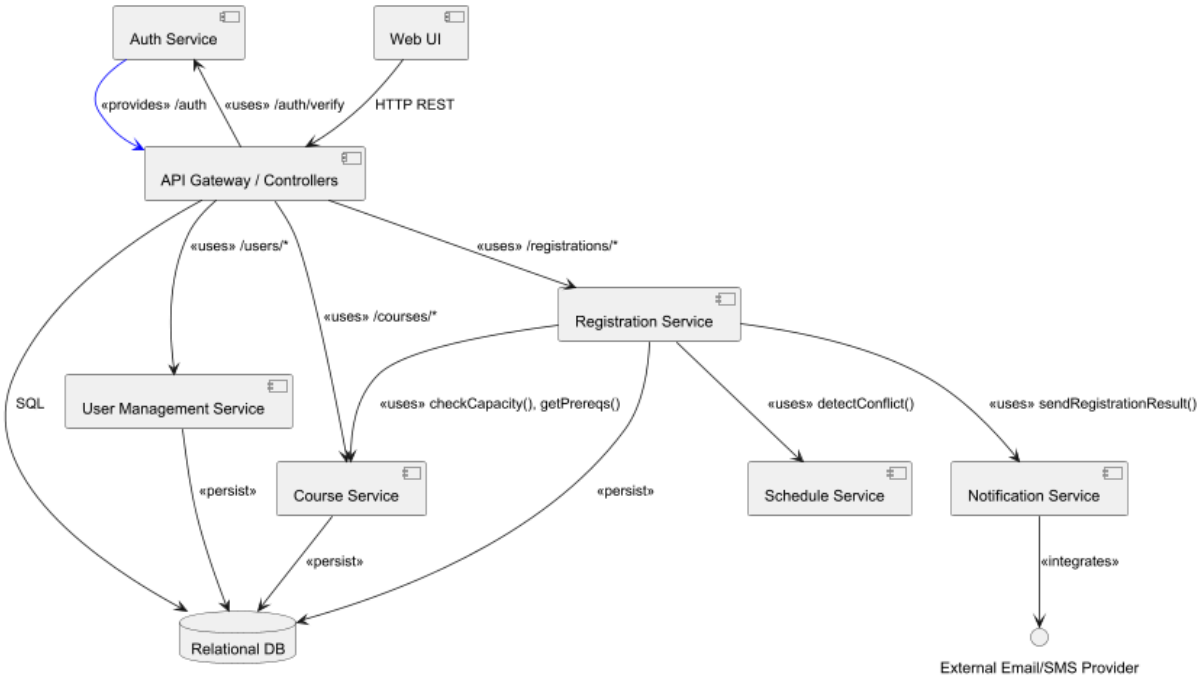
Description: Sends emails / messages to users upon success or failure of registration.

Interfaces: Triggered by Registration Service.

Component: Database (Persistence Layer)

Description: Relational database with tables such as Users, Courses, Sections, Registrations, Waitlists.

Interfaces: Provides data access to all services.



4-State Machine Diagram

State Machine Diagram Description :

State: Idle

Description:
The student has not initiated the registration process yet.

Transition:

- Event: *Select Course* → Moves to **Course Selected**

State: Course Selected

Description:
The student has chosen a specific course from the available list.

Transition:

- Event: *System starts validation* → Moves to **Checking Prerequisites**

State: Checking Prerequisites

Description:
The system verifies whether the student has completed all required prerequisite courses.

Transitions:

- If prerequisites satisfied → **Checking Capacity**
- If prerequisites missing → **Registration Failed: Prerequisite Not Met**

State: Checking Capacity

Description:
The system checks whether there are available seats in the selected course.

Transitions:

- If seats are available → **Checking Schedule Conflicts**
- If the course is full → **Registration Failed: No Capacity**

State: Checking Schedule Conflicts

Description:
The system checks if the course schedule conflicts with any other registered course.

Transitions:

- If no conflict → **Registration Approved**

- If conflict exists → **Registration Failed: Schedule Conflict**

State: Registration Approved

Description:
The student is successfully registered in the course.

Transition:

- Event: *Finish* → Returns to **Idle**

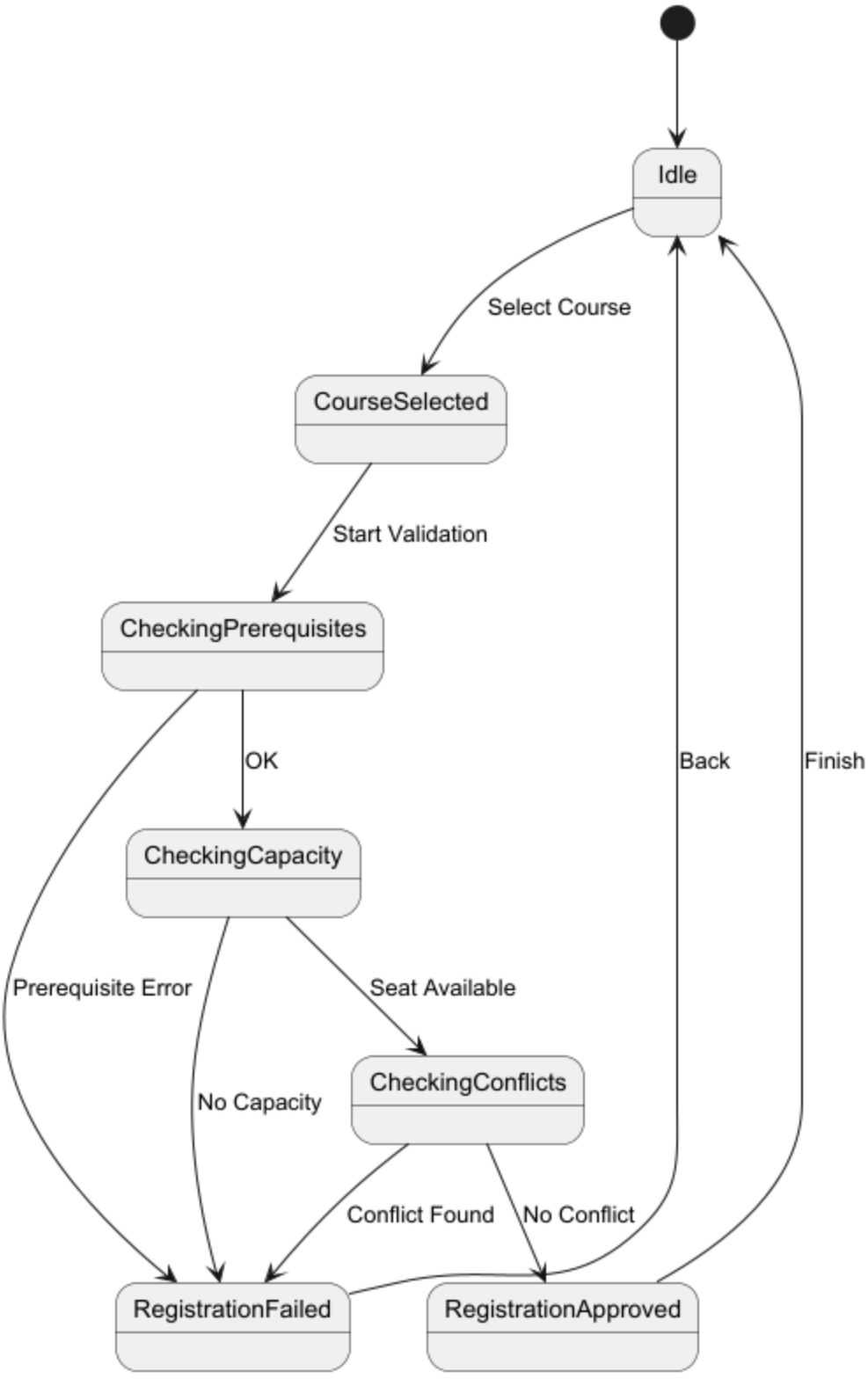
State: Registration Failed

Description:
The registration attempt failed for one of the following reasons:

- Missing prerequisites
- No available seats
- Schedule conflict
- Registration period closed

Transition:

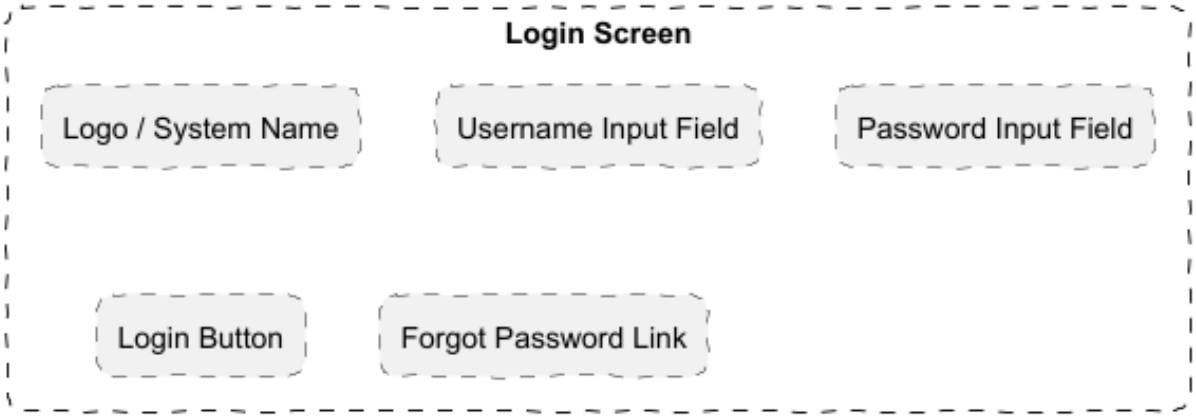
- Event: *Back to Course List* → Returns to **Idle**


Idle
Select Course
CourseSelected
Start Validation
CheckingPrerequisites
OK
CheckingCapacity
Prerequisite Error
No Capacity
Seat Available
CheckingConflicts
Conflict Found
No Conflict
RegistrationFailed
RegistrationApproved
Back
Finish

5-UI Mockups

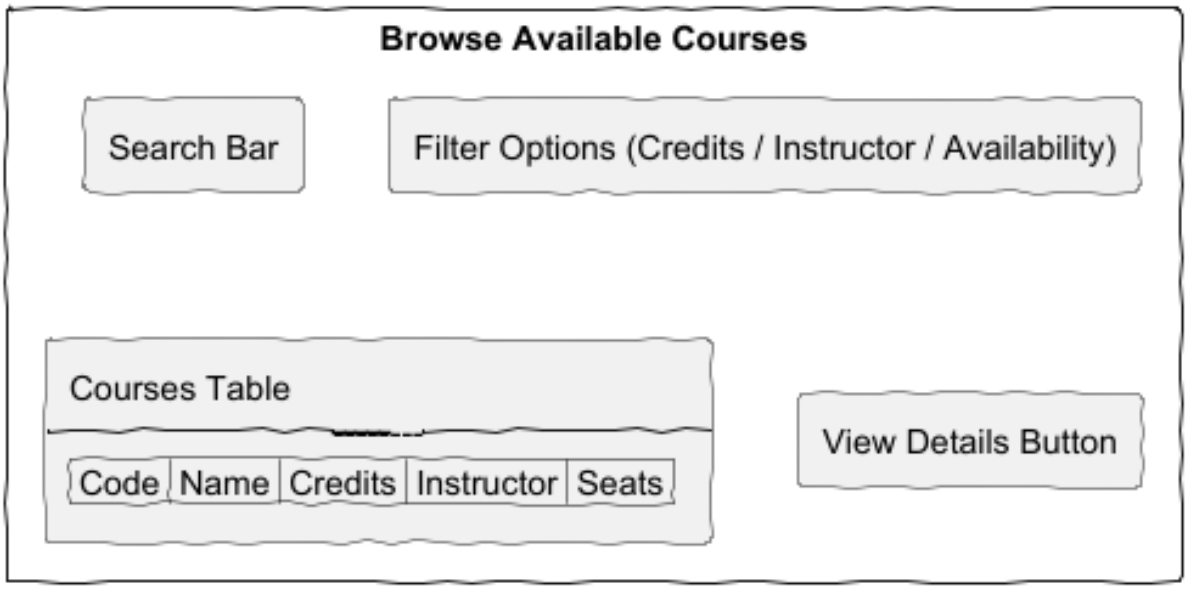
Wireframe 1 – Login Screen

```
Please use '!option handwritten true' to enable handwritten
```



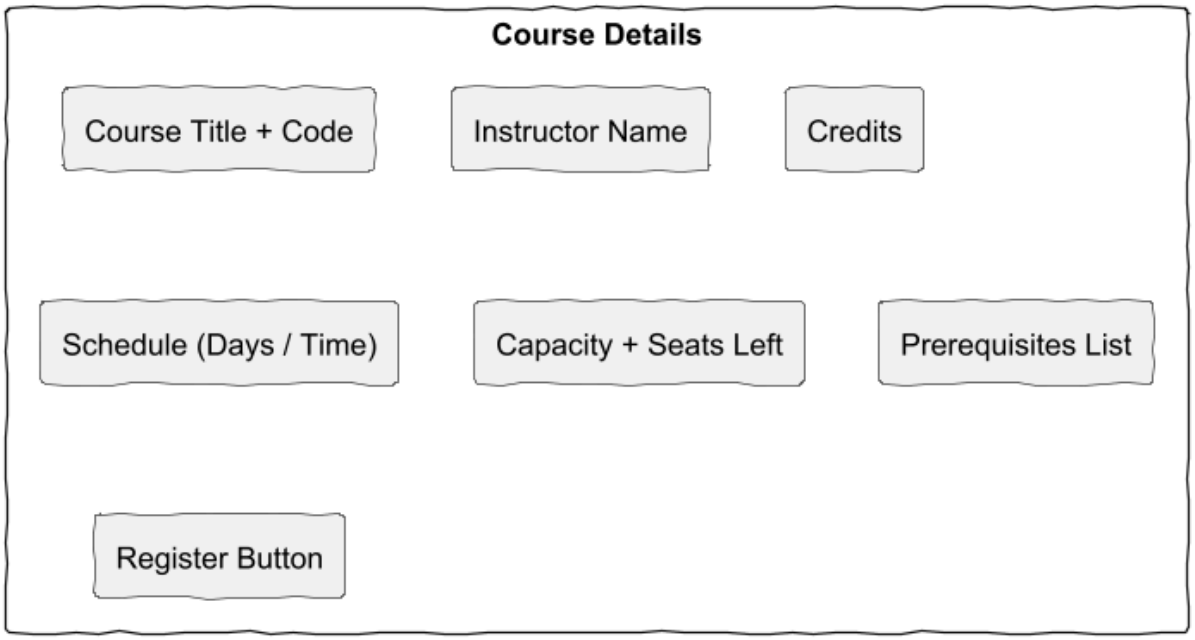
Wireframe 2 – Browse Courses

```
Please use '!option handwritten true' to enable handwritten
```



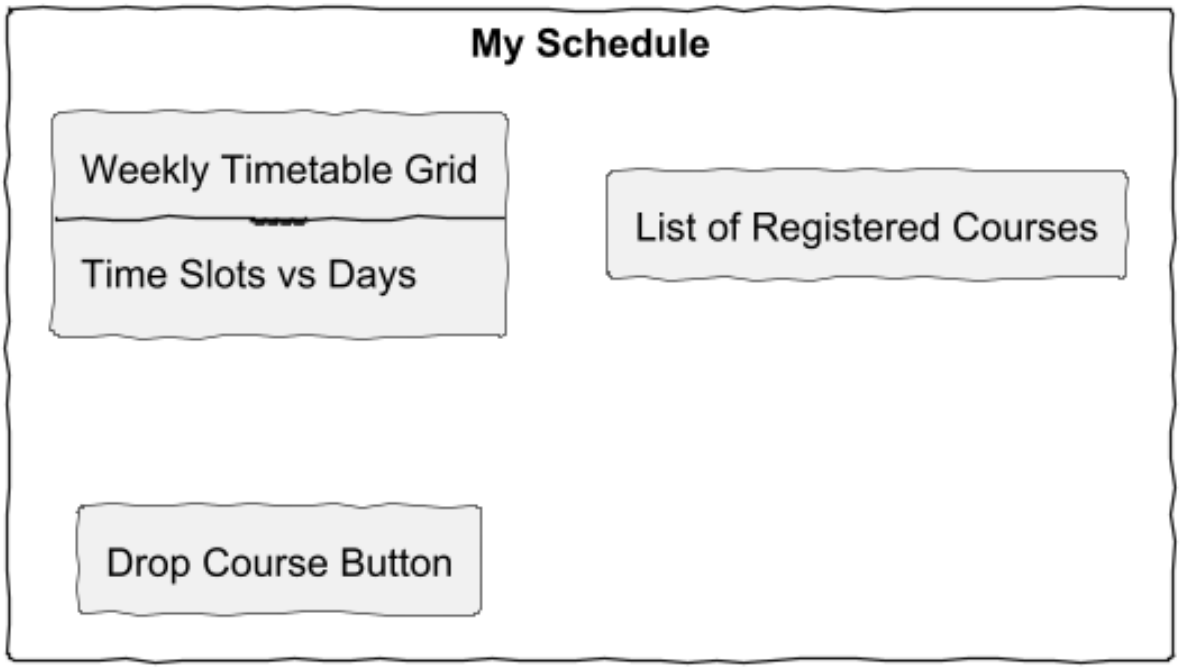
Wireframe 3 – Course Details Screen

```
Please use '!option handwritten true' to enable handwritten
```



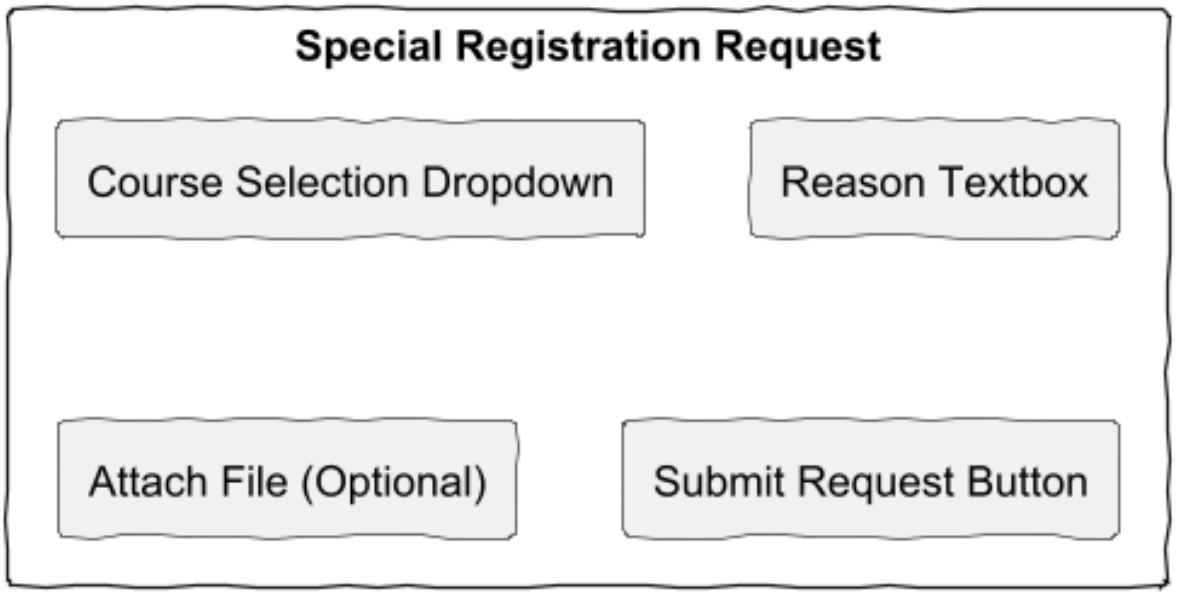
Wireframe 4 – Student Schedule Screen

```
Please use '!option handwritten true' to enable handwritten
```



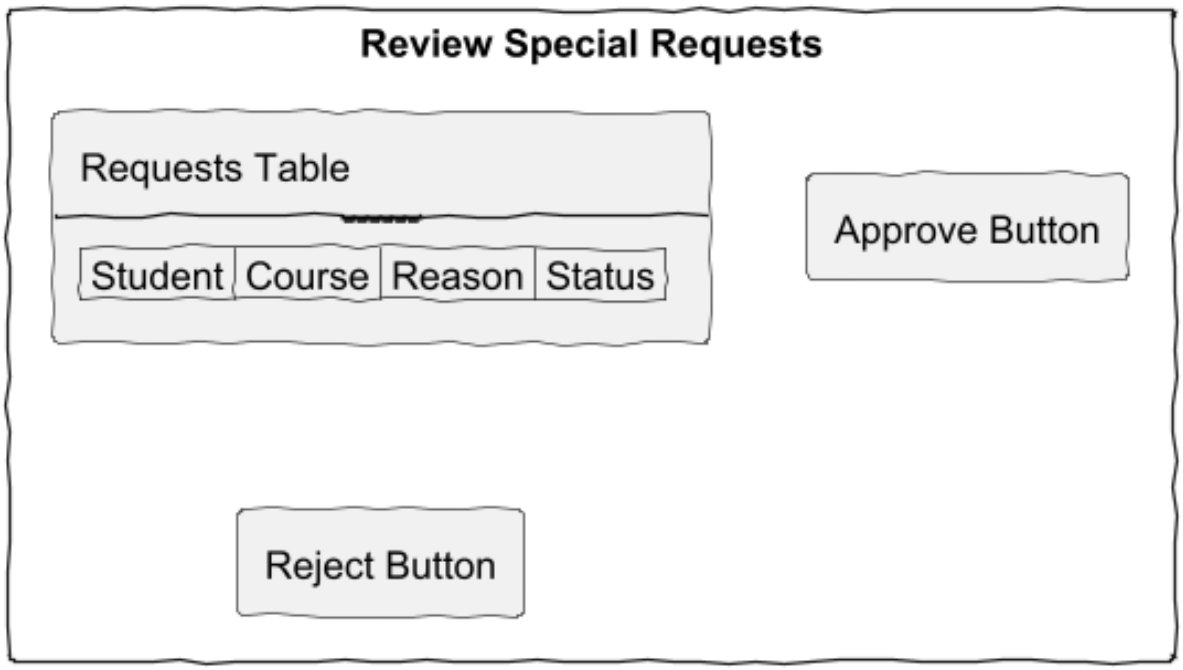
Wireframe 5 – Special Request Form

```
Please use '!option handwritten true' to enable handwritten
```



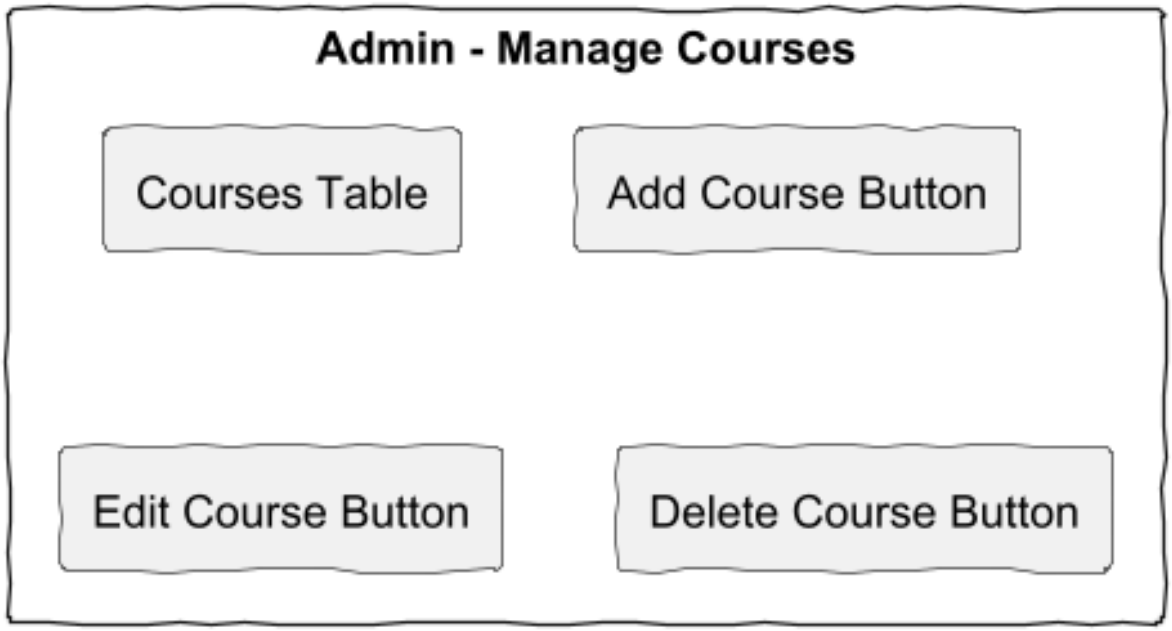
Wireframe 6 – Instructor Request Review

```
Please use '!option handwritten true' to enable handwritten
```



Wireframe 7 – Admin Manage Courses

```
Please use '!option handwritten true' to enable handwritten
```



6-Design Summary

The behavioral aspects and user interaction layers of the Course Registration System were designed to ensure a clear, maintainable, and user-friendly implementation.

1. Behavioral Modeling using State Machine Diagram

A State Machine Diagram was developed for the **Student Course Registration** process, which is the most dynamic and critical object in the system. The diagram models the lifecycle of a registration attempt from the idle state through sequential validation steps (prerequisites, capacity, and schedule conflicts) to either successful registration or failure with specific causes. This behavioral representation explicitly captures all possible states and transitions, making the validation logic traceable, verifiable, and easy to implement using the State design pattern or conditional control

structures. It directly refines the “Register for Course” use case from the analysis phase and ensures that error handling and process flow are well-defined before coding begins.

2. User Interface Design (Wireframes)

Low-fidelity wireframes were created for the main screens accessed by different actors (Student, Instructor, and Administrator). These wireframes focus on layout structure, navigation flow, and placement of key interaction elements rather than visual styling. The selected screens include Login, Browse Courses, Course Details, Student Schedule, Special Request Form, Instructor Request Review, and Admin Course Management. The wireframes illustrate a clean, task-oriented interface that supports intuitive progression through the core functionalities identified in the use cases, while maintaining clear separation of concerns between actor roles.

3. Key Design Decisions & Justifications

- **State-based modeling** was chosen for registration because the process involves multiple sequential checks with distinct outcomes, making the State pattern a natural and robust solution that avoids complex nested conditionals.
- **Separation of actor-specific interfaces** (Student / Instructor / Admin dashboards) follows the principle of least privilege and improves usability by presenting only relevant functionality to each user type.
- **Simple, low-fidelity wireframes** were preferred over high-fidelity prototypes because the focus of this phase is system design and behavioral correctness, not final visual design. These wireframes serve as a communication bridge between requirements and implementation teams.
- The overall design adopts a **layered architecture** with clear boundaries between presentation (wireframes), domain logic (state machine and design classes), and data persistence, ensuring maintainability and scalability.