

**UE22CS352B**

**Object Oriented Analysis & Design Using Java**

**Mini Project Report**

**Exam Management System**

***Submitted by:***

Vignesh Singaravelu (PES1UG22CS691)

Vedanth Padmaraman (PES1UG22CS682)

Vidwan K V (PES1UG22CS687)

Vishnu Srinivas (PES1UG22CS720)

6TH Semester Section: L

# Facultly Name

**January - May 2025**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

FACULTY OF ENGINEERING

**PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)

100ft Ring Road, Bengaluru – 560 085, Karnataka, India

**Problem Statement: Exam management system**

**Key features:**

1. User Roles and Management

* Supports multiple user roles: Teachers, Students, and Controller of Examination (CoE) and Exam coordinator
* Role-based access control and permissions
* Separate dashboards tailored to each user role

1. Exam Creation and Management

* Teachers can create and manage exams
* Exams must be proofread by another teacher before publication
* Customizable exam details:
  + Subject
  + Duration
  + Total number of questions
  + Total marks
  + Exam date and time

1. Exam States and Workflow

* Complete exam lifecycle management with the following states:
  + DRAFT: Initial creation by teacher
  + PROOFREAD: Reviewed by another teacher
  + STORED\_IN\_COE: Stored with the Controller of Examination
  + ACTIVE: Available for students to attempt
  + COMPLETED: Exam duration has ended
  + EVALUATED: Grading completed
  + RESULTS\_PUBLISHED: Results are released to students

1. Student Features

* Exam registration system
* User-friendly interface for taking exams
* View upcoming exam using the schedules
* Access to exam results

1. Results Management

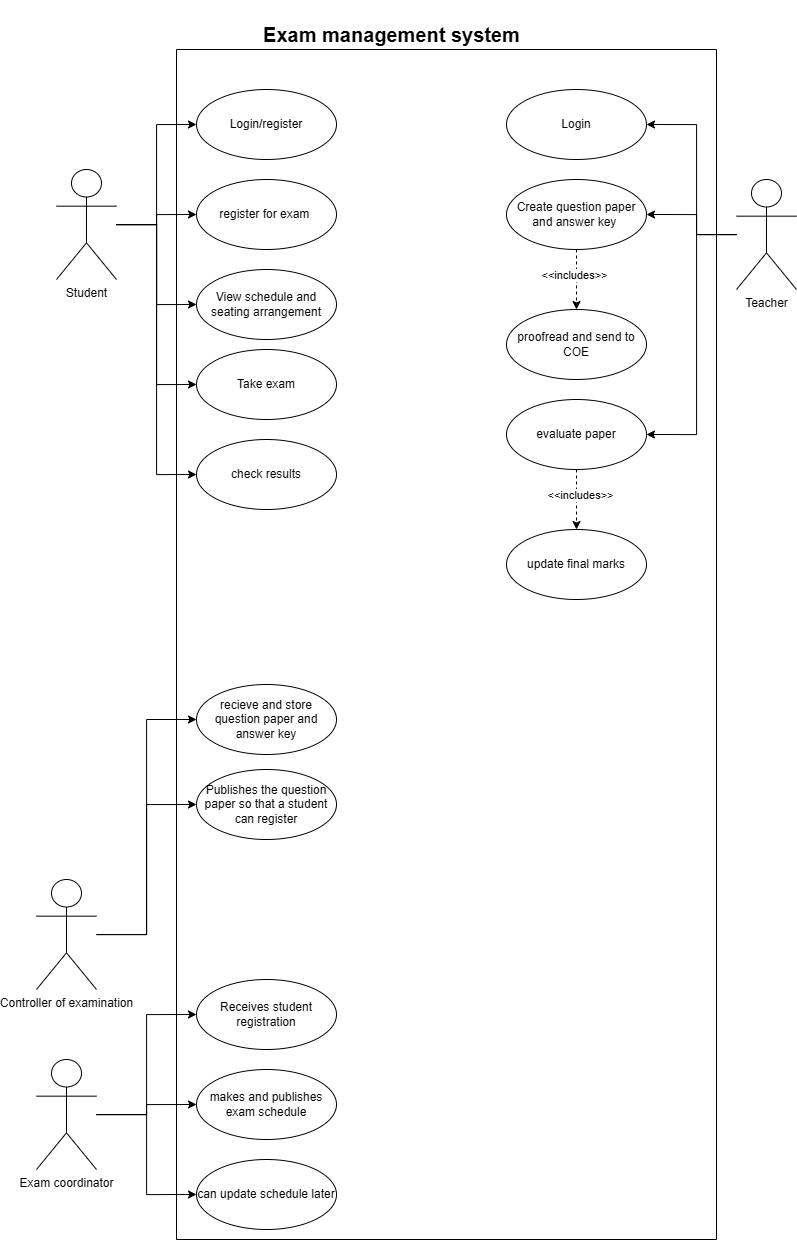
* Automatic grading for multiple-choice (MCQ) exams
* Determination of Pass/Fail status
* Detailed result tracking
* Controlled result publication system

1. Administrative Features

* Controller of examination can configure exams as online or offline. Students can register only when exams are online.
* Exam paper approval workflow
* Management of exam scheduling done by Exam coordinator

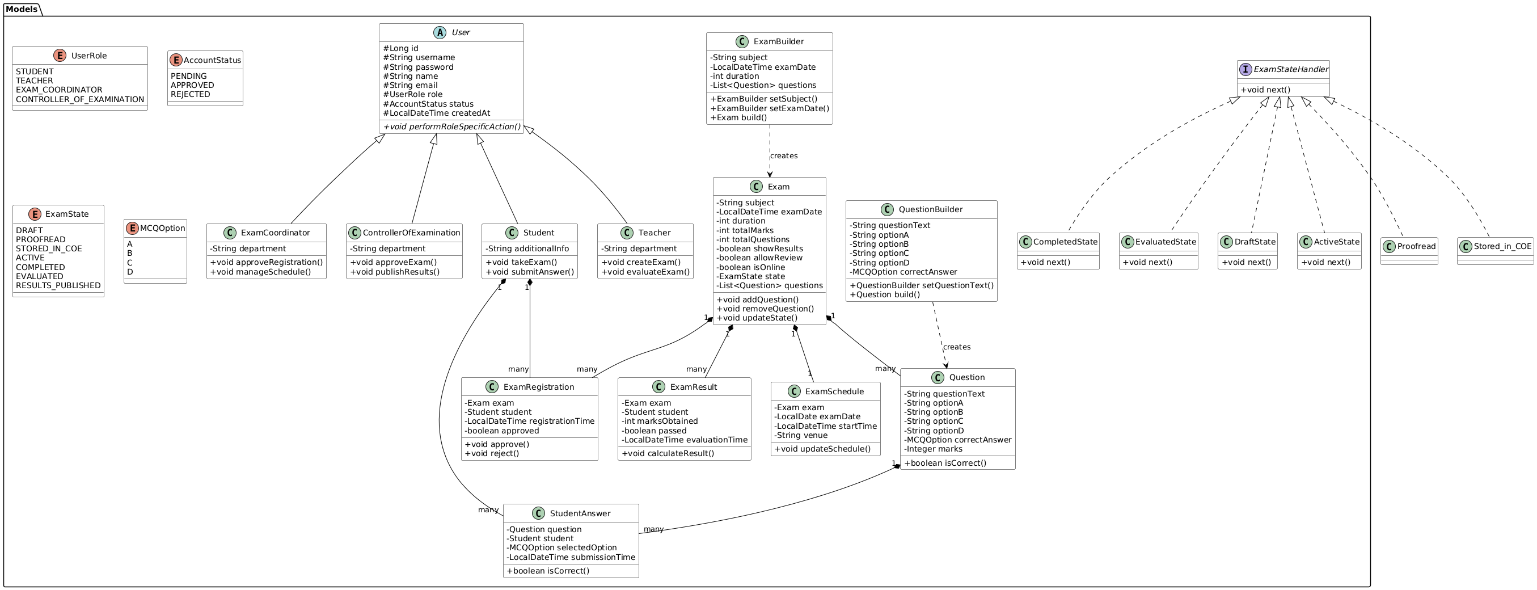
Models:

Use Case Diagram:

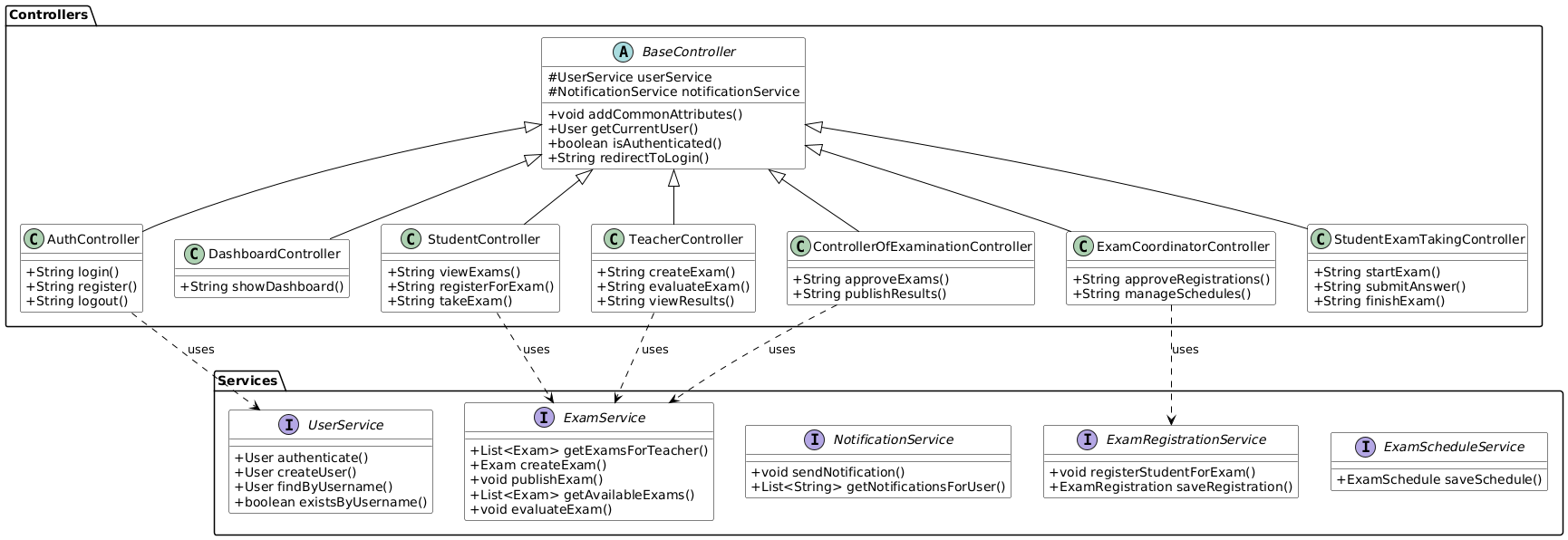


**Class Diagram in MVC:**

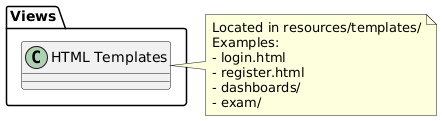
**Model:**



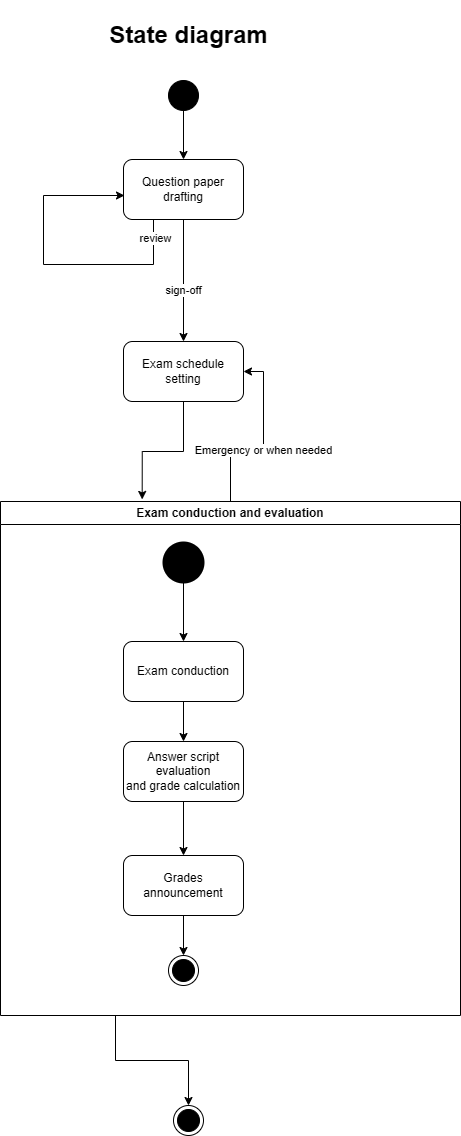
**Controller:**



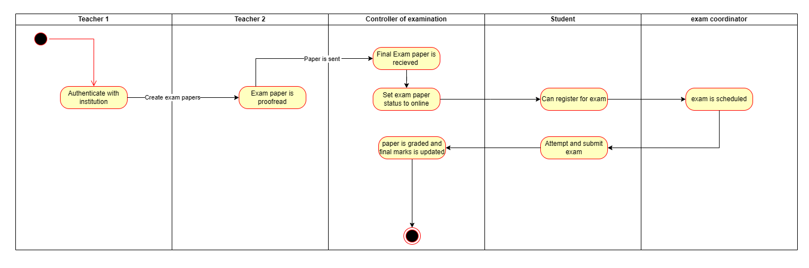
**View:**



**State Diagram:**



**Activity Diagrams:**

****

Architecture Patterns, Design Principles, and Design Patterns:

**Architecture Patterns:**

**1. Model-View-Controller (MVC) Pattern**

* The MVC pattern structures the application into three distinct components, enabling separation of concerns and promoting maintainability.
* **Model**
  + Defines the core business entities and different user types, such as Student, Teacher, and Controller of Examination, Exam coordinator.
  + Uses JPA annotations for ORM and database persistence.
  + Implements Enums to manage application states such as user roles and exam status. Example: ExamState.java, UserRole.java.
  + Applies data validation to enforce business rules.Example: Exam.java includes validation annotations (e.g., @NotNull, @Size).
* **View**
  + Provides dynamic, responsive templates tailored to different user roles.
  + Implements the Thymeleaf templating engine integrated with HTML to render server-side views.
  + Ensures consistent UI/UX through reusable templates and organized view folders.
* **Controller**
  + Manages incoming HTTP requests and user interactions, delegating responsibilities to services.
  + Implements role-based access control and session handling.
  + Coordinates communication between models and views to deliver appropriate data and responses.

**2. Repository Pattern**

* The Repository pattern abstracts and simplifies database interaction by separating persistence logic from business logic.
* **Purpose**
  + Encapsulates all database access, providing a clean API for CRUD operations.
  + Encourages reuse and modularity while supporting complex queries if needed.
* **Implementation**
  + Leverages Spring Data JPA for integration with relational databases.
  + Supports custom queries using derived query methods or @Query annotation.
  + Manages entity relationships and abstracts transaction details.

**3. Service Layer Pattern**

* The Service Layer pattern centralizes business logic and manages transactions, acting as a bridge between controllers and repositories.
* **Purpose**
  + Encapsulates core workflows and business rules.
  + Coordinates logic between different repositories.
  + Handles exception management and ensures data consistency.
* **Implementation**
  + domain-specific services are implemented.
  + Service interfaces are implemented by concrete classes.
  + Uses @Transactional to manage operations that span multiple repositories and ensure atomicity.

**Design Principles:**

1. **SOLID Principles**

* **Single Responsibility Principle (SRP)**
* Each class has one responsibility
* Examples:
* ExamService handles only exam-related operations
* UserService manages only user-related operations
* NotificationService focuses solely on notification handling
* **Open/Closed Principle (OCP)**
* Classes are open for extension but closed for modification
* Examples:
* ExamSession abstract class allows new exam types without modifying existing code
* User base class allows new user types through inheritance
* Service interfaces allow new implementations without changing existing code
* **Liskov Substitution Principle (LSP)**
* Subtypes must be substitutable for their base types
* Examples:
* Student, Teacher, Controller of Examination, Exam coordinator can be used wherever User is expected
* Different exam session types can be used interchangeably
* **Interface Segregation Principle (ISP)**
* Clients should not depend on interfaces they do not use
* Examples:
* Separate interfaces for ExamService, UserService, NotificationService
* Specific interfaces for different functionalities
* **Dependency Inversion Principle (DIP)**
* High-level modules should not depend on low-level modules
* Examples:
* Controllers depend on service interfaces, not implementations
* Services depend on repository interfaces
* Use of @Autowired for dependency injection

1. **DRY (Don't Repeat Yourself)**

* Avoids code duplication
* Examples:
* Common functionality in BaseController
* Reusable service methods
* Shared utility classes

## Design Patterns

1. **Builder Pattern**

* **Purpose**: Constructs complex Exam objects with flexible parameters
* **Implementation**: ExamBuilder class and Question builder class
* **Usage**:
* Can creates Exam objects and question objects with optional parameters
* Handles complex object construction
* Used for exam paper creation.

1. **Factory Pattern**

* **Purpose**: Creates different types of User objects
* **Implementation**: UserFactory class
* **Usage**:
* Creates different user types (Student, Teacher, Controller, Coordinator)
* Centralizes user creation logic
* Handles user type-specific initialization

1. **State Pattern**

* **Purpose**: Manages different states of an exam
* **Implementation**: Exam State Enum and state handlers
* **Usage**:
* Manages exam lifecycle (DRAFT → PROOFREAD →STORED IN COE →ACTIVE → COMPLETED→EVALUATED→RESULTS PUBLISHED)
* Controls state-specific behaviour
* Handles state transitions

1. **Template Method Pattern**

* **Purpose**: Defines the skeleton of exam-taking algorithm
* **Implementation**: ExamSession abstract class
* **Usage**:
* Defines common exam flow
* Allows customization of specific steps
* Ensures consistent exam process

1. **Command Pattern**

* **Purpose**: Encapsulates administrative actions
* **Implementation**: Command classes for administrative operations
* **Usage**:
* Handles exam administrative actions done by the controller of examination
* Supports operations like setting papers online/offline

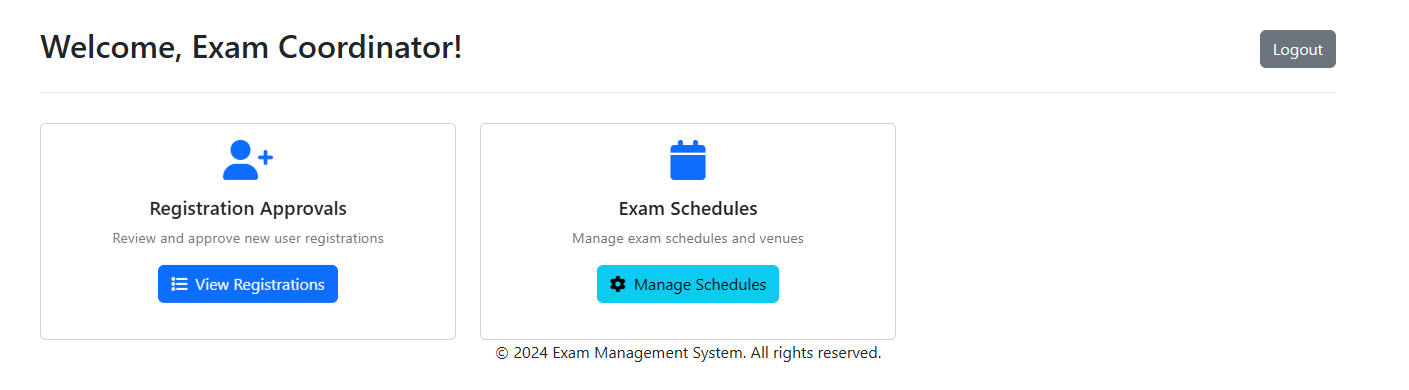
1. **Facade Pattern**

* **Purpose**: Provides simplified interface to complex subsystems
* **Implementation**: ExamRegistrationFacade
* **Usage**:
* Simplifies exam registration process
* Coordinates between multiple services
* Provides clean interface

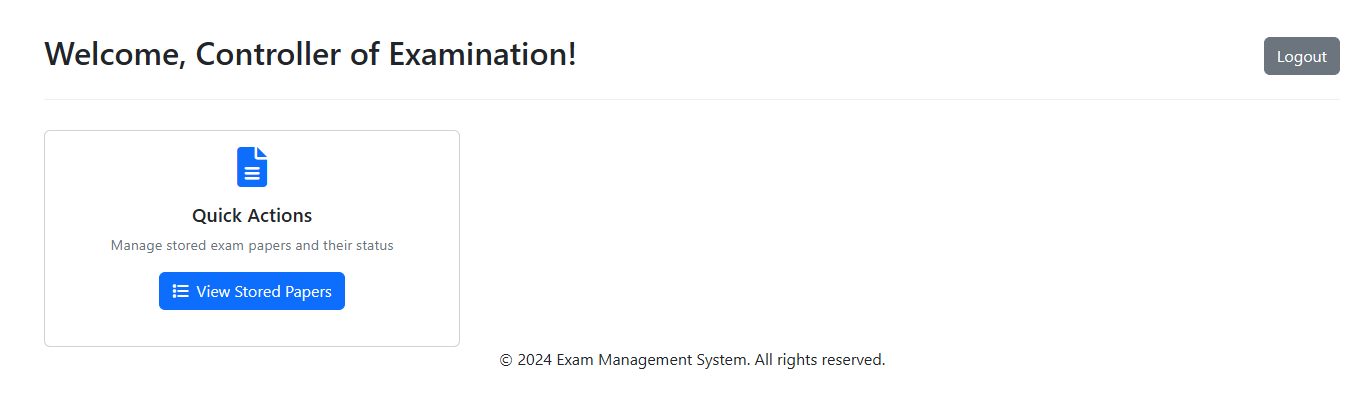
User Interface screenshots and flow of Execution:

Dashboards:

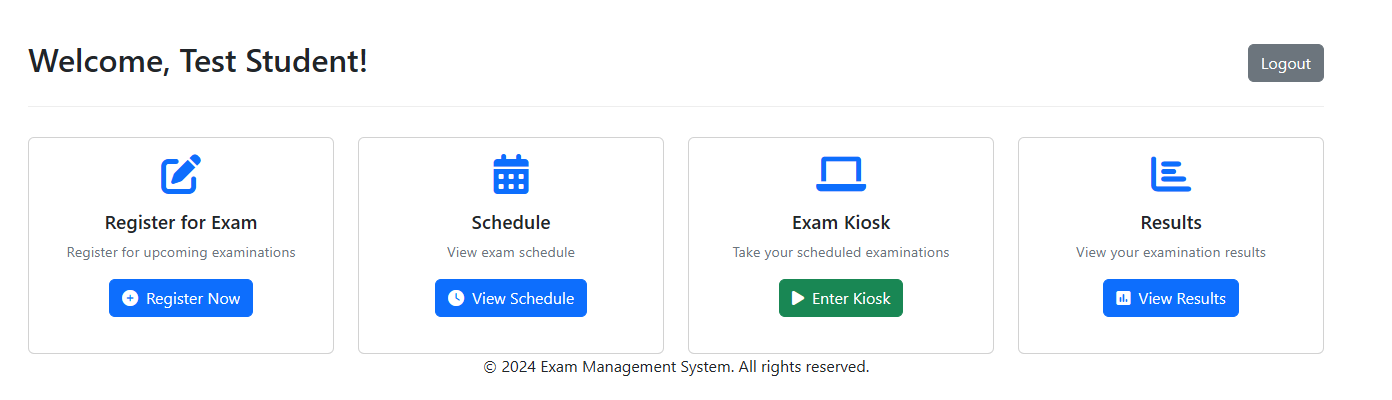
Exam coordinator dashboard:



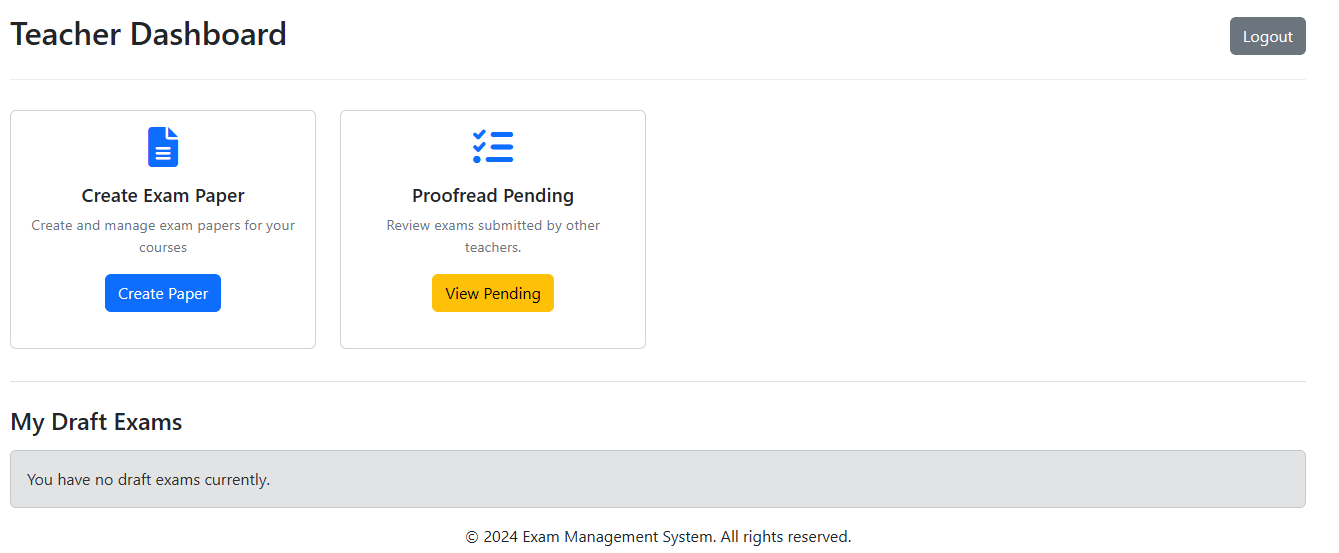
Controller of examination dashboard:



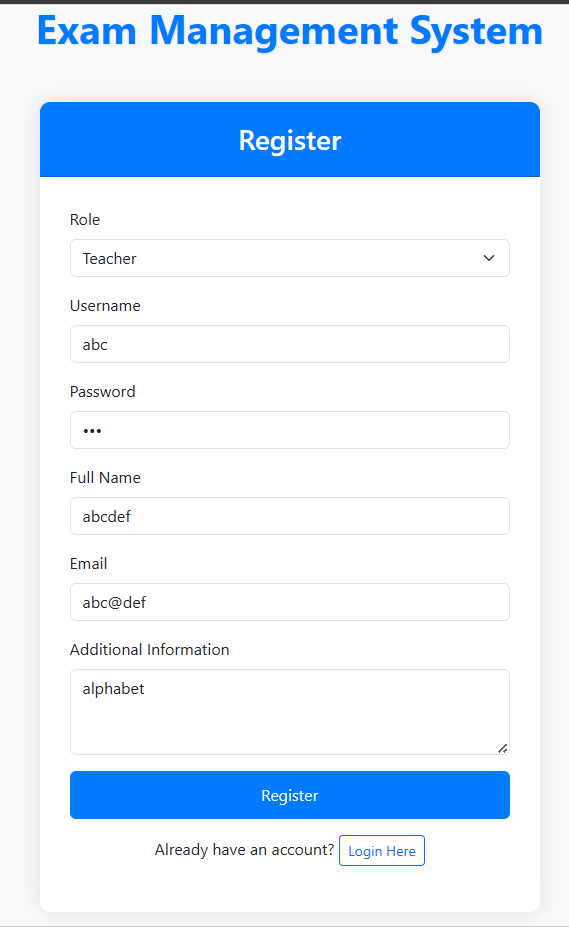
Student Dashboard:



Teacher Dashboard:



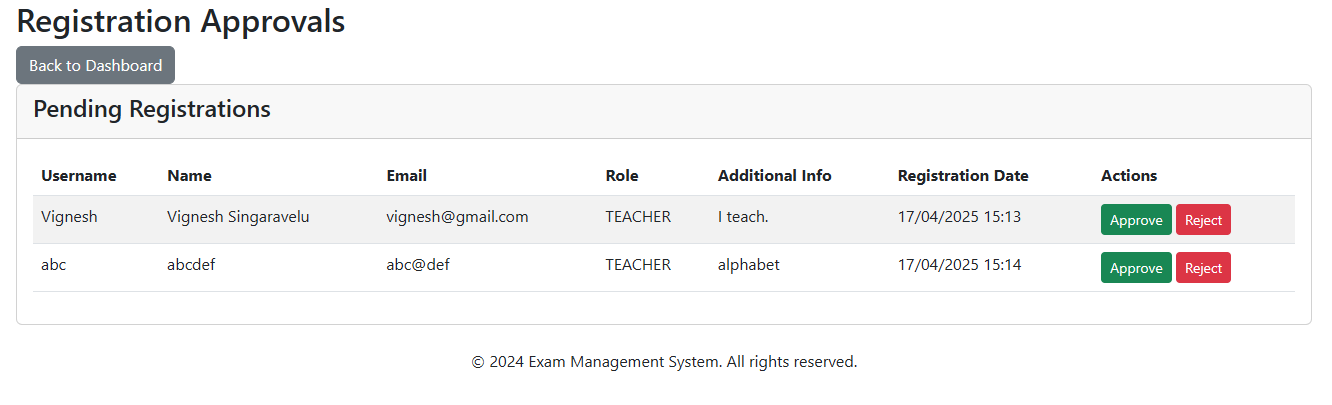
* + 1. Register student users and teacher users



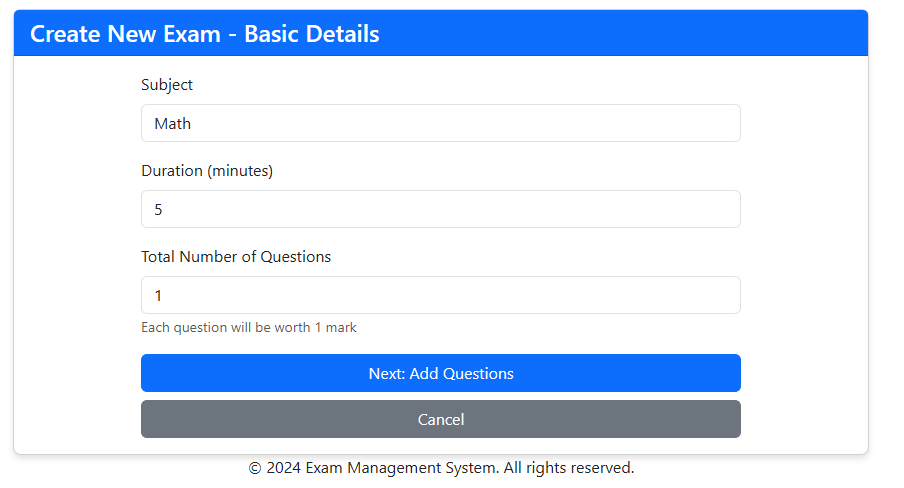
A success prompt is given



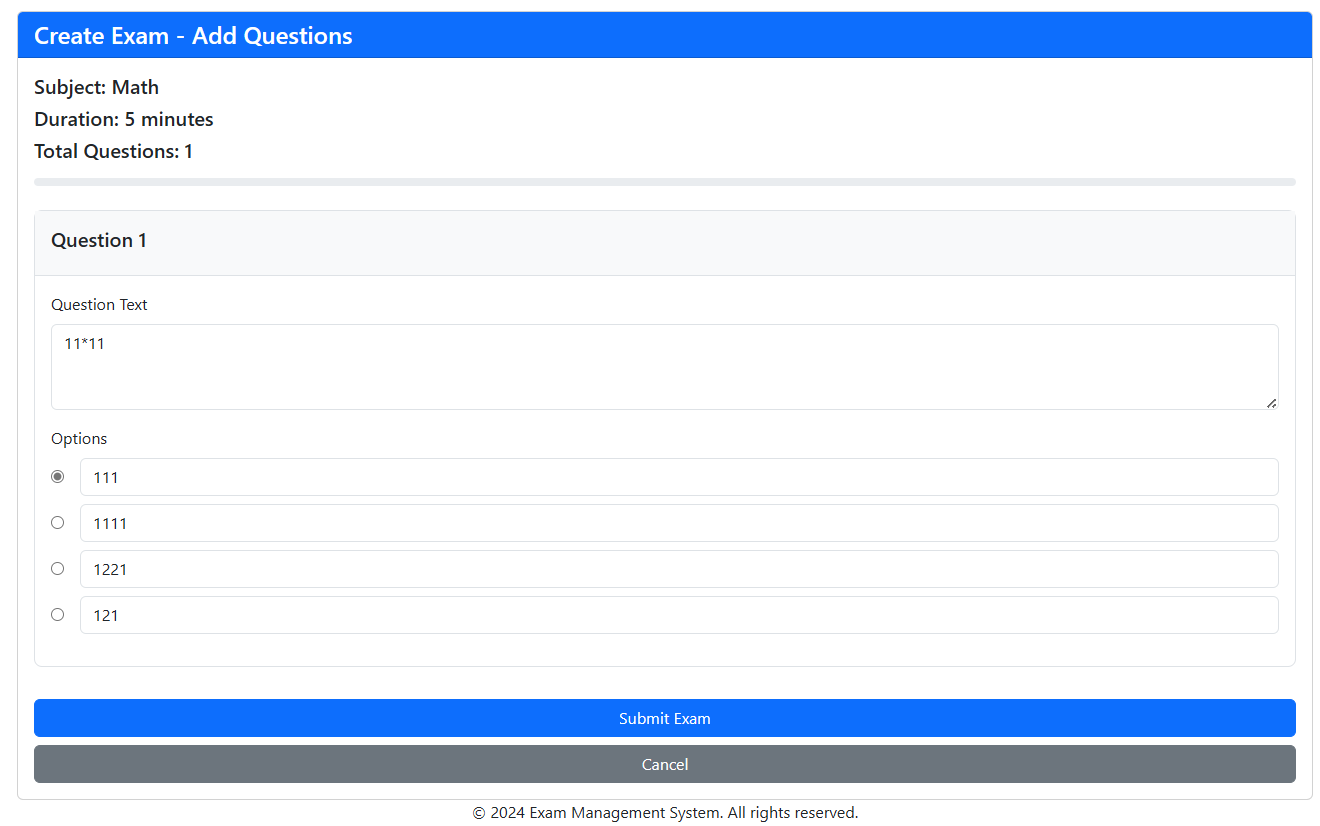
* + 1. Login as Exam coordinator to approve or reject a registration.



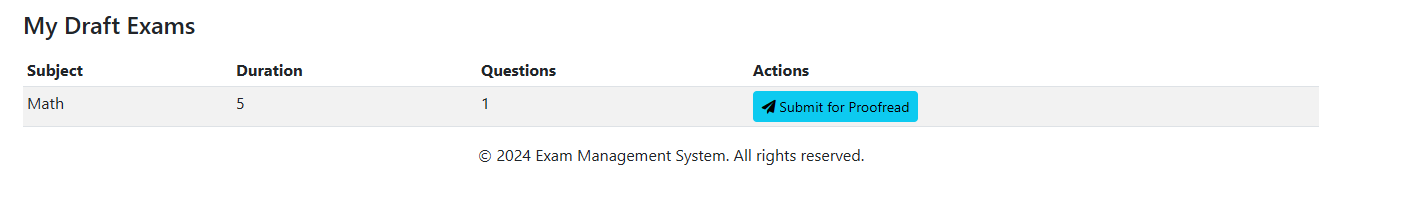
* + 1. **Filling in exam basic details**

****

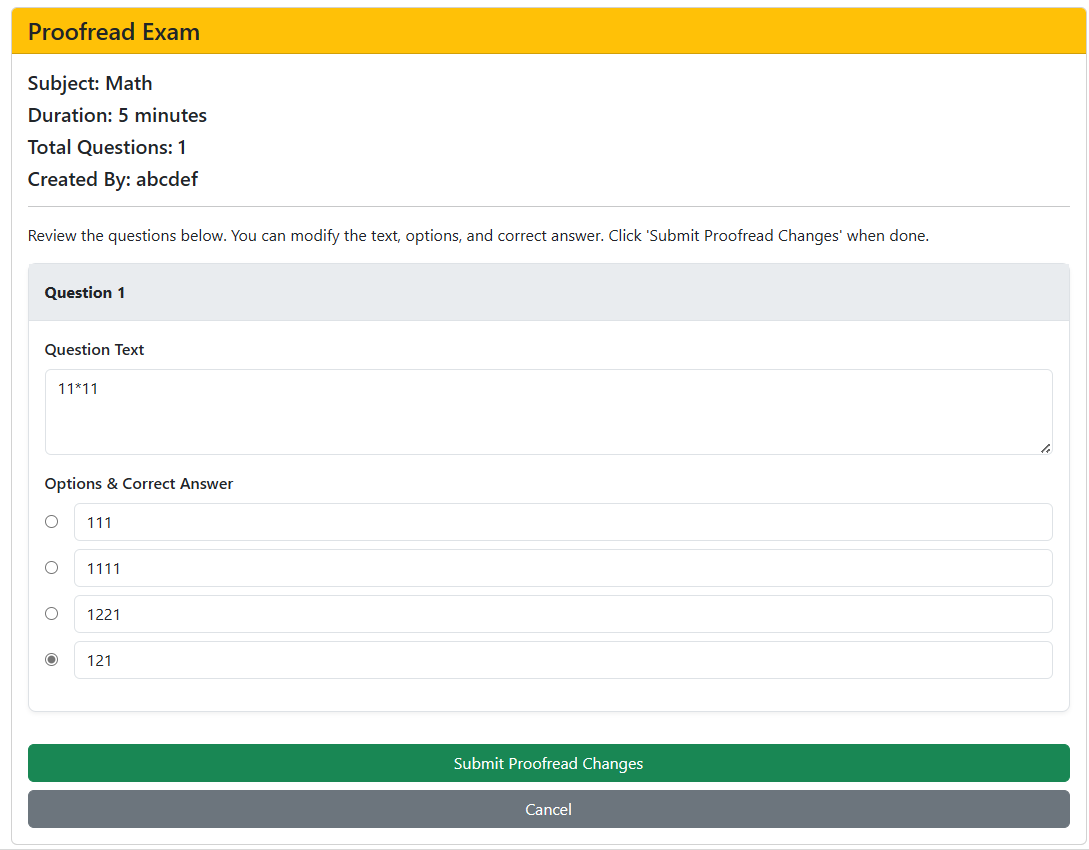
* + 1. **Populating the exam**

****

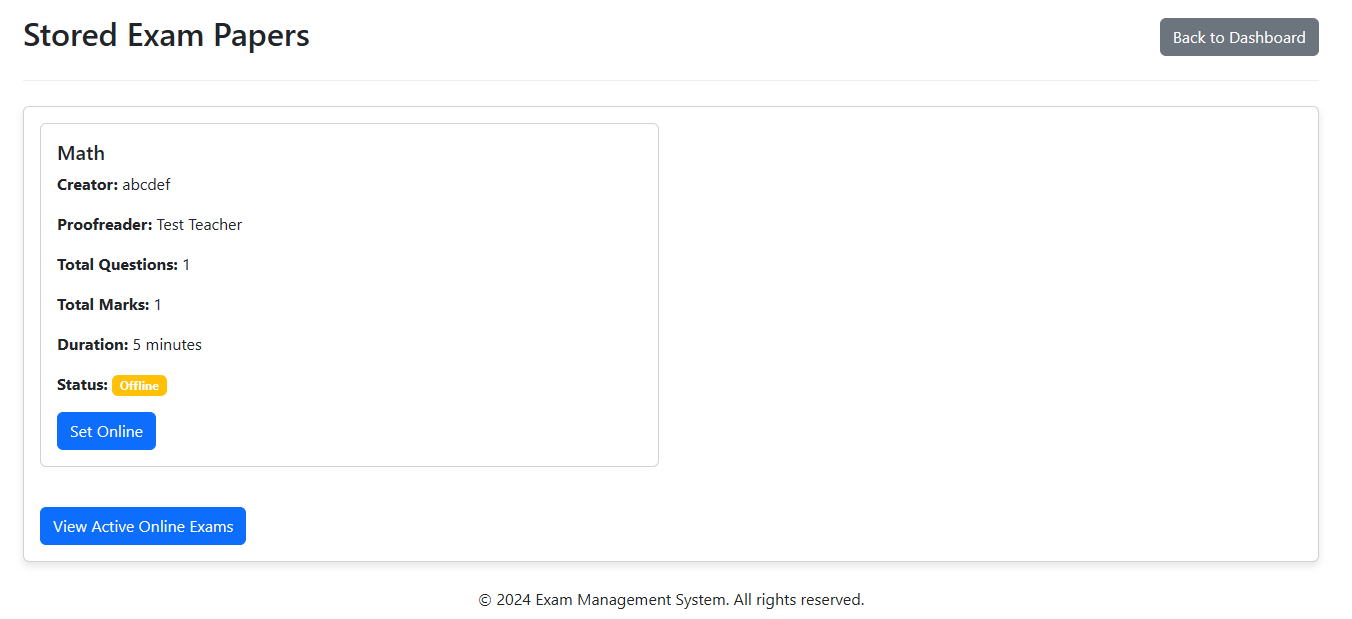
* + 1. **We can now send the draft exam paper to proofreading.**

****

**6. A teacher cannot self-proofread. The proof-read option is available only to other teachers**

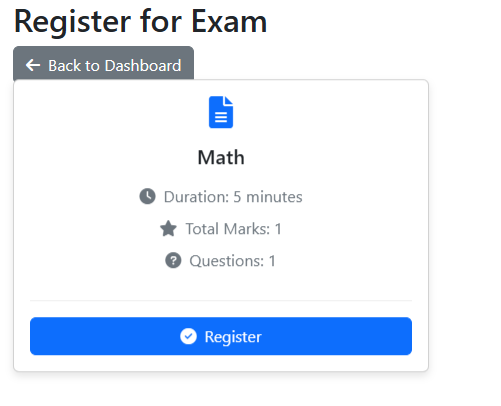
****

**7. An exam after submitting proof-read changes is sent over to the controller of examination to be set as online or offline.**

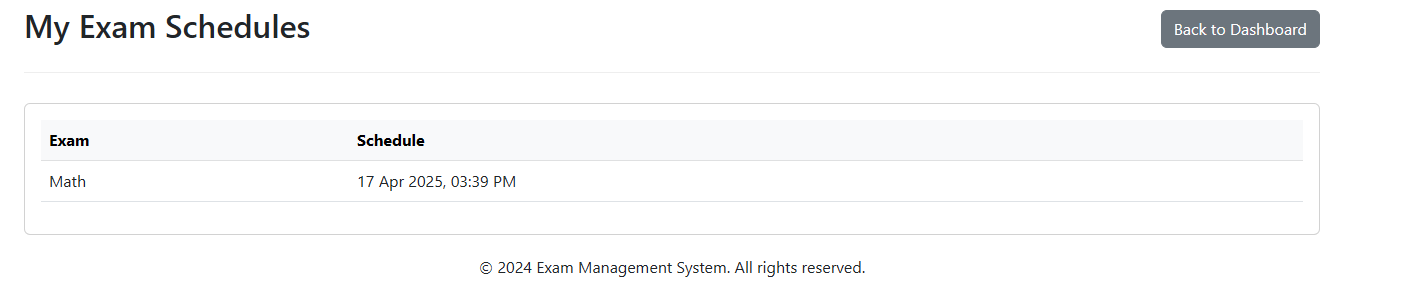
****

**A student can register only to an exam with status online.**

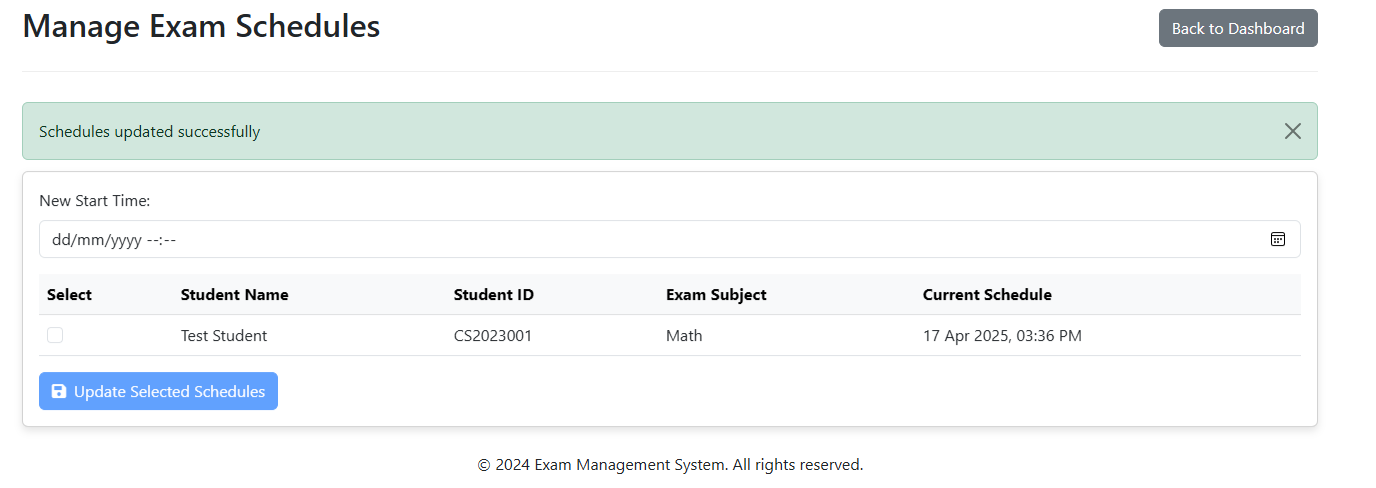
**8. A student can now register to exams**

****

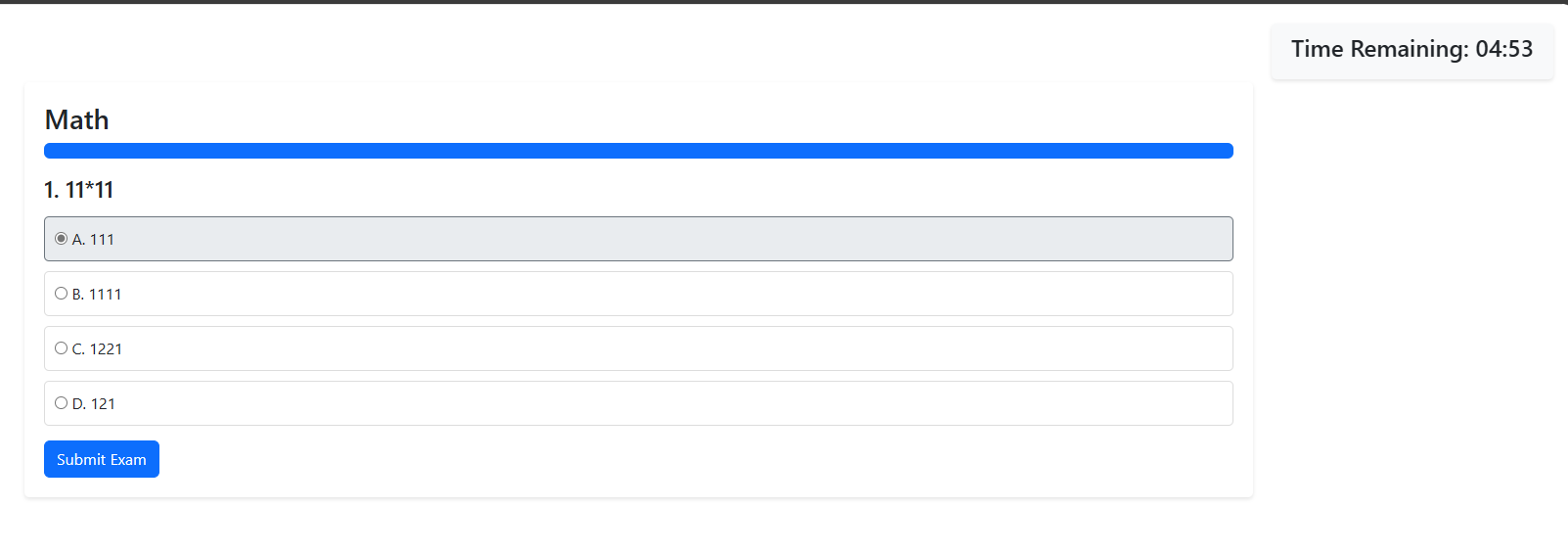
**9.As soon as a student registers for an exam schedule is automatically set.**

****

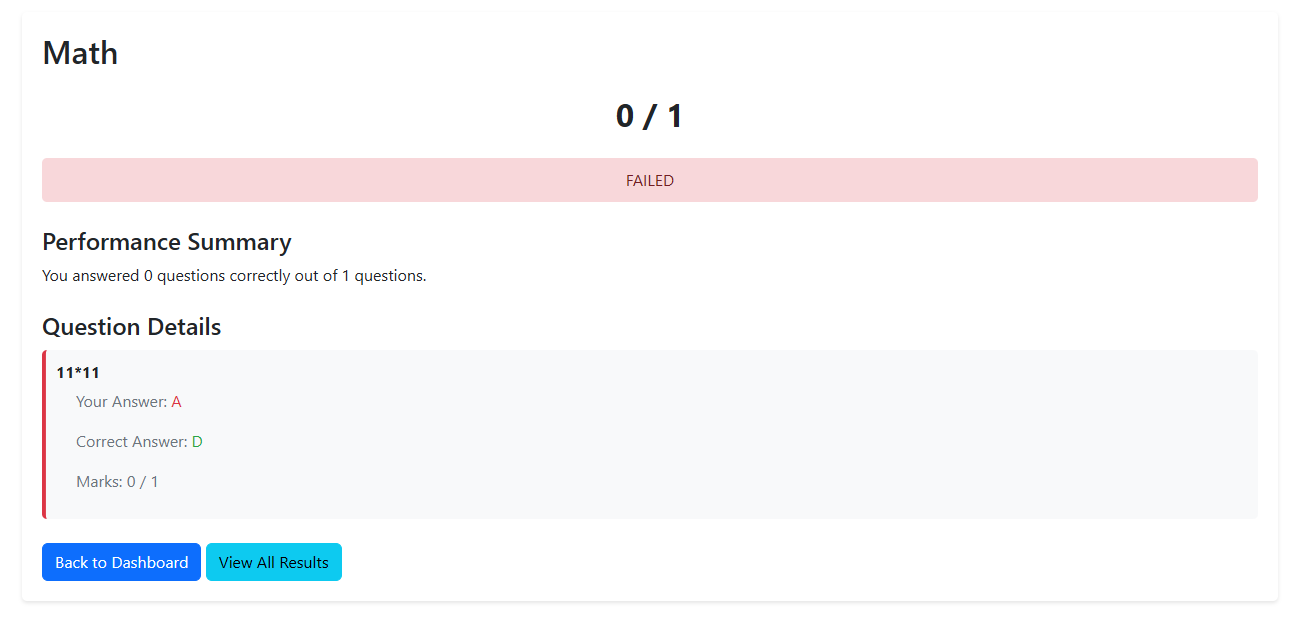
**10. The exam coordinator also has control over updating the schedule**

****

**11. Exam interface:**

****

**12. Result page**

****

**GitHub link to the Codebase** [**here**](https://github.com/usefulldivider/Exam-Management-System-OOAD)

Individual contributions of the team members:

|  |  |
| --- | --- |
| **Name** | **Module worked on** |
| **Teacher** | Vignesh Singaravelu |
| **Student** | Vedanth Padmaraman |
| Exam Coordinator | Vidwan K V |
| Exam Controller | Vishnu Srinivas |