System Design Document V2

Exam Jam

Team hireme

March 2023

Table of Contents

CRC Cards

Database Models

Course		
Parent Class ● Model	**Sub Class** ● None	**Collaborations**
Responsibilities ● Database model that represents a course at UofT		**Fields** ● courseCode: string ● title: string ● description: string ● programArea: string[] ● campus: string[]

Exam		
Parent Class ● Model	**Sub Class** ● None	**Collaborations** ● Course
Responsibilities ● Database model that represents an exam belonging to a course		**Fields** ● examId: string ● courseCode: string ● link: string ● data: string | Buffer

Post		
Parent Class ● Model	**Sub Class** ● None	**Collaborations** ● Exam
Responsibilities ● Database model that represents a discussion post belonging to an exam		**Fields** ● examId: string ● postId: string ● author: string ● body: string ● images: string[] ● upvotes: number ● downvotes: number ● tags: string[]

PiazzaPost		
Parent Class ● Model	**Sub Class** ● None	**Collaborations** ● PiazzaComment
Responsibilities		**Fields**

• Database model that represents a post scraped from a piazza forum	• courseCode: string • forumId: string • postNumber: number • title: string • content: string • createdAt: Date

PiazzaComment		
Parent Class • Model	**Sub Class** • None	**Collaborations** • PiazzaPost
Responsibilities • Database model that represents a comment under a piazza post		**Fields** • id: string • postId: string • type: Enum(s_answer, i_answer, followup, feedback) • content: string • parentId: string | null • children: PiazzaComment[]

Tag		
Parent Class • Model	**Sub Class** • None	**Collaborations** • Course • Post
Responsibilities • Database model that represents a tag that can be attached to a Post		**Fields** • courseCode: string • postId: string • name: string

Comment		
Parent Class • Model	**Sub Class** • None	**Collaborations** • Post
Responsibilities • Database model that represents a comment under a post		**Fields** • postId: string • commentId: string • parentCommentId: string | null • author: string • body: string • upvotes: number • downvotes: number

User		
Parent Class ● Model	**Sub Class** ● None	**Collaborations** ●
Responsibilities ● Database model that represents application users		**Fields** ● username: string ● email: string ● active: boolean ● password: string ● isModerator: boolean

Visited		
Parent Class ● Model	**Sub Class** ● None	**Collaborations** ● Course ● Exam ● Post ● User
Responsibilities ● Database model that represents the history of all previously visited courses, exams, or posts for a particular user		**Fields** ● email: string ● courses: string[] ● exams: string[] ● posts: string[]

Bookmark		
Parent Class ● Model	**Sub Class** ● None	**Collaborations** ● Course ● Exam ● Post ● User
Responsibilities ● Database model that represents either a course, exam, or post that the user has bookmarked.		**Fields** ● userEmail: string; ● type: string; ● courseCode: string; ● examId: string; ● postId: string;

Backend

App		
Parent Class	**Sub Class**	**Collaborations**

• None	• None	• Everything
Responsibilities • The main entry point into the backend API		

Server		
Parent Class • None	**Sub Class** • None	**Collaborations** • App
Responsibilities • Responsible for running the App on a specific port/host		

UsersController		
Parent Class • BaseController	**Sub Class** • None	**Collaborations** • User • Visited

Responsibilities
• Responsible for everything related to a users profile
  ○ Managing fields such as username, password etc
  ○ Account deletion
  ○ History management		

AuthController		
Parent Class • BaseController	**Sub Class** • None	**Collaborations** • User
Responsibilities • Responsible for handling authentication i.e. register, login, and logout		

ExamsController		
Parent Class • BaseController	**Sub Class** • None	**Collaborations** • Exam
Responsibilities • Primarily responsible for querying exam files and metadata from the database		

CoursesController		
Parent Class	**Sub Class**	**Collaborations**

• BaseController	• None	• Course
Responsibilities • Primarily responsible for querying course information from the database		

PostsController		
Parent Class • BaseController	**Sub Class** • None	**Collaborations** • Post • Comment • Tag
Responsibilities • Responsible for managing posts (Primarily basic CRUD)		

CommentsController		
Parent Class • BaseController	**Sub Class** • None	**Collaborations** • Post • Comment
Responsibilities • Responsible for managing comments (Primarily basic CRUD)		

Frontend

App Page		
Parent Class ● None	**Sub Class** ● None	**Collaborations** ● All pages
Responsibilities ● The entry point of the application into the frontend ● Responsible for routing the browser to different pages		

Register Page		
Parent Class ● None	**Sub Class** ● None	**Collaborations** ● Login Page
Responsibilities ● Provides a way for users to create accounts by filling out a form ● Will also supply a way for users to reset their forgotten password or resend account verification emails		

Login Page		
Parent Class ● None	**Sub Class** ● None	**Collaborations** ● Register Page
Responsibilities ● Provides a way for users to authenticate themselves by logging in to their accounts		

Navbar		
Parent Class ● None	**Sub Class** ● None	**Collaborations** ● All Pages
Responsibilities ● Provides an easy way for users to navigate to different pages in the app		

Dashboard Page		
Parent Class ● None	**Sub Class** ● None	**Collaborations** ● Course Search Page ● Course Page ● Navbar
Responsibilities		

● The page that users are redirected to after logging in ● Contains bookmarked/enrolled courses and recent activity

Course Search Page		
Parent Class ● None	**Sub Class** ● None	**Collaborations** ● Course Page ● Navbar
Responsibilities ● Clicking on an icon on the navbar takes users to this page ● Provides a search bar for users to search for courses ○ A paginated list of course results appear		

Course Page		
Parent Class ● None	**Sub Class** ● None	**Collaborations** ● Course Search Page ● Navbar ● Exams Page ● Posts Feed ● Piazza Posts Feed
Responsibilities ● After clicking on either a bookmarked or a search result course, the user is redirected to this page ● Contains a table with links to all exams for that specific course ● Contains a feed or posts related to the course ● Contains a link that redirects users to all exams for that specific course		

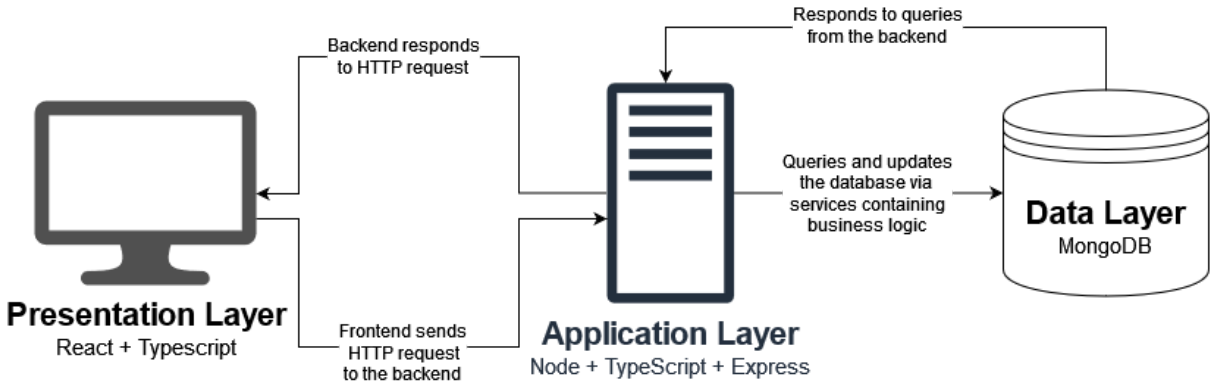
Exam Page		
Parent Class ● None	**Sub Class** ● None	**Collaborations** ● Course Page ● Navbar ● Exam Page ● Posts Feed
Responsibilities ● After a user is redirected to this page from the page of a specific course, the exam is displayed in an embedded pdf viewer ● Underneath the pdf viewer, there is a feed of posts that have been made related to that specific exam		

Posts Feed

Parent Class ● None	**Sub Class** ● None	**Collaborations** ● Course Page ● Exam Page
Responsibilities ● Responsible for displaying a feed of discussion posts related to a course or exam. ● Posts are displayed in a paginated manner and can be filtered or sorted		

Piazza Posts Feed		
Parent Class ● None	**Sub Class** ● None	**Collaborations** ● Course Page ● Exam Page
Responsibilities ● Responsible for displaying a feed of discussion posts scraped from piazza related to a course ● Posts are displayed in a paginated manner and can be filtered or sorted		

System Architecture Diagram



Reference

- https://www.linuxjournal.com/article/3508

Explanation

- Our application will follow the 3-tiered architecture
    - Presentation: Frontend with React and TypeScript
    - Application: Backend API with TypeScript, Node, and Express.js, and Python for simple scripts
    - Data Layer: Database primarily using MongoDB
- The frontend will only ever communicate with the backend which will act as a middleman between the frontend and the database

Error Handling & Exceptional Cases

- There will be two explicit layers of schema validation and one implicit layer.
    - The two explicit layers are:
        - Frontend form validation using the yup npm library
            - Primarily for a better UX
        - Backend API schema validation also using yup + TSOA
            - TSOA checks for the presence of required items and yup checks for properties of each field (i.e. a password is 8 characters long or an email follows a certain regex)
        - Database validation (on insertion)
            - We are using the mongoose ODM which allows us to define a structure/schema for a specific model. It takes care of validating all database objects on insertion or update queries.
- Generic error handling will make use of HTTP status codes and error messages