<Group 7>

<GoPass>
Software Architecture Document

Version <1.9>

Revision History

Date	Version	Description	Author
<21/Nov/25>	<1.0>	Initial Document: Established document structure, table of contents	All members
<23/Nov/25>	<1.1>	*Write Introduction	Ho Lam Ngoc Bao
<24/Nov/25>	<1.2>	*Write UI/Pages (class diagram + description): **Huynh Tran Uy:** AuthView **Nguyen Hoang Gia Han:** ExamCreationView **Tran My An:** DashboardView	Huynh Tran Uy Nguyen Hoang Gia Han Tran My An
<25/Nov/25>	<1.3>	*Write Architectural Goals and Constraints: **Pham Bao Kha:** Programming Language & Technology Stack, Platform Support, Schedule Constraint **Ho Lam Ngoc Bao:** Requirements and Constraints	Pham Bao Kha Ho Lam Ngoc Bao
<27/Nov/25>	<1.4>	Verify and update Architectural Goals and Constraints	All members
<28/Nov/25>	<1.5>	*Write UI/Pages (class diagram + description): **Nguyen Hoang Gia Han:** ContestCreationView, SubmissionListView **Tran My An:** ExamContestListView, QuizStimulationView, ContestDetailView	Nguyen Hoang Gia Han Tran My An
<29/Nov/25>	<1.6>	*Write UI/Pages (class diagram + description): **Tran My An:** ClassManagementView, ClassDetailView, UserInfo **Ho Lam Ngoc Bao & Pham Bao Kha:** High level of Logical View *Write Server-side: **Ho Lam Ngoc Bao:** Auth, User, Grading, Contest, Admin **Pham Bao Kha:** Class, Exam, Submission, QuestionBank ***Huynh Tran Uy:** Verify Server-side	All members
<01/Dec/25>	<1.6>	*Write UI/Pages (class diagram + description): **Huynh Tran Uy:** ProgressDashboard	Huynh Tran Uy Tran My An

		*Write Client Logic & State: **Huynh Tran Uy:** Source Code Organization, Validation **Tran My An:** Hooks **Nguyen Hoang Gia Han**: Utilities, Services	Ho Lam Ngoc Bao
<02/Dec/25>	<1.7>	*Write database: **Pham Bao Kha:** User & Class Management **Ho Lam Ngoc Bao:** Exam, Question, Submission & Grading, Contest & Exam	Pham Bao Kha Ho Lam Ngoc Bao
<03/Dec/25>	<1.8>	Verify and update Logical View	All members
<04/Dec/25>	<1.9>	Add final version of Use-case Model Reformat the lines, bullet points, and alignment of paragraphs in the document	Huynh Tran Uy Tran My An

Table of Contents

Software Architecture Document

1. Introduction

The Software Architecture Document (SAD) provides a comprehensive architectural overview for the *GoPass* online examination and learning platform. This document outlines the purpose, scope, architectural principles, terminology, and references that guide the design, development, and deployment of the GoPass system. Its objective is to ensure that all stakeholders including architects, developers, testers, and managers share a unified understanding of the system structure and behavior.

1.1. Purpose

The primary purpose of this Software Architecture Document is to articulate the architectural vision, key design decisions, and guiding principles that drive the development of the GoPass platform. It documents the architectural structures governing both the client and server components, the interactions between modules, and the rationale behind selected architectural patterns. This document also highlights the architectural trade-offs, constraints, and quality attributes that influence the system's design, ensuring that the GoPass platform meets its functional and non-functional requirements such as scalability, maintainability, reliability, and security

1.2. Scope

The scope of this document encompasses the entire GoPass platform, including its web-based Single Page Application (SPA), backend RESTful API services, database models, and external integrations. This SAD presents the high-level architecture, the system's major components, and how they collaborate to deliver examination, grading, and analytics functionalities.

Throughout this document, the GoPass platform is described from multiple architectural viewpoints:

- General Overview View presenting the major subsystems and execution flow of GoPass.
- Logical View detailing layers, modules, controllers, services, and core data models.
- Process / Runtime View showing component interactions and sequence diagrams for key use cases.
- Implementation View describing technologies, libraries, coding conventions, and project structure used in the system.
- Deployment View illustrating physical deployment, including the client-server model, hosting platform, and external dependencies.
- Interface View documenting REST API endpoints, validation rules, and module communication boundaries.

1.3. Definitions, Acronyms, and Abbreviations

Table 1-1: Glossary of Terms

Terms	Definition
REST API	Architectural style used by GoPass backend for stateless client–server communication.
JWT	JSON Web Token used for authentication and access control.
SPA	Single Page Application architecture used to build the GoPass frontend using React.
Controller	Backend component responsible for processing HTTP requests and orchestrating business logic.
Service Layer	Backend layer encapsulating business operations,

	external integrations, and processing rules.
Submission	Entity representing a student's answers and exam attempt status.
Question Bank	Centralized repository of reusable questions used to generate exams.
Autosave	Mechanism to periodically save student answers during an exam attempt.
AI Grading	Automated scoring mechanism for essay questions using external or internal AI scoring modules.
Aggregation	Analytics mechanism used to compute scores, metrics, and dashboard summaries.
Token Refresh	Security mechanism for issuing renewed access tokens.
Feedback	Comments or scores returned to students after grading.

Table 1-2: Glossary of Acronyms

Abbreviation/Acronym	**Definition**
API	Application Programming Interface
DBMS	Database Management System
CS	Client-side
SS	Server-side
RBAC	Role-Based Access Control
NFR	Non-Functional Requirements
UI	User Interface
URL	Uniform Resource Locator
HTTP	Hypertext Transfer Protocol
TTL	Time-To-Live (e.g., token/session expiration)
SPA	Single Page Application
AI	Artificial Intelligence

1.4. References

The following documents and standards served as references during the creation of this Software Architecture

Document:

- GoPass Vision Document
- GoPass Use Case Model
- IEEE 42010 — Systems and Software Engineering: Architecture Description
- React technical documentation
- NodeJS technical documentation
- MongoDB schema design guidelines
- REST API standards
- JWT authentication standards

2. Architectural Goals and Constraints

2.1. Programming Language & Technology Stack

- Frontend: Implemented using ReactJS (Single Page Application) and Styled with Tailwind CSS for fast and consistent UI development.
- Backend: Developed using NodeJS with Express, following a layered architecture (Controller → Service → Repository → Model).
- Database: MongoDB serves as the primary database, storing users, classes, exams, questions, and submissions.

2.2. Platform Support

The system currently supports web browser platforms only, optimized for desktops and laptops.

2.3. Schedule Constraint

The complete GoPass system must be delivered within **2 months**.
Development will follow multiple internal milestones such as:

- Core authentication & class management
- Exam-taking module
- Question bank
- Submissions & scoring
- Analytics dashboard

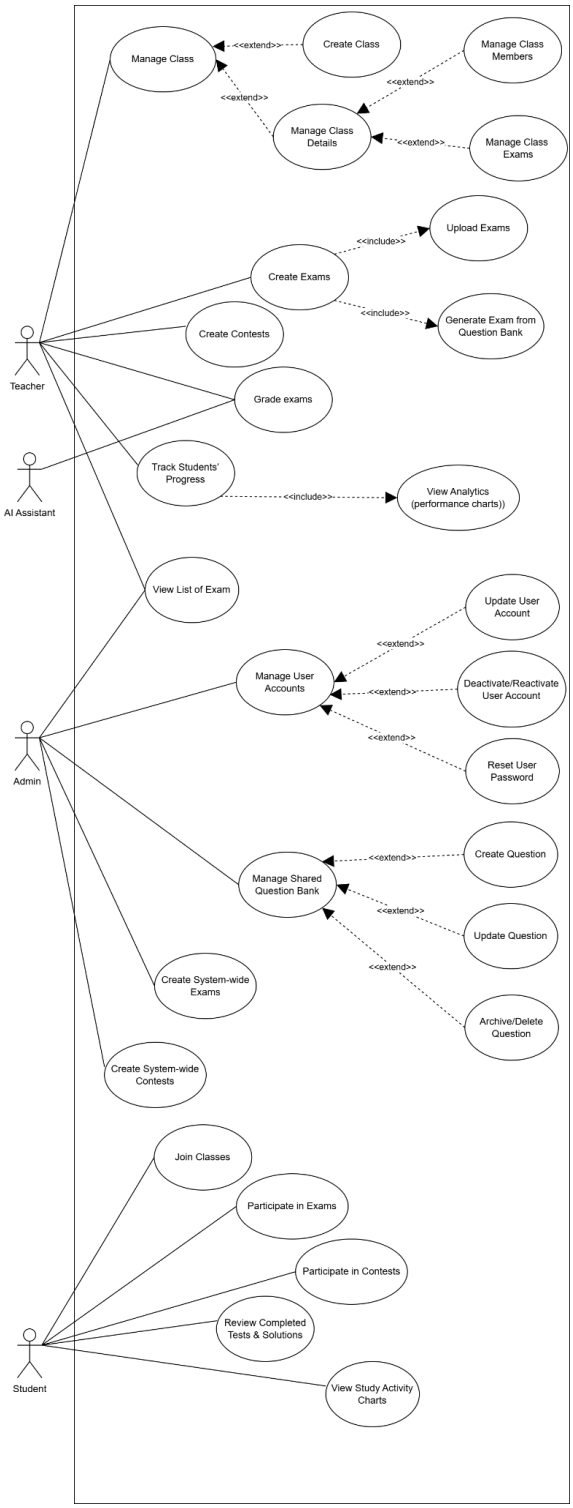
2.4. Requirements and Constraints

Table 2-1: Non-Functional Requirements & Constraints

Category	Requirement / Constraint	Description
Performance	Response Time	General operations (loading exams, saving answers, navigation) must respond in < 2–3 seconds under normal load.
	Content Loading	Exam list, dashboard, and class pages should load in ≤ 4 seconds even with large datasets.
	Auto-grading Speed	Automatic scoring for multiple-choice questions must complete within ≤ 5 seconds after submission.
	Scalability	System must support up to 1000 concurrent users (many classes taking exams simultaneously) without performance degradation.
Usability	User Interface	UI must be intuitive and easy to use for both students and teachers, with clear navigation of core functions.

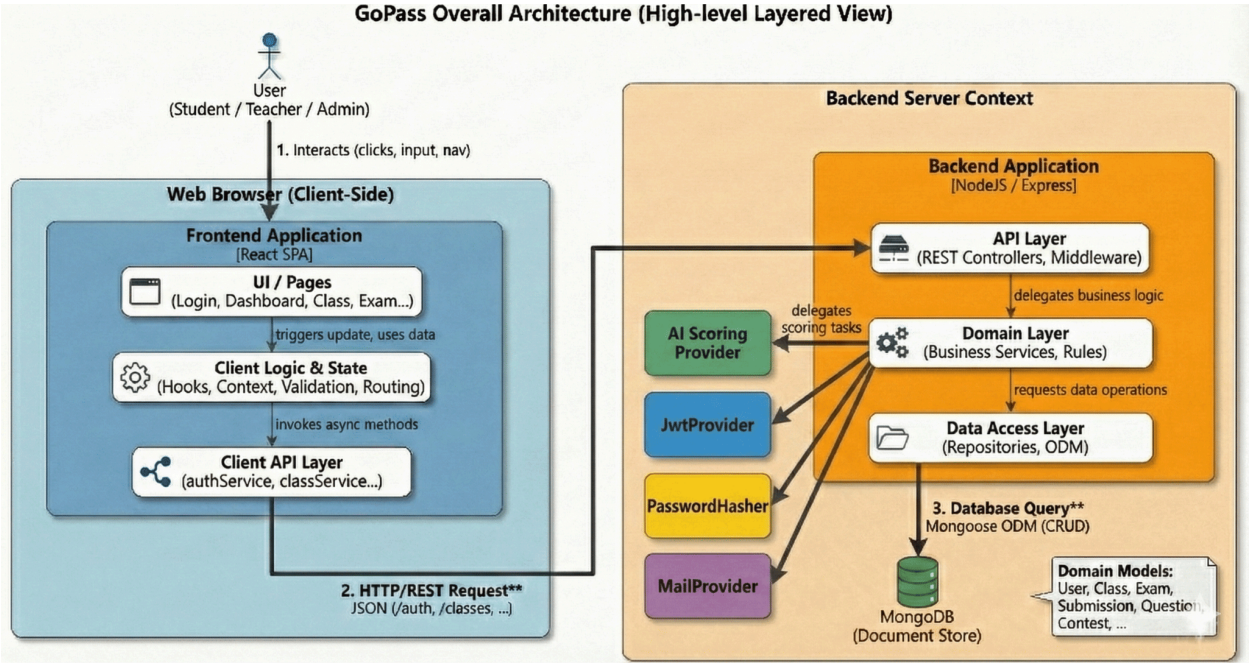
	Responsiveness	Web interfaces should be responsive across desktops, laptops (browser-based).
Maintainability	Code Quality	Code must follow standard coding conventions, include comments where necessary, and be consistently formatted.
	Modularity	System architecture must be modular (Controller–Service–Repository-Model + React Components) to support easy updates and new features.
	Documentation	Technical documents (API references, data models, deployment guide) must be available to support developer onboarding and maintenance.
Security	Data Encryption	All data must be encrypted in transit (HTTPS) and passwords hashed at rest (bcrypt).
	Authentication	Secure authentication using JWT; Admin accounts may optionally support MFA in the future.
	Access Control	Enforce Role-Based Access Control (RBAC) for Students, Teachers, and Admins.

3. Use-Case Model



4. Logical View

4.1 High-level design



The Architecture Design includes 2 parts:

- **The client-side** is a React Single Page Application (SPA) structured into three main layers:
  - Presentation Layer (UI / Pages): Renders the user interface (Login, Dashboard, Class, Exam, Contest) and handles user interactions such as clicks, form inputs, and navigation.
  - Client Logic & State Layer: Implements application logic using React Hooks, Context, and routing. Handles validation, client-side state, exam session logic (answers, timer, autosave), and prepares data for the UI.
  - Client API Layer: Encapsulates communication with the backend through service modules (e.g., authService, classService, examService). Sends REST requests and returns processed responses to the logic layer.

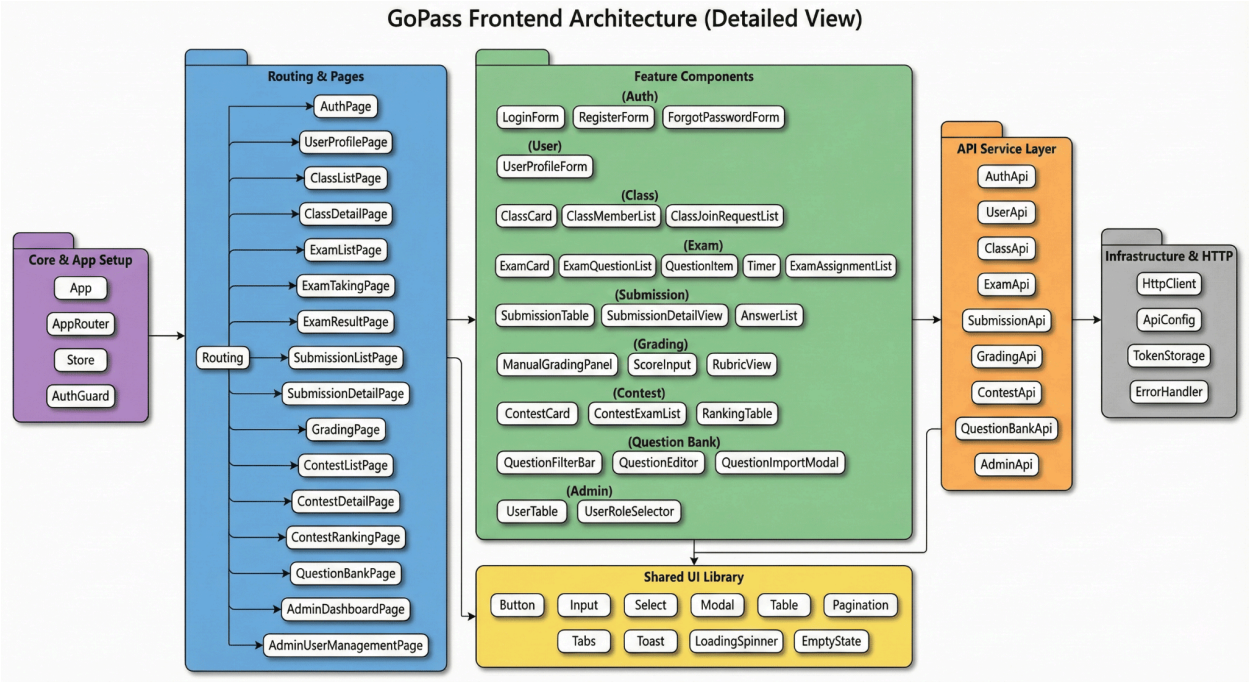


Figure 4.1.1: Overview of client-side

- **The server-side** is a NodeJS/Express application following a clear layered structure:
  - API Layer (REST Controllers, Middleware): Handles HTTP routing, authentication, validation, and delegates use-case execution to the domain layer.
  - Domain Layer (Business Services, Rules): Implements all core business logic: class management, exams, submissions, contest handling, question bank, admin actions, and analytics. This layer also communicates with the AI Scoring Provider for essay grading.
  - Data Access Layer (Repositories, ODM): Provides a clean abstraction over MongoDB through repository modules. Handles CRUD operations for all domain models (User, Class, Exam, Submission, Question, Contest, etc.).
  - AI Scoring Provider: Used by the GradingService to process essay-type answers and generate AI-based scoring suggestions and feedback.
  - Database Layer (MongoDB Domain Models): Stores all domain entities and is accessed exclusively through the repository layer.

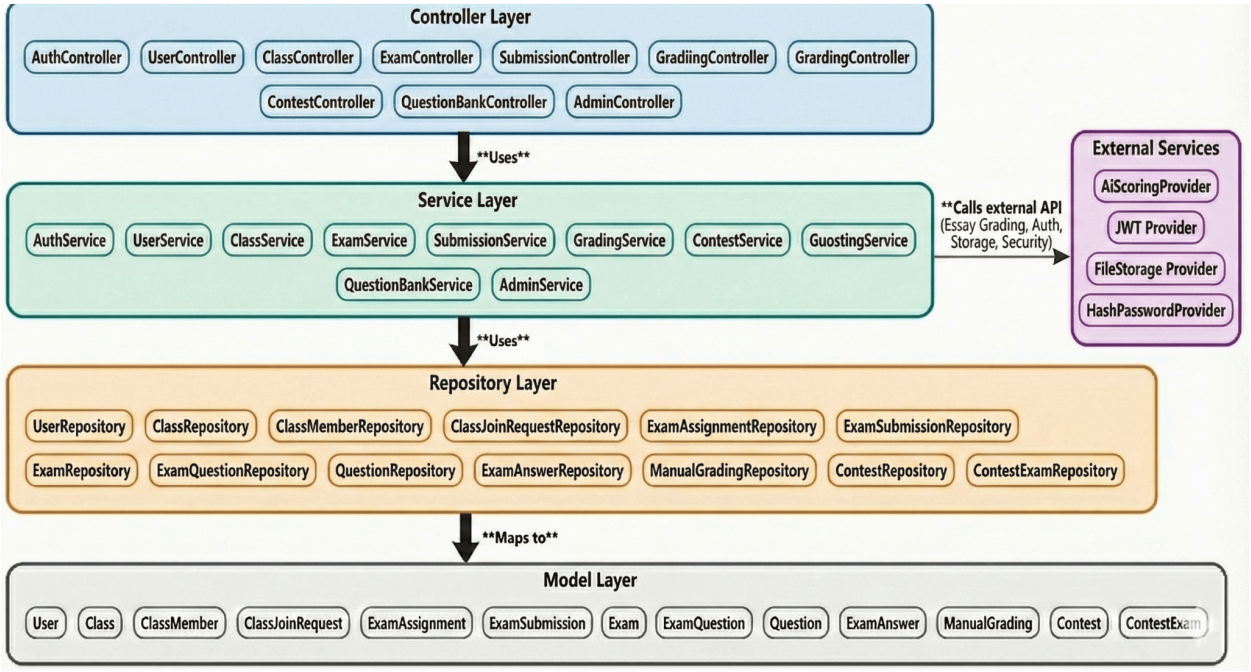


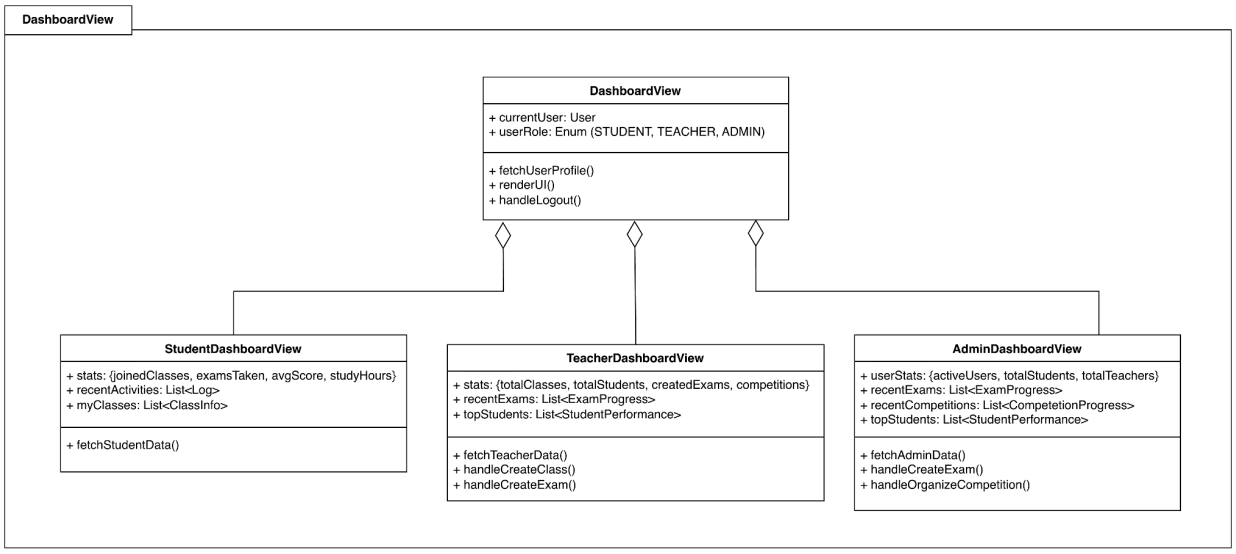
Fig 4.1.2: Overview of server-side

4.2 Low-level design

4.2.1 Client-side

4.2.1.1 UI/Pages

4.2.1.1.1 Dash Board

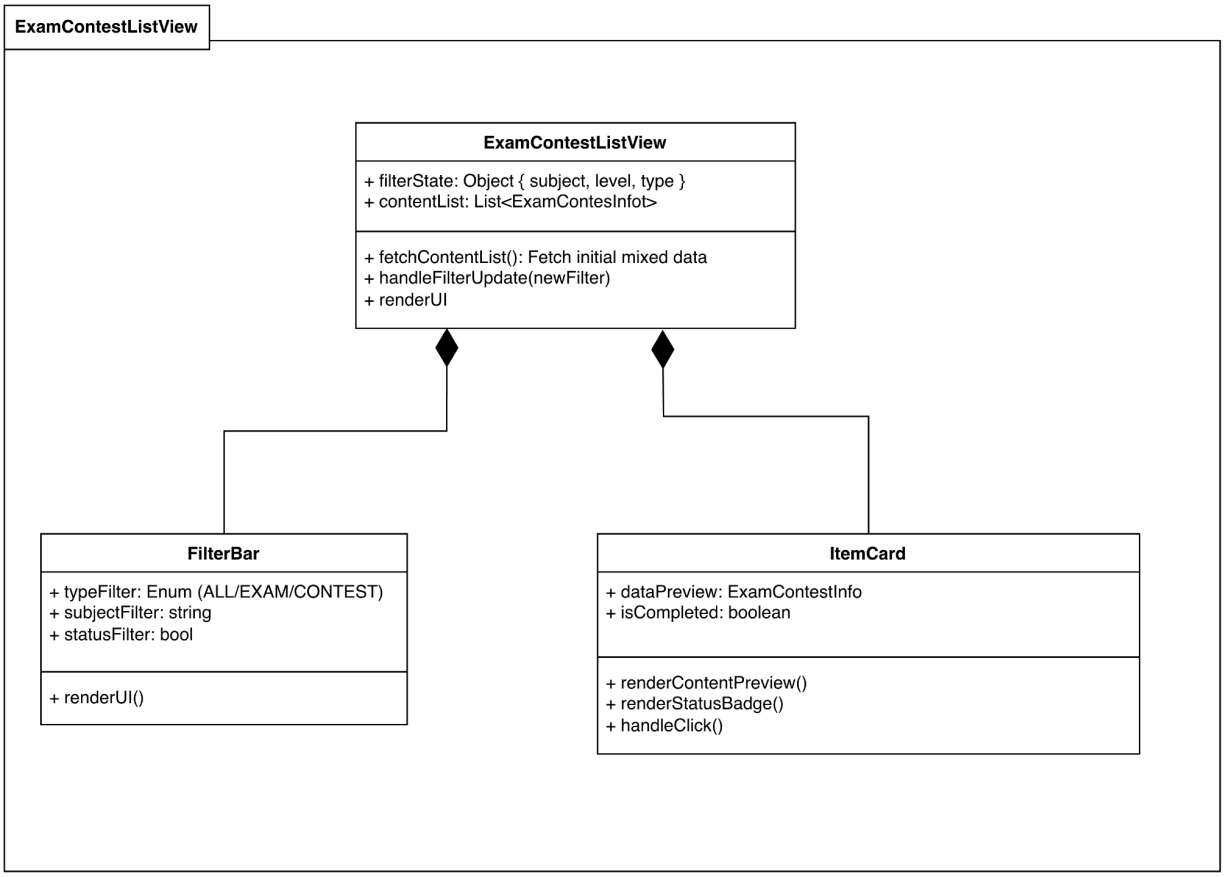


Description: This module manages the primary landing interface post-authentication, utilizing a strategy pattern to dynamically render specific dashboard layouts based on the authenticated user's role (Student, Teacher, or Admin). It

acts as the central hub for data aggregation, providing users with a personalized overview of their statistics and quick access to role-specific actions.
Explain:

- + fetchUserProfile(): Retrieves the current user's identity and role to determine which dashboard view to initialize.
- + renderUI(): Mounts the appropriate child component (Student, Teacher, or Admin view) into the main container based on the user's role.
- + handleLogout(): Clears the current user session and redirects the application to the login screen.
- + fetchStudentData(): Aggregates personal learning statistics, recent activity logs, and enrolled classes for the student interface.
- + fetchTeacherData(): Retrieves management metrics, created exam lists, and top student performance reports for the teacher interface.
- + fetchAdminData(): Collects system-wide health statistics, user counts, and engagement metrics for the administrator.
- + handleCreateClass(): Initiates the workflow for a teacher to create a new classroom.
- + handleCreateExam(): Navigates to the exam creation wizard
- + handleCreateContest(): Navigates to the contest creation wizard.

4.2.1.1.2 Quiz List

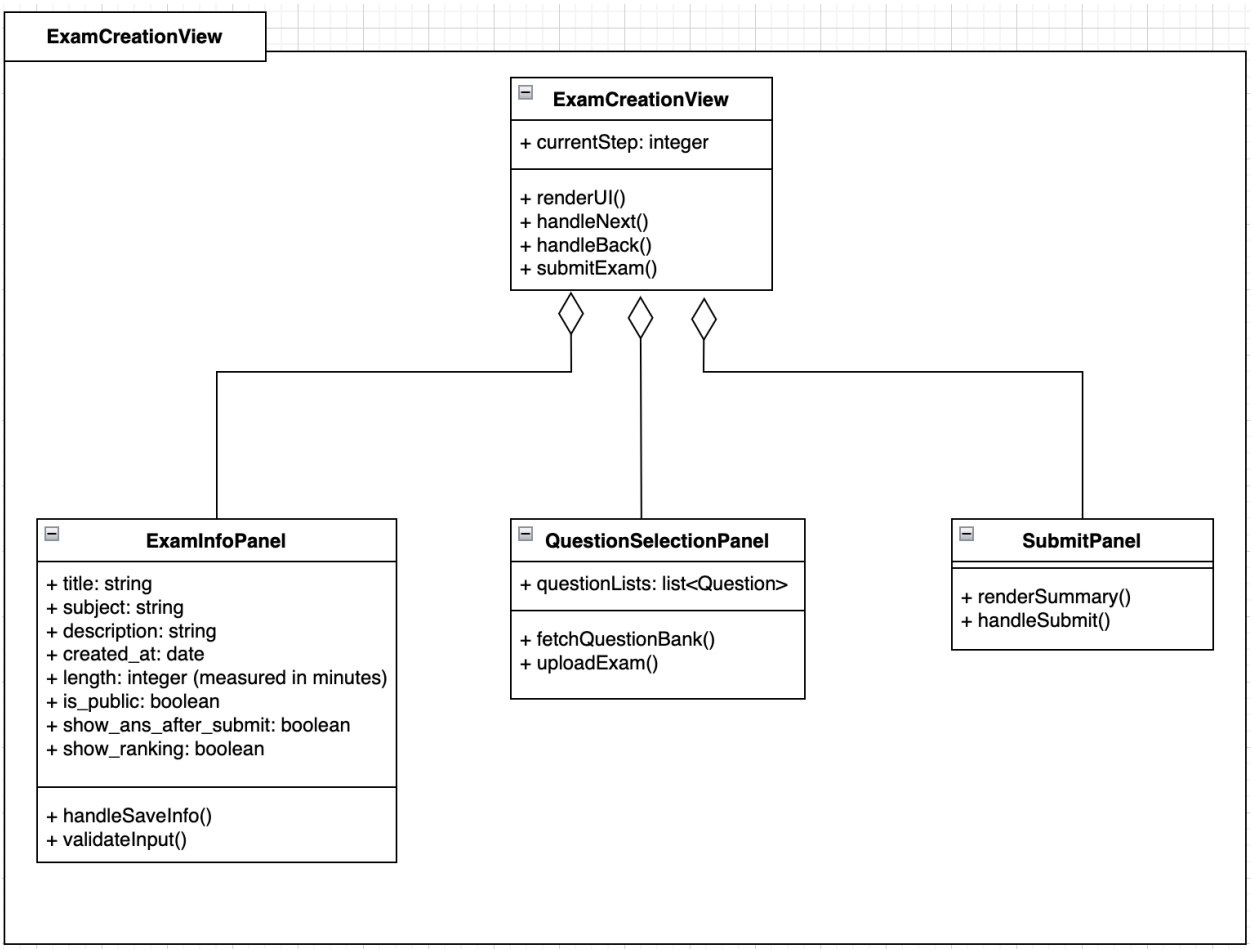


Description: This module serves as the central catalog interface, allowing users to explore and access learning materials. It integrates a unified feed that aggregates both practice exams and scheduled contests into a single view. The module enables users to sort content by type, subject, or completion status and navigate to the appropriate assessment environment by using the filter system.

Explain:

1. Class: ExamContestListView
    + fetchContentList(): Initiates an API call to retrieve the initial dataset containing both exams and contests. It handles data merging and pagination logic.
    + handleFilterUpdate(newFilter): Acts as the callback function when users change criteria in the FilterBar. It updates the internal filterState and triggers a refresh of the contentList.
    + renderUI(): Manages the overall layout, positioning the filter panel alongside or above the content grid and iterating through the data to render the list.
2. Class: FilterBar
    + renderUI(): Dynamically renders the input controls (dropdowns) based on the attributes typeFilter, subjectFilter, and statusFilter, allowing users to input their preferences.
3. Class: ItemCard
    + renderContentPreview(): Displays the core information of a specific item (title, tags). It contains logic to render details based on the item type conditionally.
    + renderStatusBadge(): Visualizes the user's interaction history with the item, such as displaying a "Completed" checkmark or a "New" label based on the isCompleted attribute.
    + handleClick(): Handles the navigation event.

4.2.1.1.3 Exam Creation



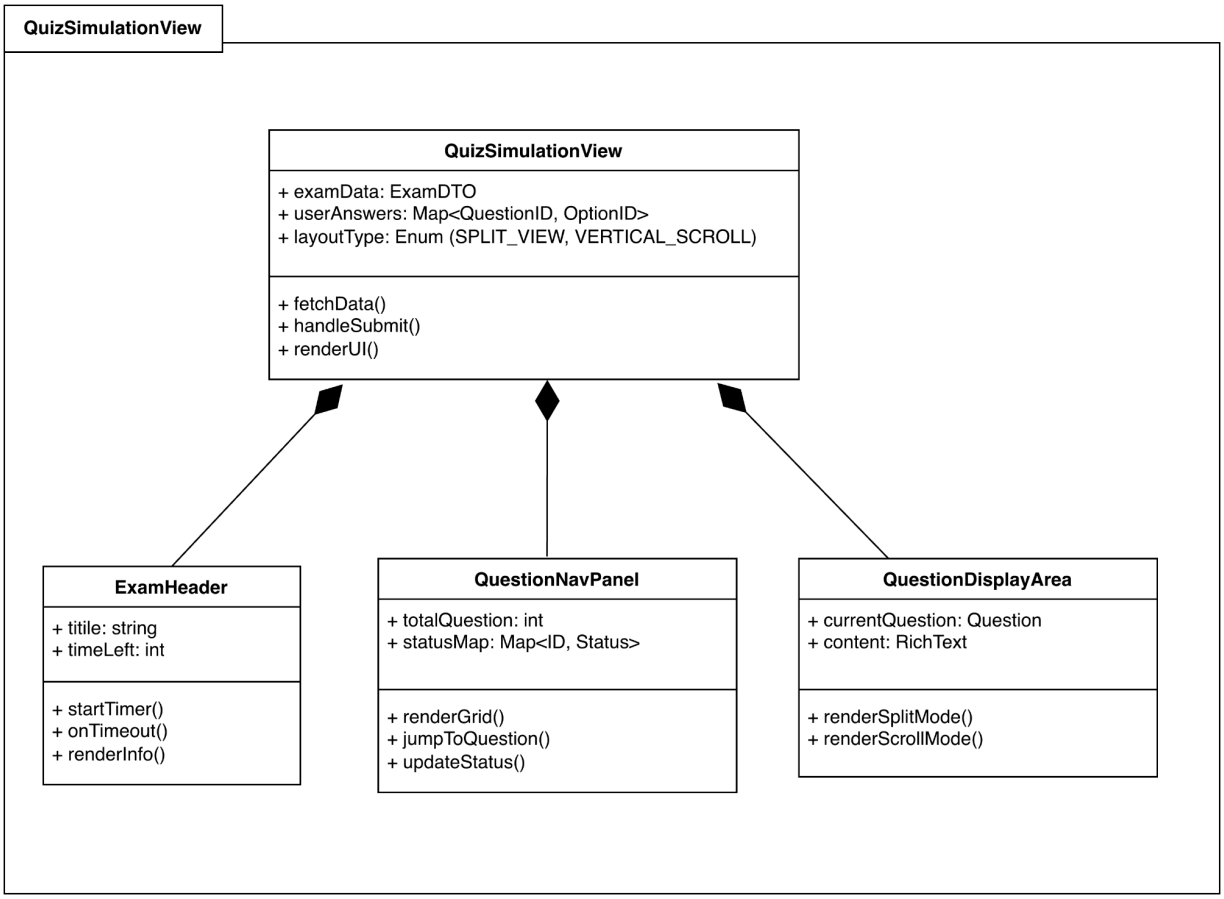
Description: This component is the primary interface responsible for managing the multi-step process required for a Teacher or Admin to create and publish a new exam in the GoPass system. It orchestrates the flow of data across

three sub-panels: basic information input, question selection/management, and final review/submission.

Explain:

+ renderUI(): Renders the correct sub-panel (0, 1, or 2) based on the currentStep.
+ handleNext(): Validates data from the current panel, updates examData, and increments currentStep.
+ handleBack(): Decrements currentStep, allowing the user to revisit previous steps for editing.
+ submitExam(): saving the exam in the database and triggering its assignment.
+ handleSaveInfo(): Collects all input data and passes it back to the ExamCreationView.
+ validateInput(): Performs checks to ensure all mandatory fields are filled and valid.
+ renderSummary(): Displays a read-only review of examData (title, settings).
+ handleSubmit(): Triggers the submitExam() function in the parent ExamCreationView after receiving final confirmation from the user.

4.2.1.1.4 Quiz Stimulation



Description: This module manages the active examination session, coordinating the timer, question navigation, and content display. It dynamically adapts the layout (Split-View or Vertical-Scroll) based on the subject type to optimize reading and answering efficiency.

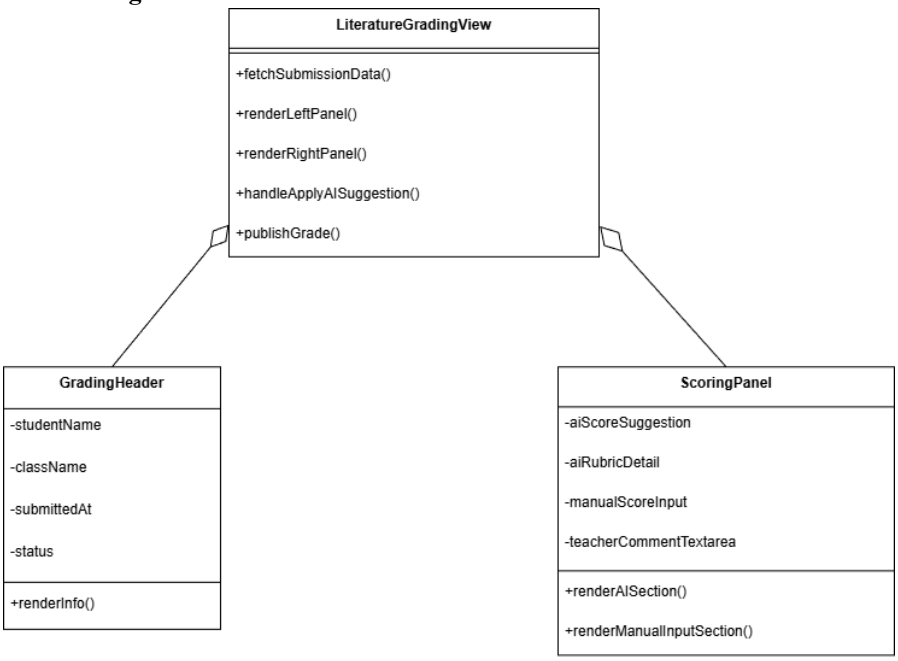
Explain:

1. Class: QuizSimulationView
    + fetchData(): Loads the exam questions and settings from the server.
    + handleSubmit(): Collects user answers and submits them for grading.
    + renderUI(): Orchestrates the layout of the Header, Navigation Panel, and Question Area.
2. Class: ExamHeader

+ startTimer(): Runs the countdown clock.
+ onTimeout(): Auto-submits the exam when time runs out.
+ renderInfo(): Displays the exam title and subject context.

3. Class: QuestionNavPanel
    + renderGrid(): Displays the clickable grid of question numbers.
    + jumpToQuestion(): Scrolls the view to the selected question.
    + updateStatus(): Visually marks questions as "Answered" or "Flagged".
4. Class: QuestionDisplayArea
    + renderSplitMode(): Renders a side-by-side layout.
    + renderScrollMode(): Renders a standard vertical list of questions.

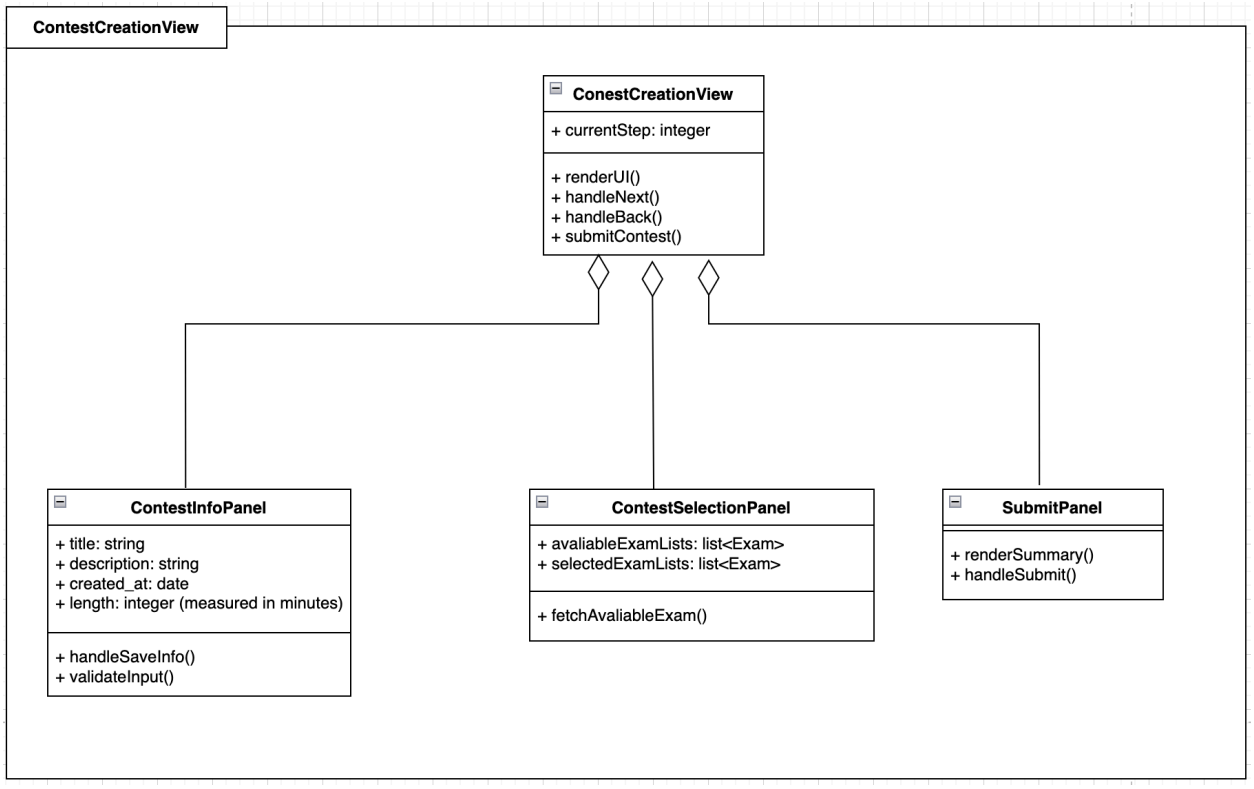
4.2.1.1.5 Literature Grading



Description: This module provides the grading interface for teachers, allowing them to review student essays and assign final scores. It displays the student's submission (left panel) and the scoring tools (right panel), including AI-generated suggestions to support faster and more consistent grading.
Explain:

1. Class: LiteratureGradingView:
    + fetchSubmissionData(): Loads the student's essay and AI analysis from the server.
    + renderLeftPanel(): Displays the prompt and student's essay in read-only mode.
    + renderRightPanel(): Renders the scoring tools, including AI suggestions and manual input fields.
    + handleApplyAISuggestion(): Copies AI-suggested score and comments into the manual input area for quick editing.
    + publishGrade(): Submits the final teacher score and comments to the server and updates the submission status to "Graded".
2. Class: GradingHeader
    + renderInfo(): Shows student name, class, submission time, and current grading status.
3. Class: ScoringPanel
    + renderAISection(): Displays AI-generated score and rubric breakdown with a highlighted style.
    + renderManualInputSection(): Provides fields for teachers to enter the final score and comments.

4.2.1.1.6 Contest Create

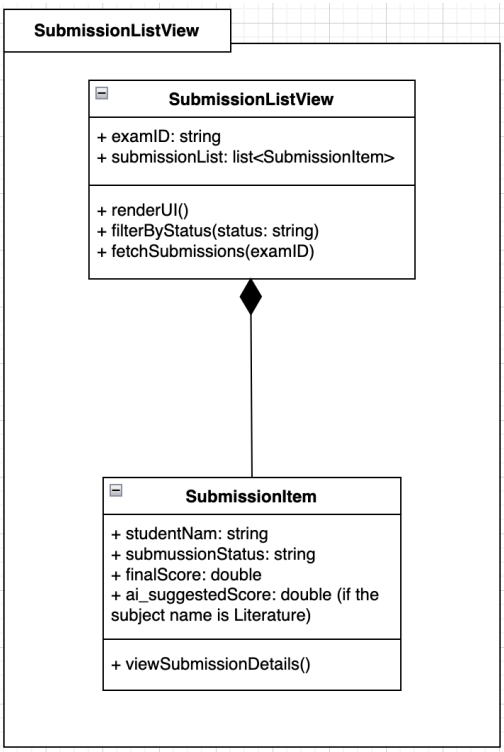


Description: This component is the primary interface responsible for managing the multi-step process required for an Administrator to create and publish a new system-wide Contest in the GoPass platform. It orchestrates the flow of contest data across three sub-panels: basic contest information input, selection of available exams, and final review/submission.

Explain:

+ renderUI(): Renders the correct sub-panel (0, 1, or 2) based on the currentStep.
+ handleNext(): Validates data from the current panel, updates contestData, and increments currentStep.
+ handleBack(): Decrements currentStep, allowing the user to revisit previous steps for editing.
+ submitContest(): saving the contest in the database and triggering its assignment.
+ handleSaveInfo(): Collects all input data and passes it back to the ContestCreationView.
+ validateInput(): Performs checks to ensure all mandatory fields are filled and valid.
+ renderSummary(): Displays a read-only review of contestData (title, settings).
+ handleSubmit(): Triggers the submitContest() function in the parent ContestCreationView after receiving final confirmation from the user.

4.2.1.1.7 List students enrolled in exam



Description: The main container responsible for displaying the list of all student submissions for a specific assigned exam. This view is primarily used by the Teacher to monitor grading status.

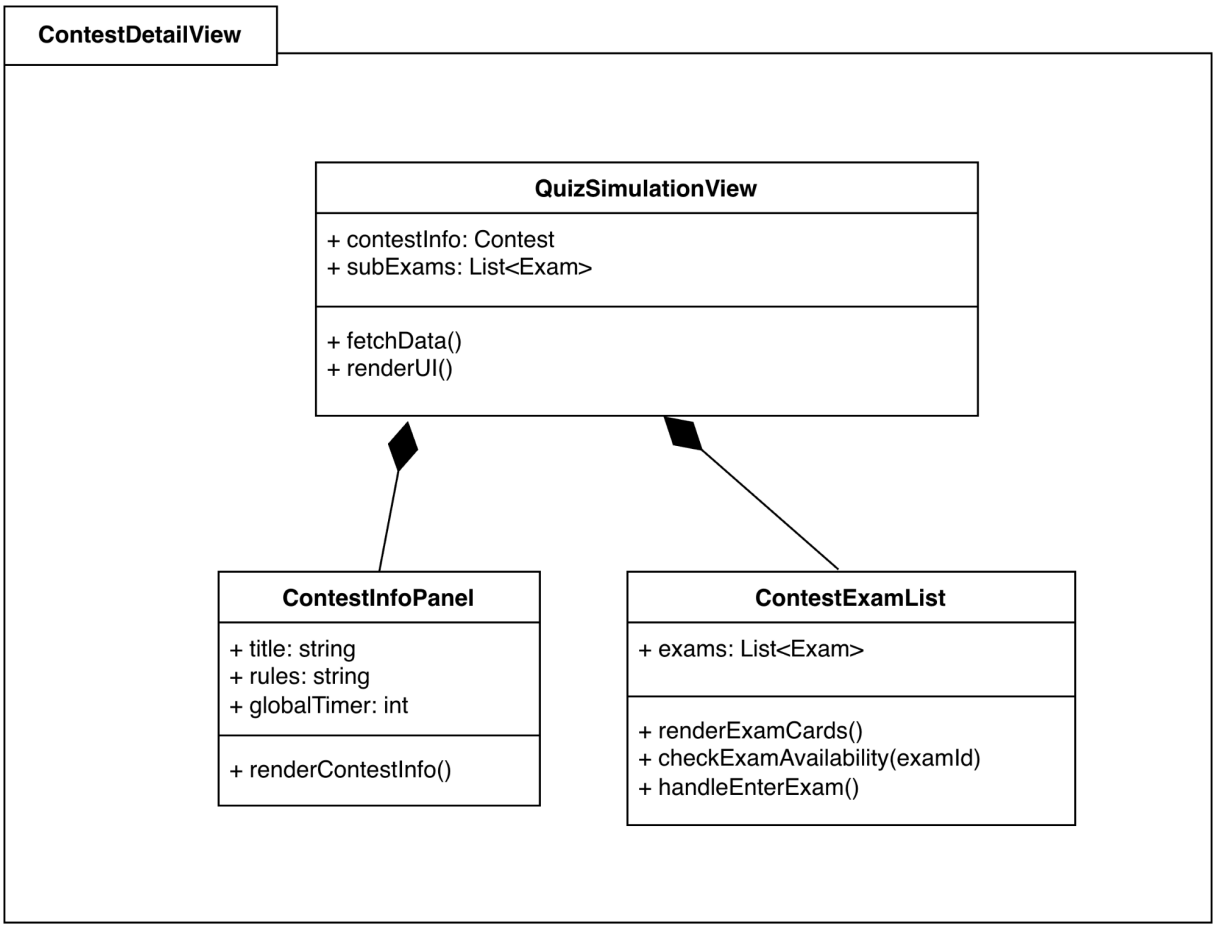
Detail:

+ filterByStatus(status): Filters the local list to show only submissions matching a specific status.

+ fetchSubmissions(examID): Calls the appropriate Controller API to retrieve the list of Submission for the given exam.

+ viewSubmissionDetails(): Handles the action when the Teacher clicks the "Review" or "Grade" button.

4.2.1.1.8 Contest Details

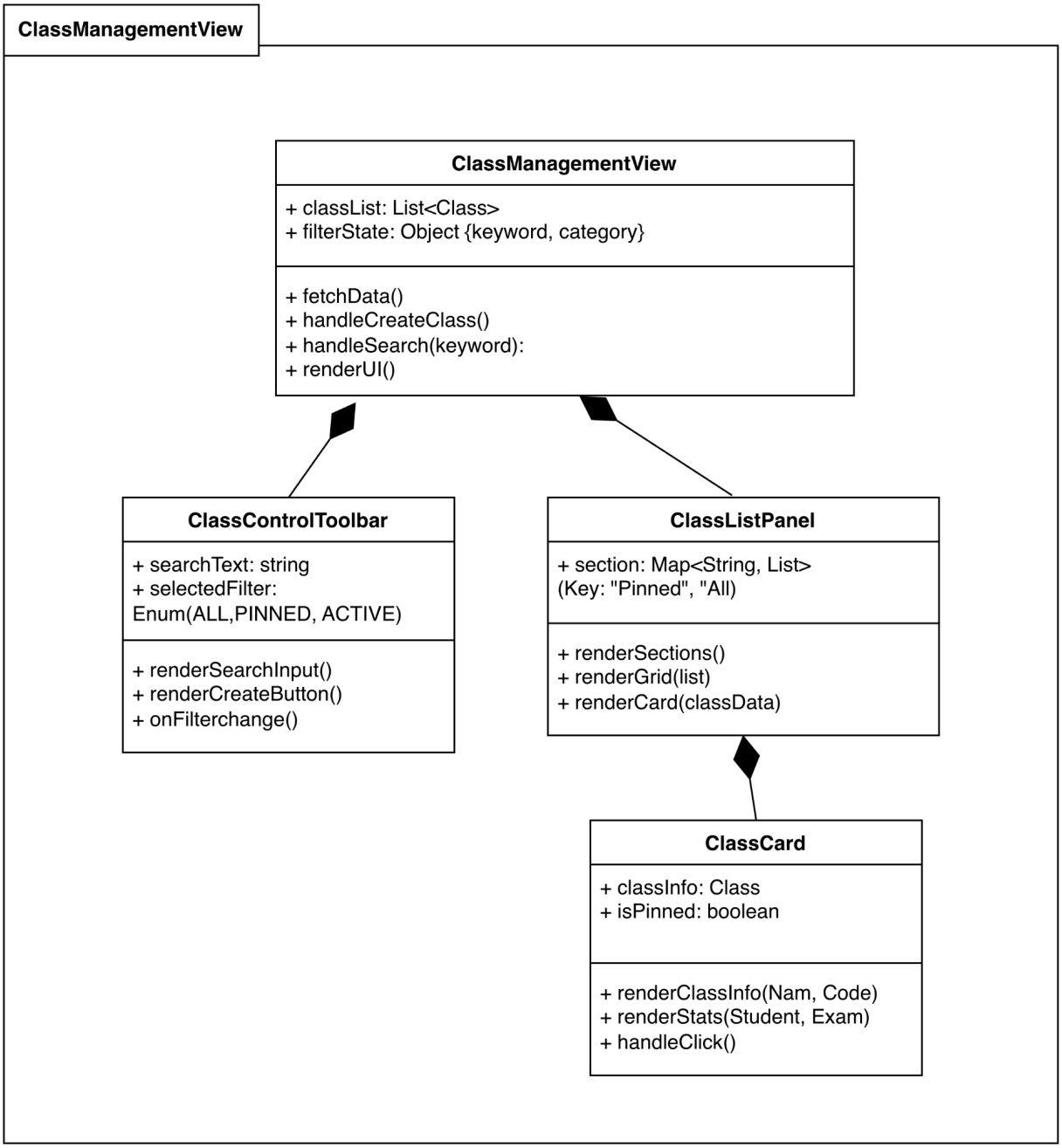


Description: This module serves as the lobby for a specific competition. It displays key metadata (rules, remaining time) and lists the available subject exams, allowing users to select and enter an active test directly without a registration step.

Explain:

1. Class: ContestDetailView (Note: Labeled QuizSimulationView in diagram)
    + fetchData(): Loads the contest details and the list of sub-exams from the server.
    + renderUI(): Combines the info panel and exam list into the main layout.
2. Class: ContestInfoPanel
    + renderContestInfo(): Displays the contest title, rules, and the live globalTimer countdown.
3. Class: ContestExamList
    + renderExamCards(): Displays the list of exams (e.g., Math, Physics) included in this contest.
    + checkExamAvailability(examId): Verifies if the selected exam is currently happening (started and not yet ended).
    + handleEnterExam(): Navigates the user to the taking screen if the exam is available.

4.2.1.1.9 Class Management



Description: This module provides the administrative interface for Teachers and Admins to manage their classrooms, allowing users to organize classes, search by specific codes/names, and filter views. It visually distinguishes between "Pinned/Marked" classes and the general list for better accessibility.

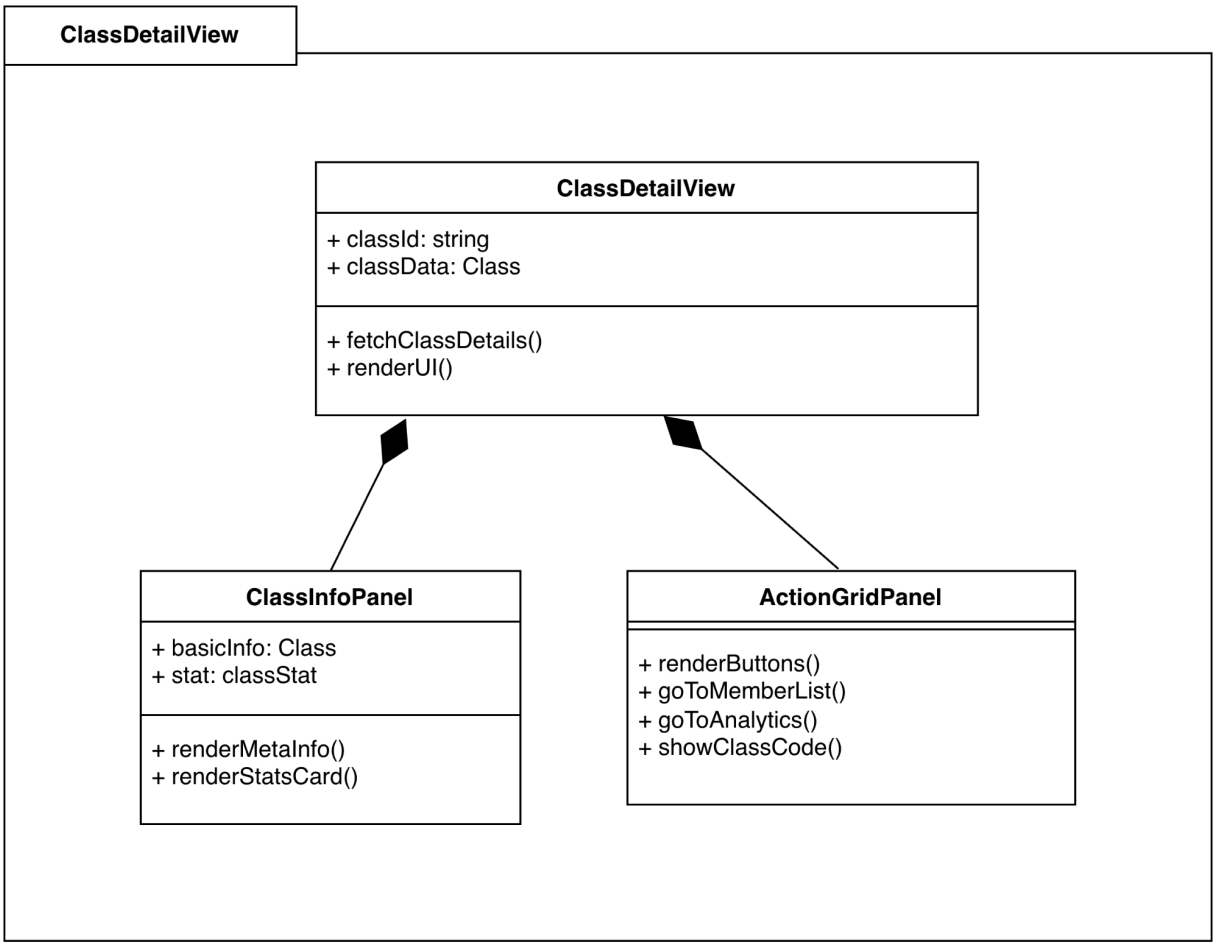
Explain:

1. Class: ClassManagementView
    + fetchData(): Initiates the data fetching process to retrieve the list of classes owned or managed by the current user.
    + handleCreateClass(): Triggered by the “Create A New Class” button. It opens the Class Creation

Wizard (modal or new screen) to input new class details.
- + handleSearch(keyword): Updates the displayed list in real-time based on the text entered in the search bar.
- + renderUI(): Orchestrates the layout, placing the control toolbar at the top and the content panel below.

2. Class: ClassControlToolbar
   - + renderSearchInput(): Renders the search bar with a placeholder.
   - + renderCreateButton(): Displays the button for adding a new class.
   - + onFilterChange(): Handles dropdown selections (e.g., “All”, “Pinned”, “Active”) to sort or filter the class list.
3. Class: ClassListPanel
   - + renderSections(): Logic to divide the class list into logical groups, such as "Pinned/Favorite Classes" and "All Classes".
   - + renderGrid(list): Renders the classes within each section as a responsive grid of cards.
4. Class: ClassCard
   - + renderInfo(): Displays the class identity, including the Class Name and the unique Class Code.
   - + renderStats(): Visualizes key metrics for the class, such as the number of students, total exams created, and the average score.
   - + handleClick(): Navigates the user to the detailed view of the selected class for deeper management.

4.2.1.1.10 Class Detail

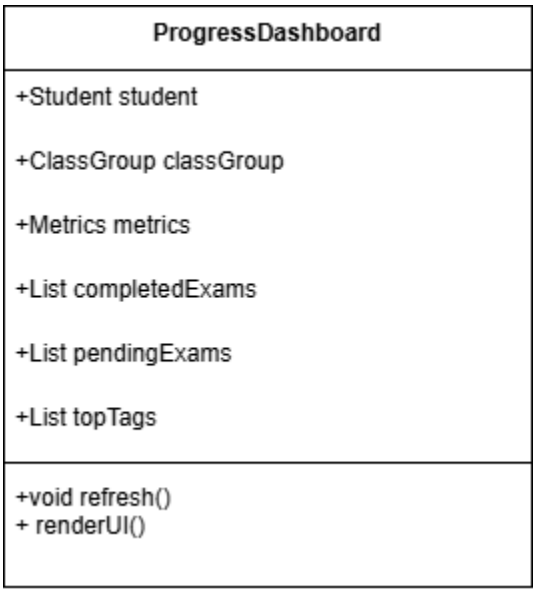


Description: This module serves as the command center for a specific class. It aggregates high-level performance metrics and provides navigation entry points for teachers to manage members, view progress analytics, or share access codes.

Explain:

1. Class: ClassDetailView
    + fetchClassDetails(): Initiates an API call using the classId to retrieve the comprehensive data package (Info, Stats, Logs) for the class.
    + renderUI(): Assembles the view by stacking the ClassInfoPanel and ActionGridPanel vertically.
2. Class: ClassInfoPanel
    + renderMetaInfo(): Displays the class identity: Title, Description, Tags (e.g., #UNI2025), and creation date.
    + renderStatsCard(): Renders the 4 key performance indicator (KPI) cards (Students, Exams, Avg Score, Growth) to give an instant health check of the class.
3. Class: ActionGridPanel
    + renderButton(): Render buttons for "Manage Members", "View Progress", "Join Code"
    + goToMemberList(): Corresponds to the "Manage Members" button. Navigates to the student list management view.
    + goToAnalytics(): Corresponds to the "View Progress" button. Navigates to the detailed analytics charts.
    + showClassCode(): Corresponds to the "Join Code" button. Opens a modal displaying the unique class code for sharing

4.2.1.1.11 Student Progress



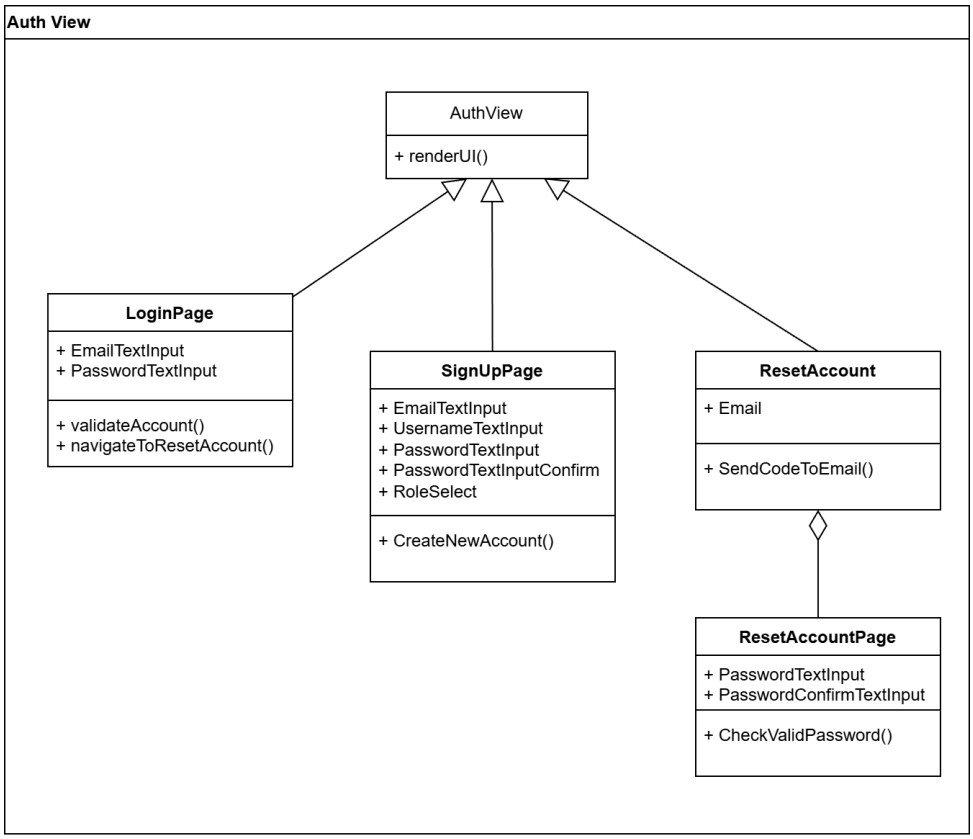
Description: This component displays a student's academic progress, including performance metrics, completed exams, pending exams, and learning trends. It aggregates data from multiple sources to provide a unified dashboard view that helps students understand their strengths, weaknesses, and study priorities.

Explain:

Class: ProgressDashboard

+ student: The student whose progress is being displayed.
+ classGroup: Information about the student's class or cohort.
+ metrics: Key performance indicators such as average score, improvement rate, or consistency index.
+ completedExams: A list of exams the student has already finished.
+ pendingExams: A list of upcoming or unfinished exams.
+ topTags: Frequently occurring topics or tags representing the student's strengths or weaknesses.

4.2.1.1.12 Auth View



Description: This module manages all authentication-related user flows, including logging in, signing up, and resetting forgotten passwords. It provides UI screens for credential input, account validation, and secure password recovery through email verification. The module ensures a consistent AuthView layout while delegating specific logic to individual pages.

Explain:

Class: AuthView
- \+ renderUI(): Renders the appropriate authentication screen (Login, Sign-Up, or Reset) based on user navigation or application state.

Class: LoginPage
- \+ EmailTextInput, PasswordTextInput: Fields for entering login credentials.
- \+ validateAccount(): Checks user credentials against the system to authenticate the account.
- \+ navigateToResetAccount(): Redirects users to the password-reset flow when they forgot their password.

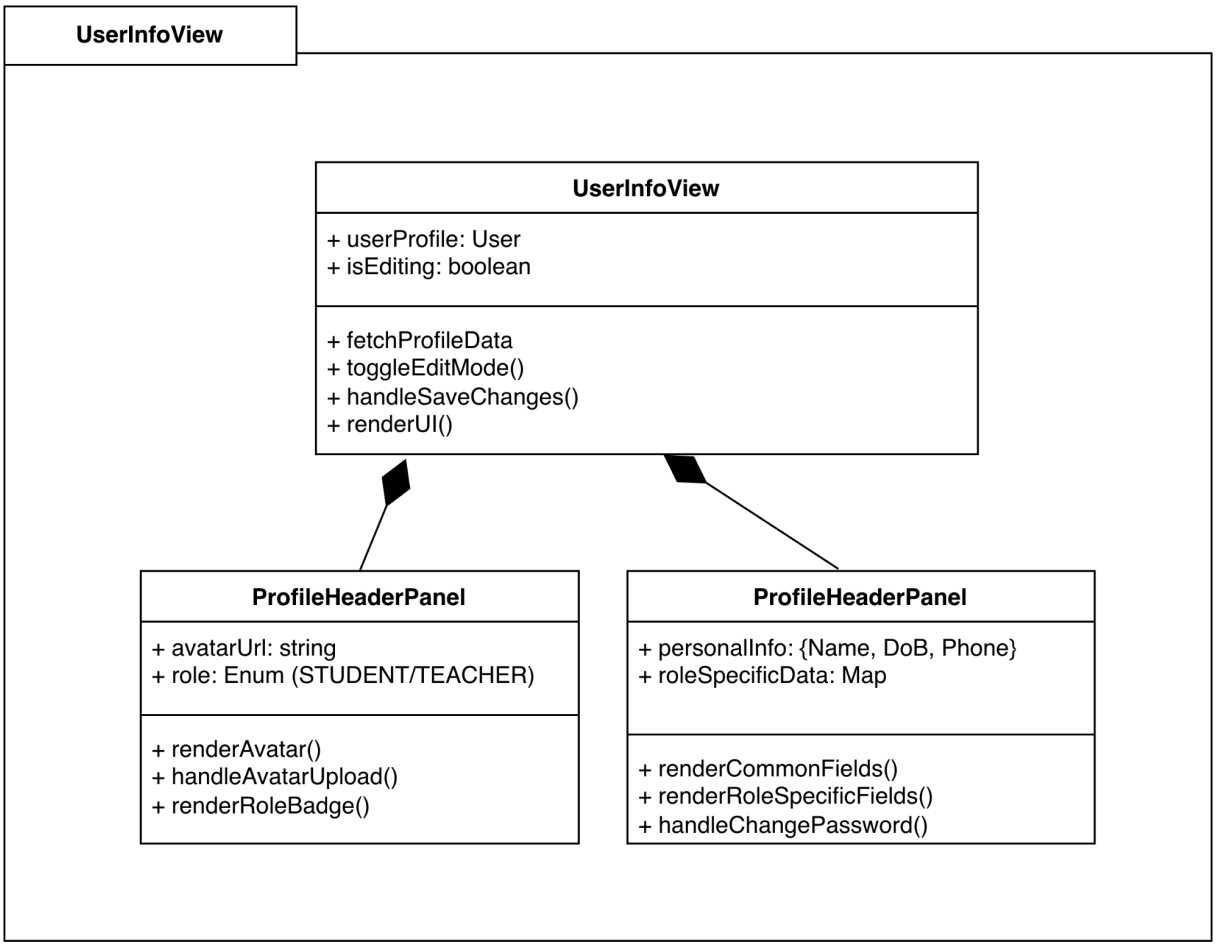
Class: SignUpPage
- \+ EmailTextInput, UsernameTextInput, PasswordTextInput, PasswordTextInputConfirm, RoleSelect: Inputs for creating a new user.
- \+ CreateNewAccount(): Validates fields and submits new account data to the backend.

Class: ResetAccount
- \+ Email: Input for the user's registered email.
- \+ SendCodeToEmail(): Sends a verification/reset code to the provided email to begin the password recovery process.

Class: ResetAccountPage
- \+ PasswordTextInput, PasswordConfirmTextInput: Inputs for entering a new password.

+ CheckValidPassword(): Validates the new password and updates it after successful email verification.

4.2.1.1.13 User Info



Description: This module manages the user's personal profile page. It adapts the interface based on the user's role (Student, Teacher, or Admin) to display relevant specific attributes (e.g., academic grade for students vs. professional qualification for teachers) and provides functionality for updating personal details and account security settings.

Explain:

1. Class: UserInfoView
    + fetchProfileData(): Fetches the full user profile details from the server.
    + toggleEditMode(): Switches the view between "Read-Only" mode and "Edit" mode (unlocking input fields).
    + handleSaveChanges(): Collects modified data, performs client-side validation (e.g., phone number format), and submits changes to the backend.
    + renderUI(): Orchestrates the layout by combining the ProfileHeaderPanel and InfoTabPanel. It passes the isEditing state down to these components to determine whether to render static text or editable input fields.
2. Class: ProfileHeaderPanel
    + enderAvatar(): Displays the user's profile picture.
    + handleAvatarUpload(): Allows users to upload or change their profile image.

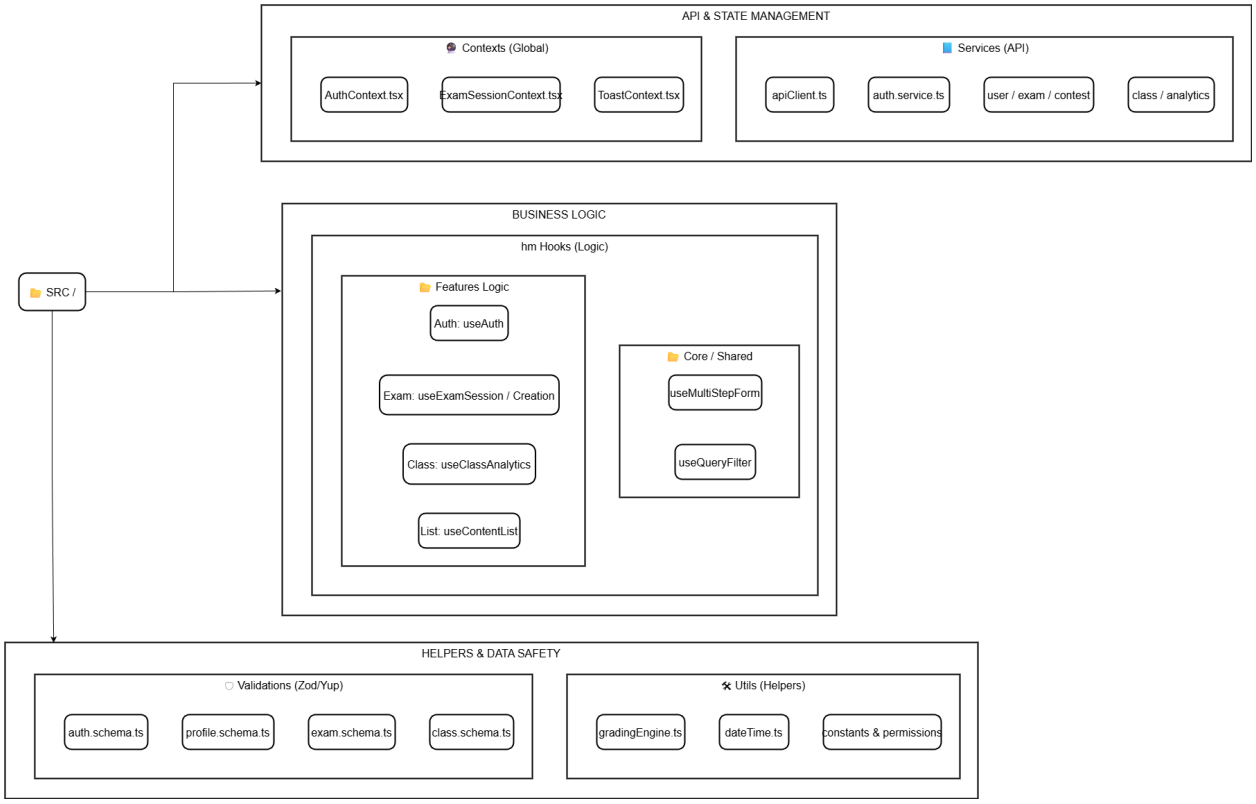
+ renderRoleBadge(): Visualizes the user's role (e.g., a "Teacher" tag) to clearly identify account privileges.

3. Class: InfoTabPanel
   + renderCommonFields(): Renders standard input fields applicable to all users (Name, Email, Phone, Bio).
   + renderRoleSpecificFields(): Dynamically renders extra inputs based on role.
   + handleChangePassword(): Provides a secure form section to update login credentials.

4.2.1.2 Client Logic & State

4.2.1.2.1 Source Code Organization

The internal structure of the BLL is organized to maximize maintainability and scalability. The directory hierarchy distinguishes between generic infrastructure and feature-specific domains.



4.2.1.2.2 Component Responsibility Specification

This section details the responsibilities of each architectural block within the file structure.

4.2.1.2.2.1 Global State Contexts (src/contexts/)

This module implements the Provider Pattern to manage states that must persist across navigation lifecycles or be accessible by disjointed components.

- **AuthContext.tsx:** The security backbone of the frontend. It maintains the currentUser object and isAuthenticated flag. It exposes methods for session hydration and permission checking (checkPermission).
- **ExamSessionContext.tsx:** Manages the transient state of an active examination. Critical for data integrity, it ensures that userAnswers and timer values are synchronized and do not reset if the user accidentally refreshes the page.

4.2.1.2.2.2 Custom Hooks (src/hooks/)

The Hooks directory represents the primary location for Business Logic implementation. It is divided into two sub-categories:

- **Core Hooks (src/hooks/core/)** Generic, framework-level logic that is agnostic to the business domain.
    - **useMultiStepForm.ts**: Implements a State Machine to manage linear workflows (e.g., Next/Back navigation, Step Validation). Used by both Exam Creation and Contest Creation modules.
    - **useQueryFilter.ts:** Manages URL search parameters to synchronize filter states (Subject, Status) with the browser's address bar, facilitating deep linking.
- **Feature Hooks (src/hooks/features/)** Domain-specific logic tailored to specific Use Cases.
    - **useDashboardStrategy.ts:** Implements the Strategy Design Pattern. It dynamically selects the appropriate data fetching strategy based on the userRole (Student, Teacher, or Admin) detected in the AuthContext.
    - **useExamSession.ts:** Orchestrates the Quiz Simulation. It combines the timer logic, answer recording, and autosubmit triggers into a single interface for the View.
    - **useClassAnalytics.ts:** Handles the aggregation of raw class data into statistical metrics (KPIs) and chart-compatible formats.

4.2.1.2.2.3 Utilities & Pure Logic (src/utils/)

This directory contains pure functions and deterministic algorithms that do not depend on React state or lifecycle methods.

- **gradingEngine.ts:** A client-side simulation engine used for immediate feedback. It calculates tentative scores based on answer keys before the official server-side grading occurs.
- **permissions.ts:** Defines the Role-Based Access Control (RBAC) matrix, mapping user roles to specific feature capabilities.

4.2.1.2.2.4 Data Validation (src/validations/)

To ensure data integrity and reduce server load, this module defines strict schemas.

- **Purpose:** These schemas act as a gatekeeper. Forms such as Exam Creation (exam.schema.ts) or User Profile (profile.schema.ts) must pass these validation rules (regex patterns, mandatory fields, value ranges) before an API request is permitted.

4.2.1.3 Client API Layer (src/services/)

- **apiClient.ts (Core Infrastructure)**
    - **Description:** This file exports a singleton axios instance configured with the application's base URL and timeout settings. It serves as the backbone for all network requests.
    - **Key Mechanisms:**
        - **Request Interceptors:** Automatically injects the JSON Web Token (JWT) from LocalStorage into the Authorization header (Bearer <token>) for every outgoing request, ensuring seamless authentication.
        - **Response Interceptors:** A centralized error handling mechanism that intercepts non-2xx responses. It automatically detects 401 Unauthorized errors to trigger a logout/redirect flow and standardizes server error messages before they reach the UI.
- **user.service.ts (Profile & Account Management)**
    - **Description**: Manages the retrieval and modification of user-centric data.
    - **Responsibilities**:
        - **Profile Synchronization:** Fetches detailed user information (Avatar, Bio, Role) to populate the UserInfoView.
        - **Account Security:** Handles sensitive operations such as password changes and personal information updates.
        - **DTO Mapping:** transforms raw user data from the backend into a sanitized UserProfile interface used by the frontend.
- **exam.service.ts (Assessment Lifecycle)**
    - **Description:** Handles the core business logic related to the creation, execution, and submission of examinations.
    - **Responsibilities:**
        - **Content Retrieval:** Fetches exam metadata and question sets. It includes logic to shuffle questions if required by client-side settings.
        - **Submission Handling:** Transmits the user's answer map (Map<QuestionId, Answer>) to the backend for grading.

        - **Creation Workflow:** Orchestrates the multi-step payload generation for creating new exams, ensuring the data structure matches the backend's CreateExamDto.
- **contest.service.ts (Event Coordination)**
    - **Description:** Manages logic specific to scheduled competitions, distinct from standard practice exams.
    - **Responsibilities:**
        - **Logistics Check:** Verifies server time against contest start/end times to determine availability (Upcoming, Ongoing, Ended).
        - **Enrollment:** Handles the logic for registering a student for a contest or verifying access permissions (e.g., checking whitelist status).
        - **Leaderboard:** Fetches and caches ranking data for specific contests.
- **class.service.ts (Academic Group Management)**
    - **Description:** Facilitates the administrative governance of virtual classrooms for Teachers and Admins.
    - **Responsibilities:**
        - **Membership Management:** Provides methods to fetch student lists, remove members, or approve join requests.
        - **Access Control:** Generates and refreshes unique "Class Invitation Codes" or links.
        - **Classroom CRUD:** Handles the creation of new classes and the modification of class metadata (Name, Description, Tags).
- **analytics.service.ts (Business Intelligence)**
    - **Description:** Aggregates data for visualization purposes, serving the Dashboard and Progress views.
    - **Responsibilities:**
        - **Metric Aggregation:** Fetches pre-calculated KPIs (Key Performance Indicators) such as "Average Score," "Exam Completion Rate," and "Total Activity Time."
        - **Growth Tracking:** Retrieves time-series data (e.g., scores over the last 30 days) formatted specifically for rendering charts (Line/Bar charts).
        - **Role-Based Data Fetching:** Supports distinct endpoints for different user roles (e.g., getStudentStats() vs. getTeacherMetrics()).

4.2.2 Server-side

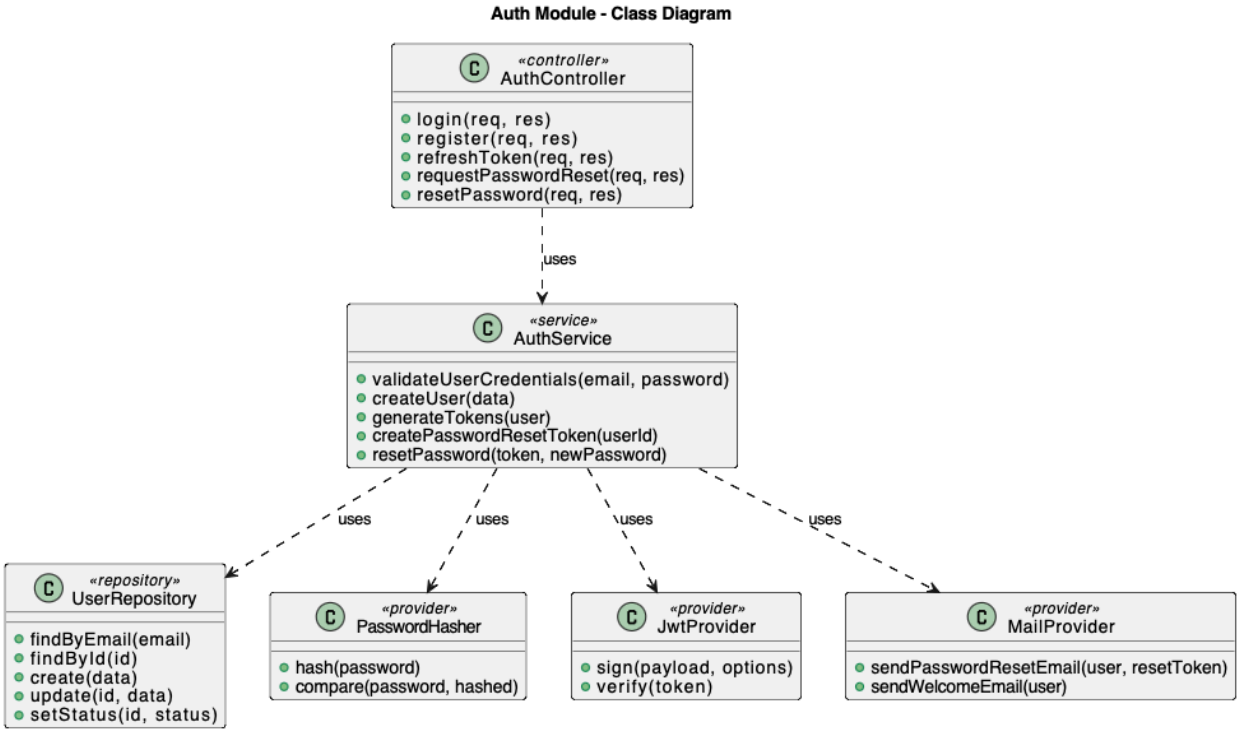
4.2.2.1 Auth

Overview:

Component	Auth – Authentication & Registration
Description	The Authentication module manages all user identity and login-related operations. It provides endpoints for login, registration, token refresh, and password recovery. This module ensures that only authenticated and authorized users can access protected resources.
Responsibility	Controller Responsibilities ● Handle user login and registration requests. ● Manage JWT access and refresh token generation. ● Process password reset requests and token verification. Service Responsibilities ● Validate login credentials and authenticate users. ● Create new user accounts with proper data validation. ● Generate access/refresh tokens using secure JWT mechanisms. ● Create and process password reset tokens. Repository & Provider Dependencies

	• UserRepository: Retrieve and persist user data. • PasswordHasher: Hash and verify user passwords. • JwtProvider: Create and validate JWT tokens. • MailProvider: Send password reset and welcome emails.
Purpose	To provide a secure authentication mechanism and centralized account management for the entire system.

Class Diagram:



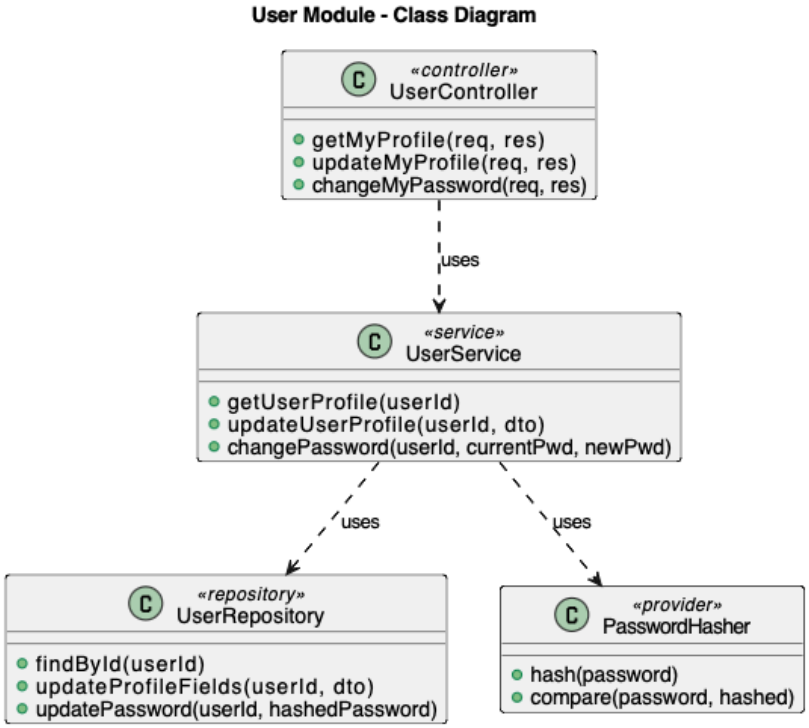
4.2.2.2 User

Overview:

Component	User – Profile Management
Description	The User Profile module allows authenticated users to view and update personal information as well as change their passwords. It provides essential user self-service functionalities.
Responsibility	Controller Responsibilities • Expose endpoints for viewing and updating user profiles. • Allow users to change their own passwords. Service Responsibilities • Retrieve profile information from the database. • Update user profile fields such as name, avatar, and contact info.

	• Validate old passwords before accepting new ones. Repository & Provider Dependencies • UserRepository: Query/update user profile data. • PasswordHasher: Verify and hash passwords for secure storage.
Purpose	To enable users to manage their personal information without admin intervention, improving usability and autonomy.

Class Diagram:



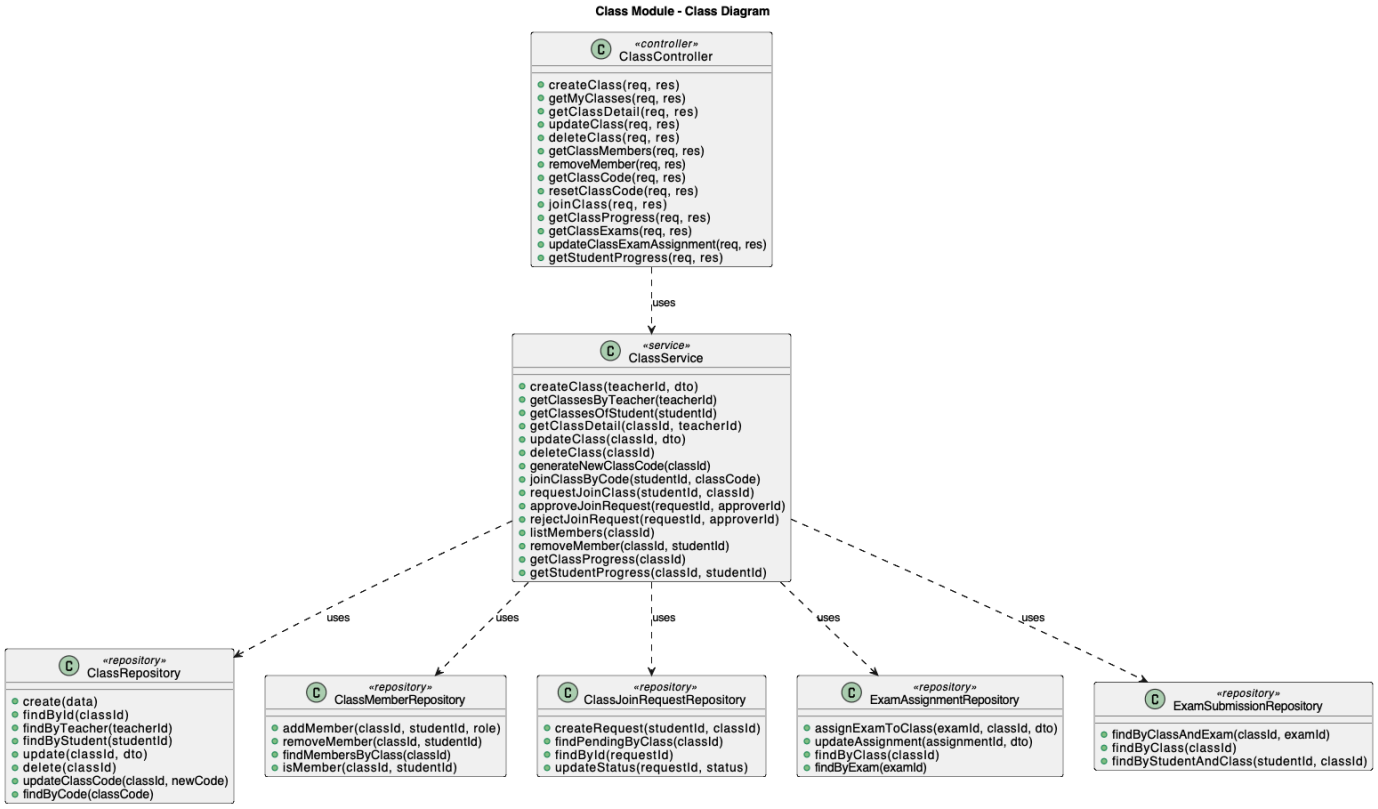
4.2.2.3 Class

Overview:

Component	Class – Class Management & Join Class
Description	This module handles all class-related operations for teachers and students. It includes class creation, membership management, class access via join code, progress tracking, and exam assignment management.
Responsibility	Controller Responsibilities • Provide endpoints for creating, updating, deleting classes. • Allow students to join a class using a class code. • Retrieve class members, assignments, and progress statistics. Service Responsibilities • Perform validation when creating/updating classes.

	• Generate secure class codes and reset them when needed. • Manage class membership and join requests. • Compute class progress and student-level progress. • Handle exam assignments and their scheduling. Repository Dependencies • ClassRepository • ClassMemberRepository • ClassJoinRequestRepository • ExamAssignmentRepository • ExamSubmissionRepository
Purpose	To provide a comprehensive management system for classes and their academic workflows, supporting both teacher and student use cases.

Class Diagram:



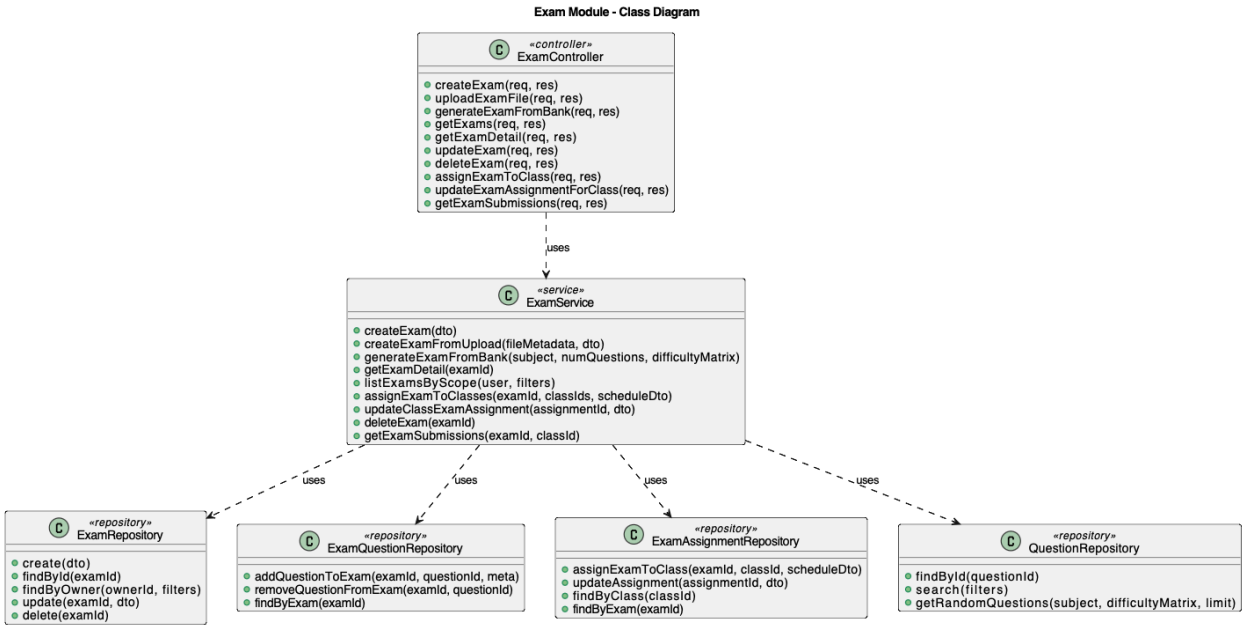
4.2.2.4 Exam

Overview:

Component	Exam – Exam Creation & Assignment
Description	This module focuses on exam creation, modification, uploading exam files, generating

	exams from the question bank, and assigning exams to classes.
Responsibility	Controller Responsibilities ● Manage exam CRUD operations. ● Handle exam uploads and automated exam generation. ● Assign exams to classes and update assignment schedules. Service Responsibilities ● Create new exams manually or from uploaded files. ● Generate exams using the question bank with difficulty filtering. ● Retrieve exam details and list exams by scope or ownership. ● Assign exams to multiple classes with scheduling parameters. ● Retrieve submissions associated with an exam. Repository Dependencies ● ExamRepository ● ExamQuestionRepository ● ExamAssignmentRepository ● QuestionRepository
Purpose	To allow teachers to efficiently create and manage exams, ensuring flexibility in exam generation and assignment.

Class Diagram:



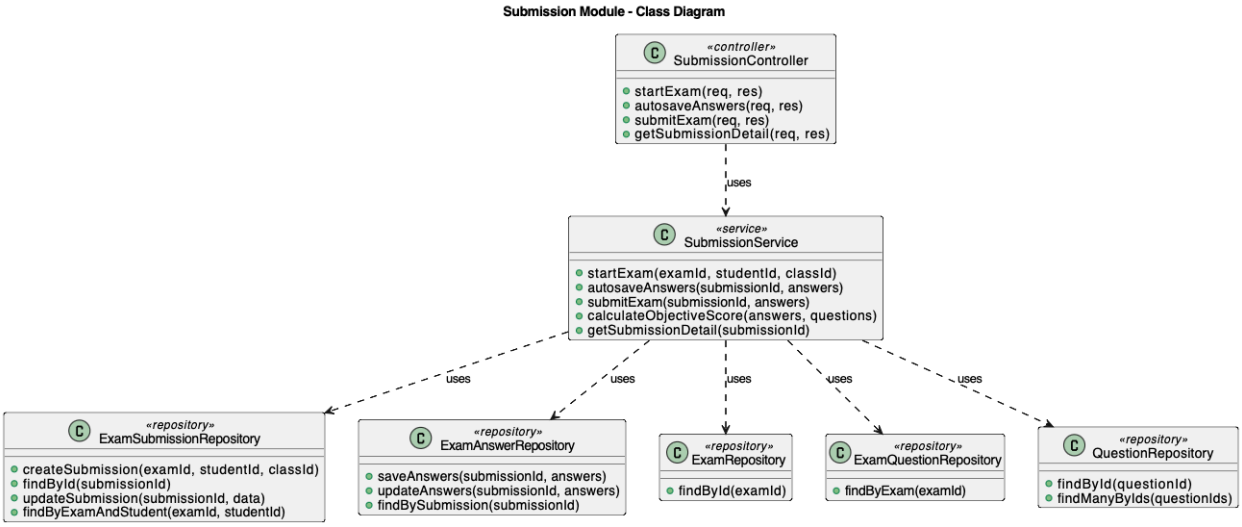
4.2.2.5 Submission

Overview:

Component	Submission – Exam Taking & Submission

Description	This module supports the exam-taking flow for students, including answer autosaving, final submission, and automatic scoring of objective questions
Responsibility	Controller Responsibilities ● Start exam sessions. ● Autosave student answers during the exam. ● Submit the completed exam. ● View submission details. Service Responsibilities ● Initialize an exam session for a student. ● Save answers incrementally to prevent data loss. ● Calculate objective question scores automatically. ● Aggregate student answers and question metadata. ● Provide detailed submission results. Repository Dependencies ● ExamSubmissionRepository ● ExamAnswerRepository ● ExamRepository ● ExamQuestionRepository ● QuestionRepository
Purpose	To provide a secure and robust exam-taking experience with autosave and automated scoring.

Class Diagram:

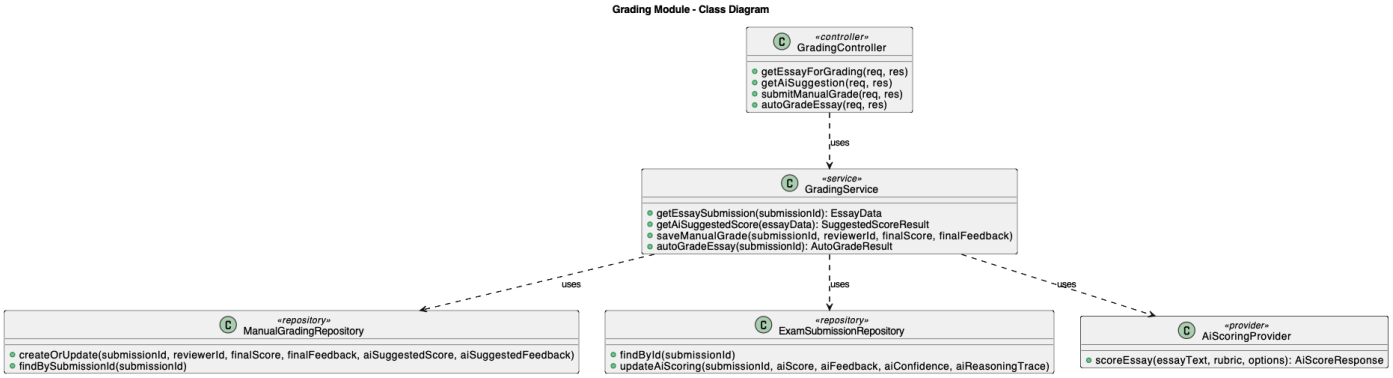


4.2.2.6 Grading

Overview:

Component	Grading – Manual & AI-assisted Essay Grading
Description	This module handles grading for essay-type questions. It supports both manual grading by teachers and AI-assisted grading using a dedicated scoring provider.
Responsibility	Controller Responsibilities ● Retrieve essay submissions for grading. ● Request AI-generated score suggestions. ● Submit final manual grades. ● Automatically grade essays via AI. Service Responsibilities ● Fetch essay data and prepare it for grading. ● Request AI scoring with feedback and metadata. ● Persist manual grades from teachers. ● Update submission records with AI-generated scoring data. Repository & Provider Dependencies ● ManualGradingRepository: Store manual and AI grading records. ● ExamSubmissionRepository: Attach AI results to submissions. ● AiScoringProvider: Integrate with OpenAI/Gemini or internal AI model.
Purpose	To streamline essay grading, reduce teacher workload, and provide consistent, explainable AI scoring with manual override capability.

Class Diagram:

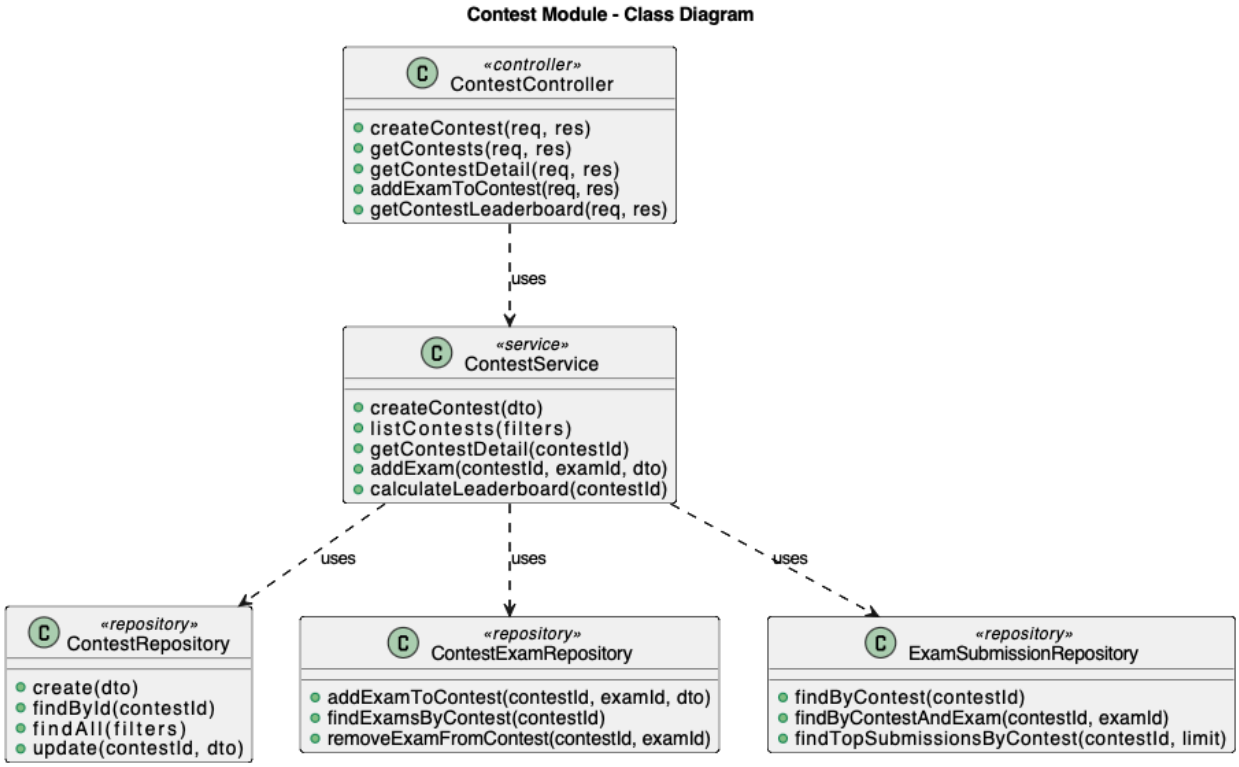


4.2.2.7 Contest

Overview:

Component	Contest – Contests & Leaderboards
Description	This module manages academic competitions, including contest creation, exam assignment within contests, and leaderboard generation.
Responsibility	Controller Responsibilities ● Create and list contests. ● Retrieve contest details. ● Add exams to contests. ● Generate leaderboard results. Service Responsibilities ● Validate contest configuration. ● Manage contest-level exam associations. ● Compute rankings based on exam submissions. ● Retrieve statistics and scoring data. Repository Dependencies ● ContestRepository ● ContestExamRepository ● ExamSubmissionRepository
Purpose	To support competitive exam activities with automatic scoring, ranking, and leaderboard management.

Class Diagram:



4.2.2.8 QuestionBank

Overview:

Component	QuestionBank – Question Storage & Retrieval
Description	The QuestionBank module provides CRUD operations for questions and enables advanced search and random selection for exam generation.
Responsibility	Controller Responsibilities ● Create, update, archive, and delete questions. ● Search questions with filters. ● Provide random questions for exam generation. Service Responsibilities ● Validate question content before saving. ● Retrieve filtered question sets. ● Select random questions based on subject and difficulty. ● Ensure question availability before inclusion in exams. Repository Dependencies ● QuestionRepository ● ExamQuestionRepository (to check question usage)
Purpose	To maintain a structured question database that supports flexible exam generation.

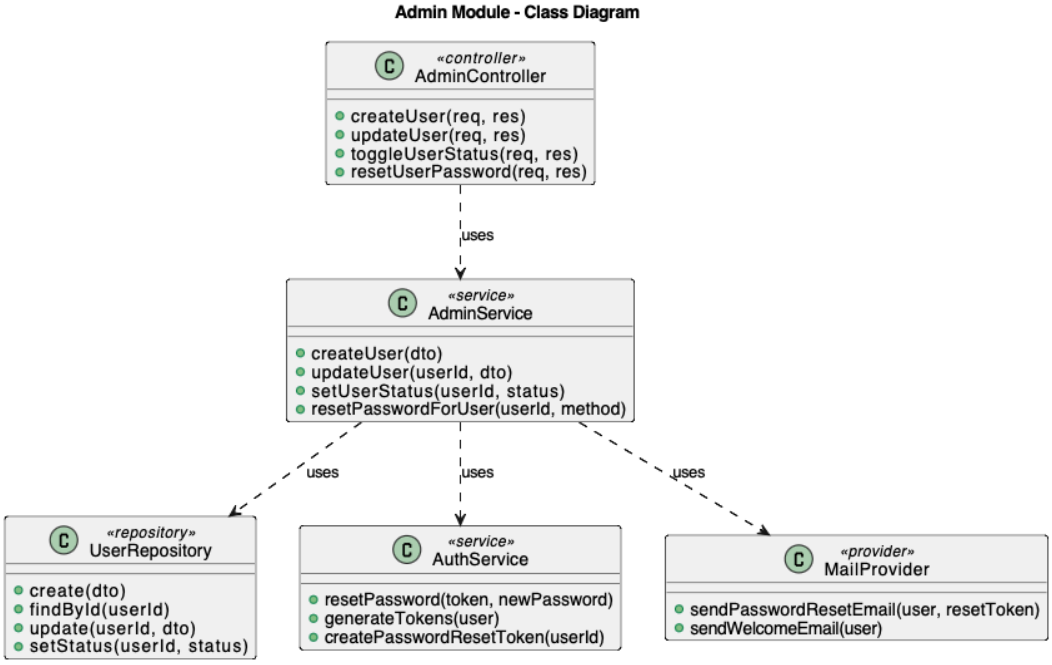
Class Diagram:

4.2.2.9 Admin

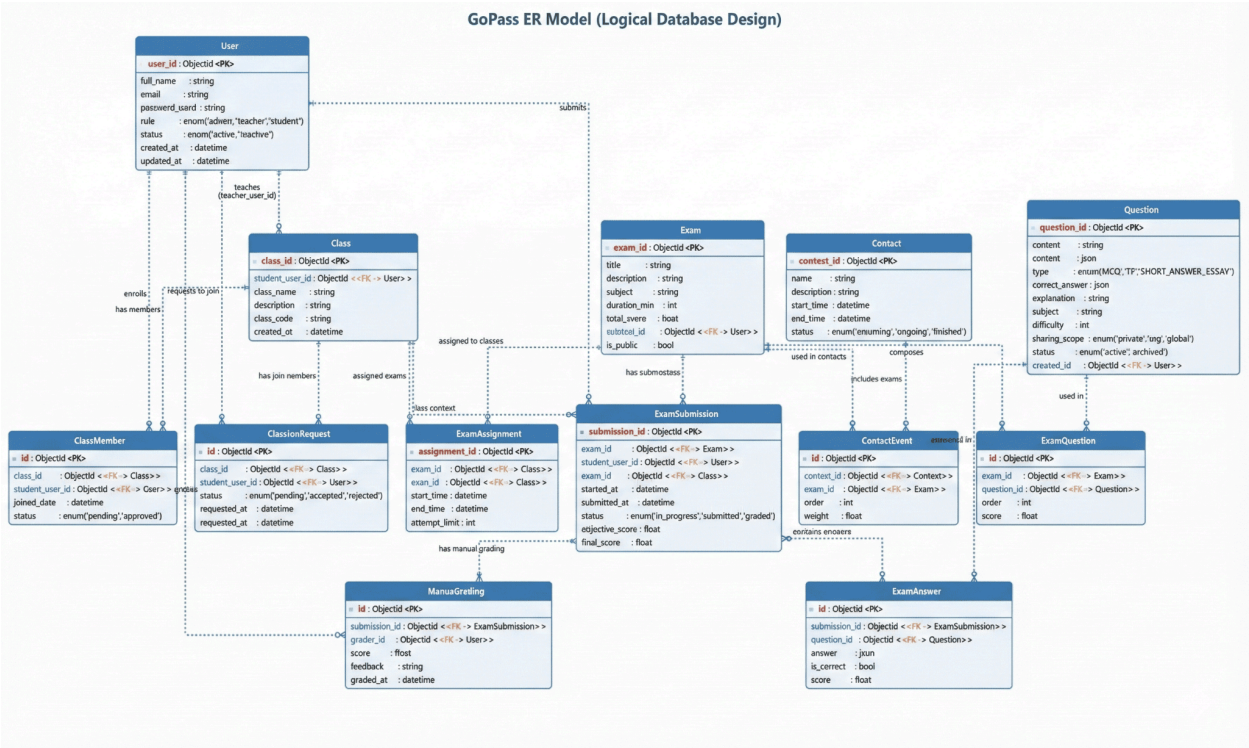
Overview:

Component	Admin – System Administration
Description	The Admin module manages system-level operations such as user creation, updates, status control, and administrative password resets.
Responsibility	Controller Responsibilities ● Create and modify user accounts. ● Activate/deactivate user accounts. ● Reset passwords for users. Service Responsibilities ● Perform privileged user management actions. ● Validate admin-level changes. ● Trigger email notifications when needed. Repository & Provider Dependencies ● UserRepository: Manage user data. ● AuthService: Handle password reset logic. ● MailProvider: Communicate password reset emails or status notifications.
Purpose	To allow system administrators to maintain control over users and ensure system integrity.

Class Diagram:



4.2.3 Database

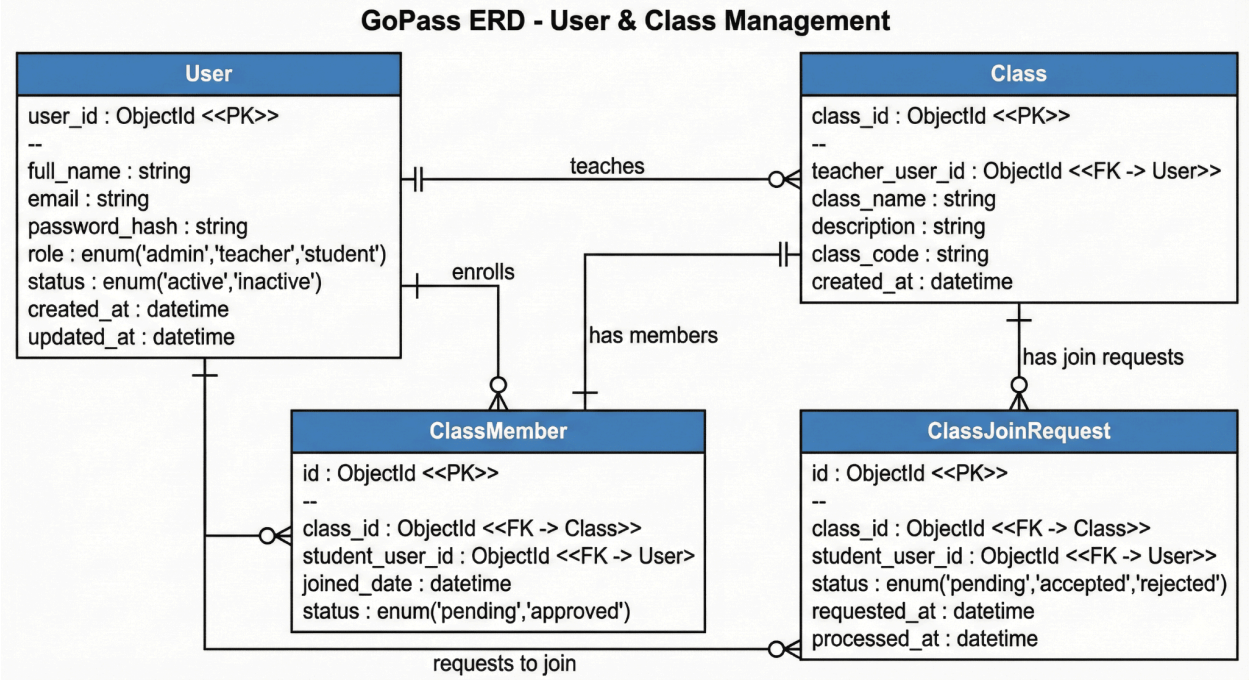


4.2.3.1 User & Class Management

This part of the ER model describes how users and classes are managed in GoPass.

- The User entity stores basic account information such as name, email, role (admin/teacher/student), and status.
- The Class entity represents a teaching class created by a teacher. Each class is linked to exactly one teacher through the teacher_user_id foreign key.
- The ClassMember entity models the many-to-many relationship between students and classes. It records which student joins which class, the join date, and the membership status.
- The ClassJoinRequest entity stores join requests made by students. It tracks which student requested to join which class, the request status (pending/accepted/rejected), and the time of processing.

Together, these entities support core use cases such as creating classes, students joining classes (via code or request), managing members, and approving/rejecting join requests.



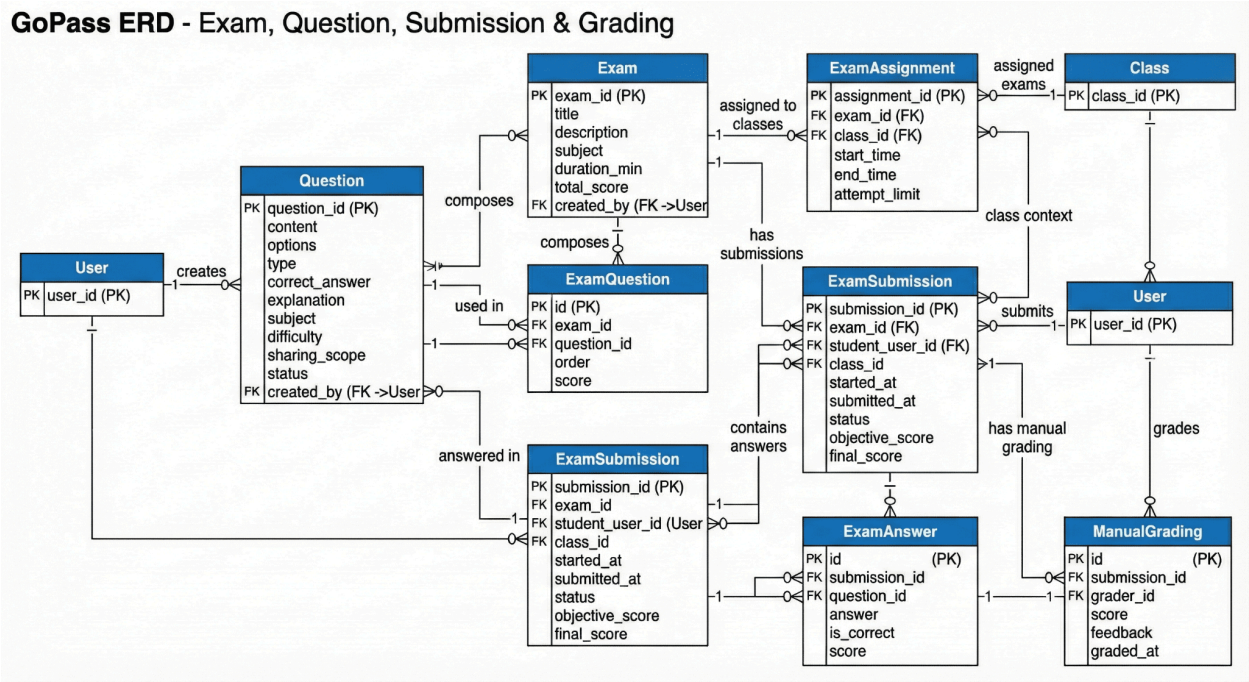
4.2.3.2 Exam, Question, Submission & Grading

This part of the ER model focuses on exams, questions, student submissions, and grading.

- The Question entity represents items in the question bank, including content, options, type (MCQ/TF/short answer/essay), correct answer, and metadata such as subject and difficulty.
- The Exam entity describes an exam with its title, subject, duration, total score, and creator.
- The ExamQuestion entity links exams to questions and defines the order of questions and the score for each question.
- The ExamAssignment entity assigns exams to classes, with start/end time and attempt limits.
- The ExamSubmission entity records each student's attempt for a specific exam in a class, including timestamps, status, and scores.

- The ExamAnswer entity stores the student's answer to each question within a submission, along with correctness and awarded score.
- The ManualGrading entity stores manual grading results (typically for essay questions), linking a submission to a grader and their score/feedback.

This group of entities supports end-to-end assessment workflows: building exams from the question bank, assigning them to classes, students taking exams, automatic scoring for objective questions, and manual/AI-assisted grading for essay questions.

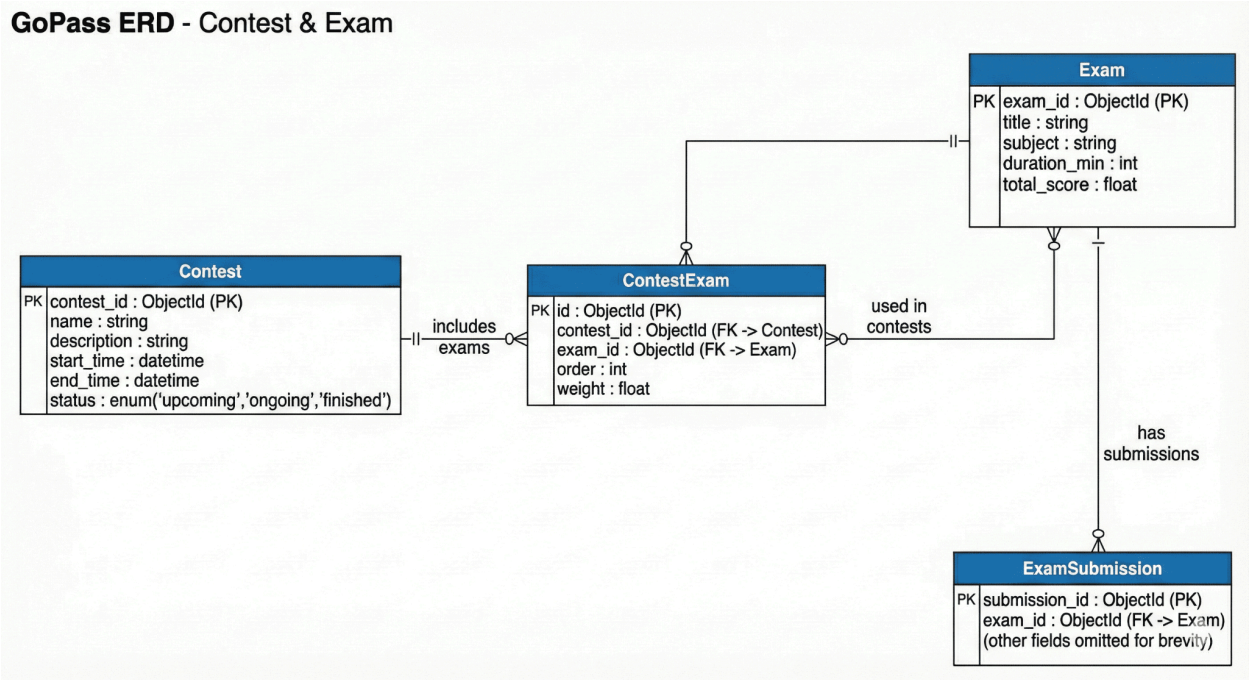


4.2.3.3 Contest & Exam

This part of the ER model describes how exams are organized into contests.

- The Contest entity represents a competition event with a name, description, start/end time, and status (upcoming/ongoing/finished).
- The ContestExam entity links contests to one or more exams. It defines which exams belong to a contest, their order, and relative weight in the contest scoring.
- The ExamSubmission entity (shared with the previous section) is used to capture individual student attempts for contest exams and later calculate rankings.

This structure allows the system to reuse existing exams within larger contest events, aggregate exam results, and support features such as leaderboards and competitive practice.


GoPass ERD - Contest & Exam
Exam
PK exam_id : ObjectId (PK)
title : string
subject : string
duration_min : int
total_score : float
Contest
PK contest_id : ObjectId (PK)
name : string
description : string
start_time : datetime
end_time : datetime
status : enum('upcoming','ongoing','finished')
includes exams
ContestExam
PK id : ObjectId (PK)
contest_id : ObjectId (FK -> Contest)
exam_id : ObjectId (FK -> Exam)
order : int
weight : float
used in contests
has submissions
ExamSubmission
PK submission_id : ObjectId (PK)
exam_id : ObjectId (FK -> Exam)
(other fields omitted for brevity)

5. Deployment

[Leave this section blank for PA4 if you are writing this document for PA3.

In this section, describe how the system is deployed by mapping the components in Section 4 to machines running them. For example, your mobile app is running on a mobile device (Android, iOS, etc), your server runs all components on the server side including the database]

6. Implementation View

[Leave this section blank for PA4 if you are writing this document for PA3.

In this section, provide folder structures for your code for all components described in Section 4.]