University of Dhaka

Department of Computer Science and Engineering

CSE-3112 : Software Engineering Lab

Project Title: **TestCraft**

Software Design Document

Developers:

Name: Md Emon Khan

Roll: 30

Name: Mahmudul Hasan

Roll: 60

Contents

1 Introduction

1.1 Purpose

TestCraft is an online test creating, giving and taking platform. The audience of this document are expected to be the developers, designers, stakeholders and anyone involved in the development process of the project. The purpose of this software design document is to present the audience with the outlines of the architecture and design of the project.

1.2 Scope

The process of teaching, for the teachers, sometimes can be more of a hassle. They need to prepare regular tests to keep the students checked. But setting up tests is a time consuming task. The teacher has to manage time for the students in the middle of his everyday task. So, **TestCraft** comes with a solution to make the task of test designing easy, so that the teaching body can relax all the while effectively managing his tutorship.

TestCraft makes test making easier by letting the tutors create their own problems database and sharing with others. This way, they can have a large number of problems in hand to choose from to create the most optimum problem set for the students. They can maintain the standard of the set with the ongoing academic calendar. They can have the students take practice tests, and in some cases, official quizzes or class tests when it's needed to be taken online. And with the AI assistant tool, the teachers can modify or create new problems based on some given problems. This way the task of modifying previously seen questions into newer ones become very convenient. This will save time for their personal aspects.

TestCraft also offers help to the students by letting them analyze their own progresses based on all the tests they've taken. The students can also create their own personal problems database, and practice tests to practice on. This way they can hone their skills and become more efficient for the upcoming tests.

1.3 Overview

The Software Design Document has the sole purpose of giving the **TestCraft** developers a written design schema. The document gives clear description of the system design by defining the entities, their relationships, the modules to handle the relationships.

1.4 Reference Material

- **About SDD:** Overview Software Documentation, Software Planning and Technical Documentation
- **Project Documents (RAD):** Project Documents

1.5 Definitions and Acronyms

- **RAD** - Requirement Analysis Document
- **MCQ** - Multiple Choice Question
- **API** - Application Programming Interface
- **REST** - REpresentational State Transfer

2 System Description

TestCraft is an online platform designed to provide the services of the creation, administration, and completion of tests. It provides educators and instructors with tools to create customized tests according to their curriculum and teaching objectives. It also offers students a user-friendly interface to access and complete tests to their convenience. It promotes self-assessment and enhances learning outcomes.

Functionality:

1. **Personal and Shared Database:** Each user can create their own collections of problems. They can share it with others to expand their collections. This way they can develop the problems and collections to optimize for their own needs. They can use these problems to create tests.
2. **Channels and Groups:** The users can create their own channels to create a teaching organization. They can then invite teachers and students into the channel. In the organization they can create groups. Students can then be divided into groups. Each group will facilitate its own test creation and announcement giving functionalities.
3. **Test Creation:** Educators can easily create tests using **TestCraft**'s intuitive interface. They can add various types of questions, including multiple-choice, short answer, and essay questions, and customize parameters such as test duration and scoring methods.

4. **Test Administration:** Once created, tests can be administered to students efficiently through **TestCraft** channels and channel based announcements. Educators can schedule tests, assign them to specific groups, and monitor test progress in real-time.

5. **Test-Taking:** Students can access assigned tests through their **TestCraft** channels and complete them according to the provided instructions. The platform supports various question types and provides features such as timer functionality and question navigation.

6. **Assessment and Analysis:** Upon completion, tests are assessed by Teachers, providing feedback to students. Detailed analytics and performance metrics are also available, allowing educators and students to assess student progress and identify areas for improvement.

Context: TestCraft is designed to meet the evolving needs of modern educational environments. Today digital tools play an increasingly integral role in teaching and learning. With the growing need of personalized education and continuous assessment, **TestCraft** gives a flexible and scalable solution that aligns with the best pedagogical practices.

Design: TestCraft 's design prioritizes usability, scalability, and performance. The platform follows a modular architecture, allowing for easy integration of new features and scalability to accommodate growing user bases. User interfaces are designed with simplicity and intuitiveness in mind.

Background: The concept of **TestCraft** arises from the recognition of the challenges faced by educators in creating and administering tests efficiently, as well as the growing demand among students for flexible and accessible assessment methods. **TestCraft** aims to address these challenges and makes it easier for the teachers to maintain the efficiency of their teaching job while saving time for personal aspects.

3 Design Overview

TestCraft application architecture is going to be a Three-Tier Architecture:

- **Presentation Tier(Frontend):** The presentation tier or the frontend, developed using the **ReactJS** framework, provides the user interface. It directly communicates with the client and the backend server making a bridge between the client and the application.

- **Application Tier(Backend):** The backend acts as the logic tier. It processes requests from the presentation tier, performs necessary

computations, and interacts with the database tier to retrieve or update data. The backend is to be implemented using **NodeJS** and **Express JS**.

- **Data Tier (Database):** This tier stores and manages the data used by the application. It is a relational database system implemented using **MySQL**. The database server is responsible for storing and managing all the data of the users and application functionalities efficiently.

3.1 Design Rationale

The decision to adopt a Three-Tier Architecture for **TestCraft** application was made after careful considerations of various factors, including scalability, maintainability, and performance. Here's the rationale behind this choice:

1. **Scalability:** By separating the system into three tiers - presentation, application, and data - we can scale each tier independently based on its specific requirements. This allows us to handle increasing loads more effectively, as we can allocate resources as needed to each tier without affecting the others.

2. **Maintainability:** The three-tier architecture promotes modularity and separation of concerns, making it easier to maintain and update the system over time. Changes to one tier can be made without impacting the others, reducing the risk of unintended side effects.

3. **Performance:** With a dedicated backend tier responsible for processing requests and interacting with the database, we can optimize performance by offloading computationally intensive tasks from the frontend. This helps ensure a responsive user experience even under heavy loads.

4. **Technology Compatibility:** The selected technologies - **ReactJS** for the frontend, **NodeJS** and **Express JS** for the backend, and **MySQL** for the database - are well-established and widely used in the industry.

5. **Trade-offs and Considerations:** While the three-tier architecture provides numerous benefits, it also has some trade-offs and challenges. For example, maintaining consistency and synchronization between tiers can be complex, especially in distributed systems. Additionally,

deploying and managing multiple tiers may require more infrastructure resources and operational overhead compared to simpler architectures like monolithic or client-server.

Other architectures, such as Microservices or Serverless, were considered but ultimately not chosen for **TestCraft**. Microservices architecture could introduce additional complexity and overhead in managing a large number of services. And the Serverless architecture might not provide enough control over the underlying infrastructure and execution environment. Additionally, given the relatively straightforward nature of the application and its requirements, a three-tier architecture was deemed to be a suitable balance between scalability, maintainability, and performance.

3.2 System Architecture

TestCraft's system architecture follows a modular program structure. It consists of three high-level subsystems, frontend, backend, and database, each responsible for specific functionalities. These subsystems collaborate to achieve the overall functionality of the system. Then we've introduced some extra subsystems inside the backend. Each subsystem communicates with the database using various classes, and uses the **Express JS** app server to communicate with the frontend via HTTP requests. Below is an overview of the major subsystems in the backend, besides the frontend and the database, and their roles:

1. **User Management Subsystem:**
    - **Responsibilities:** Handles user authentication, authorization, and profile management.
    - Collaborates with other subsystems by providing user authentication and authorization services, ensuring that only authorized users can access the system's functionalities.
2. **Collection Management Subsystem:**
    - **Responsibilities:** Manages the creation, edition, and sharing of personal problemsets
    - Collaborates with the users to provide the authorized users to interact with their collections
    - Provides problems that may be used in test creation

3. **Channel Mangement Subsystem:**

    - **Responsibilities:** Provides channel based services for inviting and adding other users, managing a students and teachers body, giving regular tests and announcements.
    - Collaborates with the User Management Subsystem to add different users to the channel and assigns them roles.
    - Also collaborates with the Test Creation Subsystem to provide test creation tools.
    - Provides Announcements and interactions on the announcements.

4. **Test Creation Subsystem:**

    - **Responsibilities:** Manages the creation, editing, and scheduling of tests by educators.
    - Collaborates with the User Management Subsystem to ensure that only authorized educators can create and manage tests.
    - Provides interfaces for educators to create and customize tests according to their requirements.

5. **Test-Taking Subsystem:**

    - **Responsibilities:** Facilitates the administration of tests to students and allows them to complete tests.
    - Collaborates with the User Management Subsystem to authenticate students and ensure that they have access to assigned tests.
    - Provides a user-friendly interface for students to access and complete tests, including features such as question navigation and time tracking.

6. **Assessment and Analysis Subsystem:**

    - **Responsibilities:** Lets the teachers to assess the completed tests and provides analytics and performance metrics to educators and the students.
    - Collaborates with the Test Management Subsystem to retrieve test data and with the User Management Subsystem to authenticate educators.
    - Analyzes student responses, generates progress reports, and presents detailed analytics to educators and the student for assessment purposes.

Interconnections: The subsystems interact with each other through well-defined interfaces. The connect with the frontend through REST API calls. For example:

- The Test Management Subsystem communicates with the backend server to retrieve changes or any other information from the User Management Subsystem to verify educator credentials before allowing test creation.
- In a similar fashion, the Test-Taking Subsystem interacts with the User Management Subsystem to authenticate students and retrieve assigned tests.
- Similarly, the Assessment and Analysis Subsystem retrieves test data from the Test Management Subsystem and provides grading and analytics services to educators.

The interactions method will be clear to the audience from the diagram provided below

Diagram:

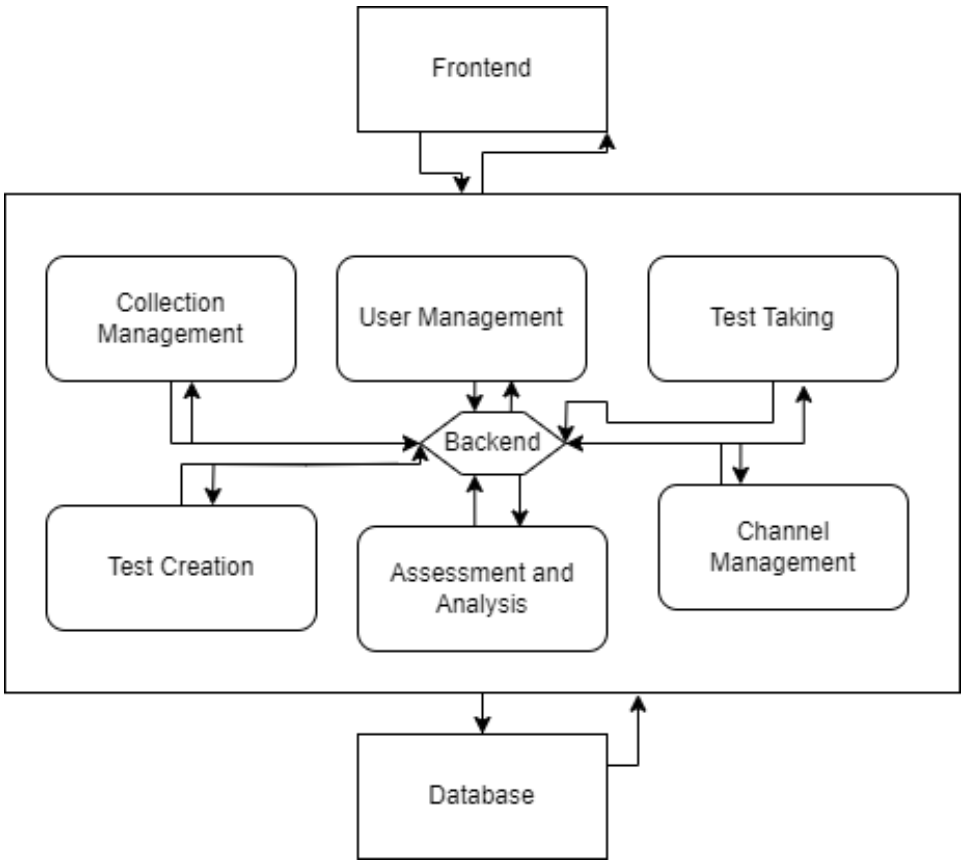

Figure 1: High Level Subsystem Interaction Diagram

The diagram illustrates the high-level subsystems of **TestCraft** and their interconnections. The frontend subsystem sends a request to the backend subsystem that consists of six modules. Then the backend communicates with the database subsystem to execute an operation. Each subsystem communicates with others as necessary to fulfill its responsibilities, ultimately contributing to the overall functionality of the system. The communications are done indirectly through the database server and the backend server ensuring scalability and maintainability.

3.3 Constraint and Assumptions

3.3.1 List of Assumption

- **Scalability: TestCraft** assumes that the platform will need to accommodate a large number of users, including educators and students, without compromising performance or user experience.
- **Security:** It is assumed that the best security measures will be implemented to safeguard user data, including encryption of sensitive information and protection against unauthorized access.
- **Compatibility: TestCraft** assumes compatibility with modern web browsers and devices to ensure accessibility for users across various platforms.
- **Reliability:** It is assumed that the platform will maintain high availability and reliability, minimizing downtime and service interruptions.
- **User Engagement: TestCraft** assumes that user engagement features, such as notifications and reminders, will enhance user interaction and adoption of the platform.

3.3.2 List of Dependencies

- **Database System: TestCraft** is dependent on a reliable database system to store user data, test configurations, and performance metrics. Any issues with the database system, such as data corruption or downtime, would impact the functionality of the platform.
- **Authentication Service:** The platform relies on an internal authentication service or mechanism to verify user identities and grant access to authorized users. Integrity of this service is crucial for user authentication and authorization processes.
- **External Libraries/Frameworks: TestCraft** may depend on third-party libraries or frameworks for functionalities such as user interface components, data visualization, or analytics. Any changes or issues with these dependencies could affect the performance and behavior of the platform.
    - **Frontend Dependecies:**
        - **js-cookie:** Version ^3.0.5

- **lucide-react:** Version ^0.364.0
- **react-router-dom:** Version ^6.22.3
- **react-katex:** Version ^3.1.3

- **Backend Dependencies:**
  - **body-parser:** Version ^1.20.2
  - **cookie-parser:** Version ^1.4.6
  - **cors:** Version ^2.8.5
  - **cross-env:** Version ^7.0.3
  - **express:** Version ^4.18.2
  - **mysql2:** Version ^3.9.3
  - **vite-express:** *

- **Network Infrastructure:** The platform's functionality is dependent on a stable and high-speed network infrastructure to establish communication between subsystems and ensure timely data exchange. Poor network connectivity or bandwidth limitations may degrade system performance.

- **Browser Compatibility: TestCraft** relies on web browsers to deliver its user interface to educators and students. Compatibility with a wide range of browsers, devices, and versions is necessary to ensure consistent user experience across different devices and platforms.

4 Object Model

4.1 Object Description

- **User:**
  - **Class Name:** user
  - **Attributes:**
    - **userID:** the unique identity of the user
    - **authToken:** the authentication token of the user
  - **Methods:**
    - **GetUser:** Gets a user and their details
    - **LoginUser:** Authenticates a user
    - **LogoutUser:** Removes all authentication from the database
    - **RemoveUser:** Removes the user from the database

        - **IsAuthorized:** Checks if the user is authorized with the credentials
        - **EditUserDetails:** Edits user's personal details
        - **ChangePassword:** Changes the user's password

- **Collection:**
    - **Class Name:** collection
    - **Attributes:**
        - **collectionID:** The unique id of the collection
        - **ownerID:** The unique id of the collection owner
    - **Methods:**
        - **AddCollection:** Adds a collection to the database
        - **RemoveCollection:** Removes collection from database
        - **GetCollection:** Gets a collection
        - **AddProblem:** Adds a problem to a collection
        - **GetProblems:** Gets problems

- **Channel:**
    - **Class Name:** channel
    - **Attributes:**
        - **channelID:** The unique id of the channel
        - **ownerID:** The unique id of the channel owner
    - **Methods:**
        - **AddChannel:** Adds a channel to the database
        - **RemoveChannel:** Removes channel from database
        - **GetChannel:** Gets a channel
        - **AddUserAs:** Adds a user to the channel as teacher or student
        - **IsTeacher:** Checks if a user is a teacher in the channel
        - **IsStudent:** Checks if a user is a student in the channel
        - **GetAnnouncements:** Gets the announcements in the channel
        - **GetTeachers:** Gets the teachers in that channel
        - **GetStudents:** Gets the students in that channel
        - **GetGroups:** Gets the groups in a channel

- **Invitation and Request:**
  - **Class Name:** invreq
  - **Methods:**
    - **AddInvitation:** Adds an invitation to the channel
    - **CancelInvitation:** Cancels an invitation
    - **AddJoinRequest:** Creates a join request to join in the channel
    - **RemoveJoinRequest:** Deletes a join request
- **Group:**
  - **Class Name:** group
  - **Attributes:**
    - **groupID:** Unique ID of the group
    - **creatorID:** The unique id of the creator
  - **Methods:**
    - **CreateGroup**: Creates a group in the channel
    - **AddUserToGroupAs:** Adds a user to group as teacher or student
    - **DeleteGroup:** Removes a group from the channel
    - **IsTeacher:** Checks if the teacher has access to the group
    - **AddTeacher:** Adds a teacher to group
    - **AddStudent:** Adds a student to group
    - **GetAnnouncements:** Gets the announcements in the group
    - **IsStudent:** Checks if the student has access to the group
- **Problem:**
  - **Class Name:** problem
  - **Attributes:**
    - **problemID:** The unique id of a problem
    - **creatorID:** The unique id of the creator
  - **Methods:**
    - **IsMCQ:** Checks if the requested problem is an MCQ or not
    - **AddProblem:** Adds a new Problem to the database
    - **RemoveProblem:** Removes problem from database

        - **GetProblem:** Gets a problem
        - **GetProblemsByCollection:** Gets the problems in a specific collection
        - **GetProblemsByUser:** Gets the problems created by a user
        - **FilterProblem:** Filters with various parameters, such as, subject, date created, creator etc to get a problem
        - **AddSolution:** Adds a solution to a problem

- **Test:**
    - **Class Name:** test
    - **Attributes:**
        - **testID:** The unique id of a test
        - **creatorID:** The unique id of the creator
    - **Methods:**
        - **AddTest:** Adds a new test to the database
        - **RemoveTest:** Removes test from database
        - **GetTest:** Gets a Test
        - **GetTestByChannel:** Gets the problems in a specific collection

- **Announcement:**
    - **Class Name:** announcement
    - **Attributes:**
        - **announcementID:** The unique id of an announcement
        - **channelID:** The unique id of the channel the announcement is part of
        - **creatorID:** The unique id of the creator of the announcement
    - **Methods:**
        - **AddAnnouncement:** Adds a new announcement to the database
        - **DeleteAnnouncement:** Removes announcement from database
        - **EditAnnouncement:** Edits an announcement details
        - **GetAnnouncementsByChannel:** Gets the announcements in a specific channel

        - **AddComment:** Adds a comment to an announcement

- **Advertisement:**
    - **Class Name:** advertisement
    - **Attributes:**
        - **advertisementID:** The unique id of the advertisement
        - **cannelID:** The unique id the advertisement is of
    - **Methods:**
        - **AddAdvertisement:** Adds a new advertisement to the database
        - **DeleteAdvertisement:** Removes advertisement from database
        - **EditAdvertisement:** Edits an advertisement details
        - **GetAdvertisementByChannel:** Gets the advertisement of a specific channel
        - **FilterAdvertisements:** Gets the advertisements with various filters

4.2 Object Collaboration Diagram

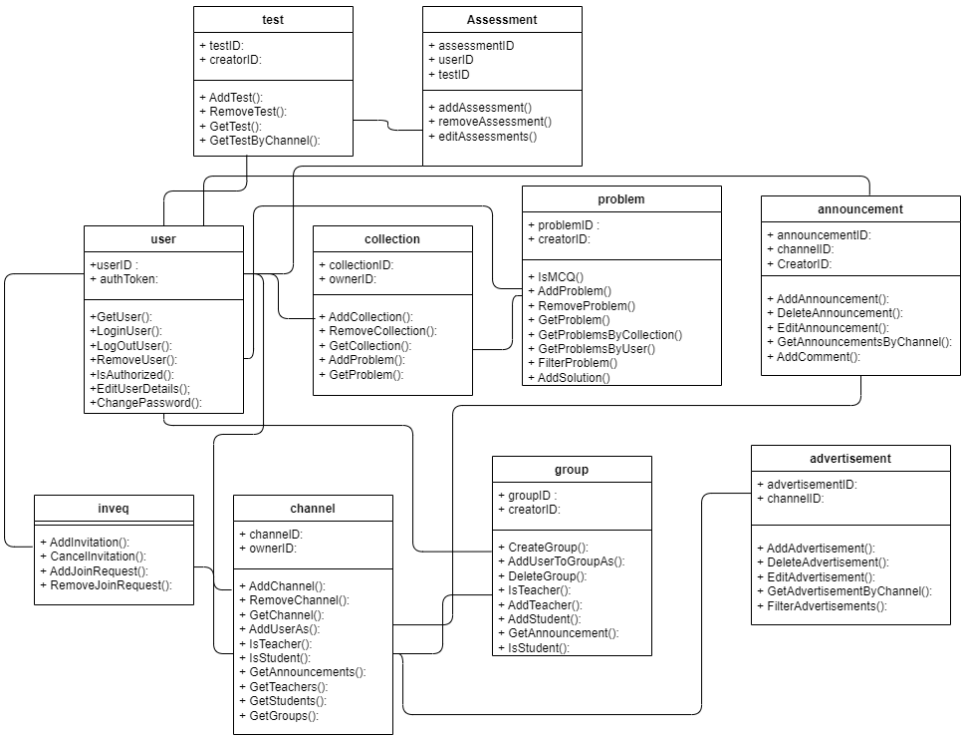

Figure 2: Object Collaboration Diagram

5 Subsystem Decomposition

5.1 Complete Package Diagram

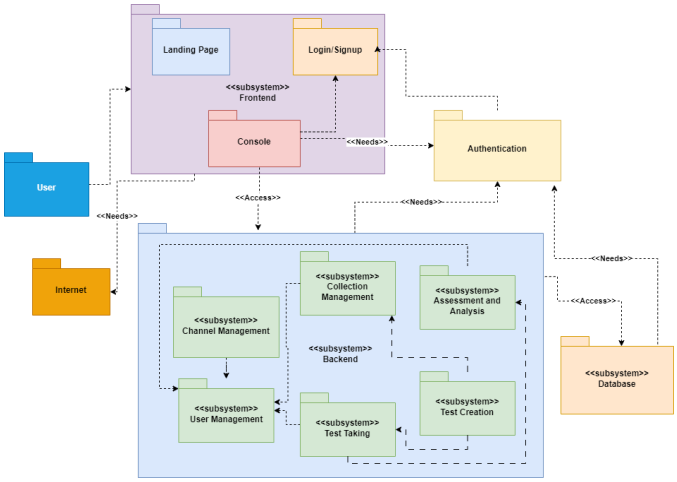

Figure 3: Complete Package Diagram

5.2 Subsystem Detail Description

1. **User Management Subsystem**

   - **Module Description:** Handles user authentication, authorization, and profile management. Collaborates with other subsystems by providing user authentication and authorization services, ensuring that only authorized users can access the system's functionalities
   - **Class Diagram:**

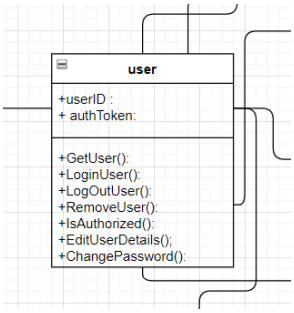

Figure 4: User Management Subsystem

- **Subsystem Interface:** The user management subsystem provides the user with various user details edition perspectives:
  - A user can create a new account
  - Users can delete their account
  - The users can login to their accounts
  - A user may change/update their personal details
  - Share with other users their own collections
  - Find users and invite them
  - Assign a user to a role in the channel

2. **Collection Management Subsystem**

- **Module Description:** Manages the creation, edition, and sharing of personal problemsets. Collaborates with the users to provide the authorized users to interact with their collections. Provides problems that may be used in test creation
- **Class Diagram:**

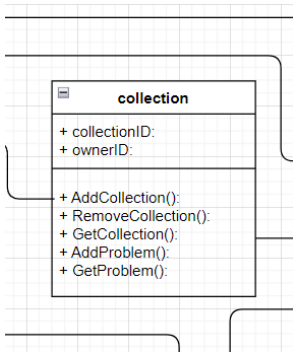

Figure 5: Collection Management Subsystem

- **Subsystem Interface:**
  - Creation of a collection request
  - Fill up a form that describes the details of the collection
  - Add a problem to the collection
  - Share the collection with other users
  - Edit collection
  - Remove collection

3. **Channel Management Subsystem**

   - **Module Description:** Provides channel based services for inviting and adding other users, managing a students and teachers body, giving regular tests and announcements. Collaborates with the User Management Subsystem to add different users to the channel and assigns them roles. Also collaborates with the Test Creation Subsystem to provide test creation tools. Provides Announcements and interactions on the announcements.
   - **Class Diagram:**

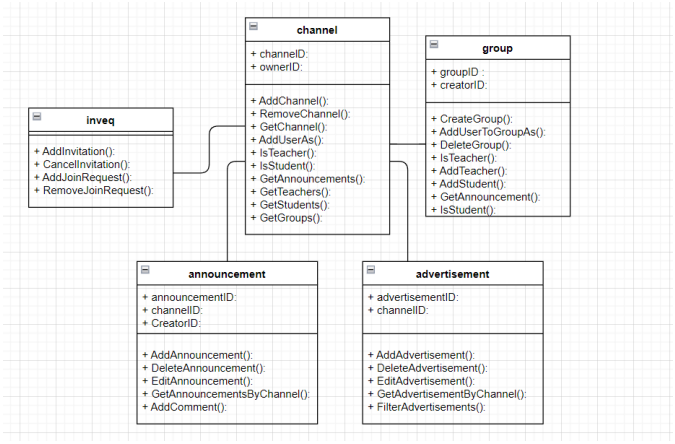

Figure 6: Channel Management Subsystem

- **Subsystem Interface:**
  - Create a channel
  - Fill up a form that describes the details of the channel
  - Invite others into channel as Teacher or Student
  - Create an announcement in the channel
  - Create Groups within the channel
  - Create announcement into the group of a channel
  - Add a student or teacher to the channel
  - Add a new test announcement
  - Delete announcements
  - Advertise the channel

4. **Test Creation Subsystem**

   - **Module Description:** Manages the creation, editing, and scheduling of tests by educators. Collaborates with the User Management Subsystem to ensure that only authorized educators can create and manage tests. Provides interfaces for educators to create and customize tests according to their requirements
   - **Class Diagram:**

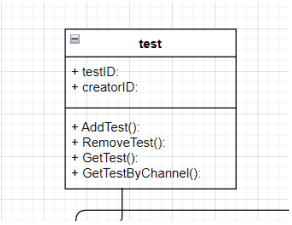

Figure 7: Test Creation Subsystem

- **Subsystem Interface:**
  - Create a Test by adding problems to the test
  - Announce the test in an announcement in the channel
  - Schedule the test announcement
  - Write new problem into test
  - Add, edit, remove problems in the test

5. **Test Taking Subsystem**

- **Module Description:** Facilitates the administration of tests to stu- dents and allows them to complete tests. Collaborates with the User Management Subsystem to authenticate students and ensure that they have access to assigned tests. Provides a user-friendly interface for students to access and complete tests, including features such as question navigation and time tracking.
- **Class Diagram:**

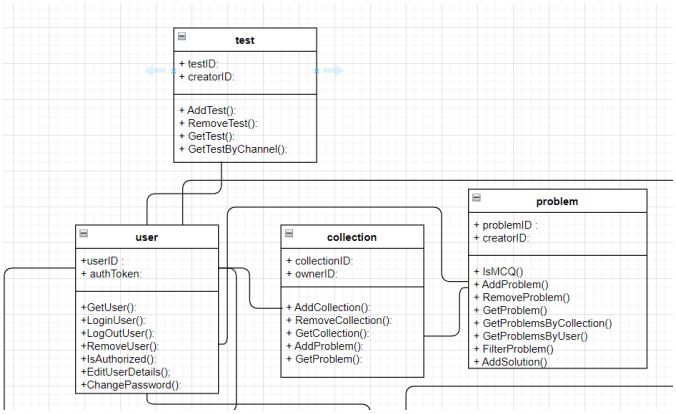

Figure 8: Test Taking Subsystem

- **Subsystem Interface:**
  - Give answers to the question
  - Insert necessary files in each question
  - Ask or complaint about a question
  - Submit the answers

6. **Assessment and Analysis Subsystem**

- **Module Description:** Lets the teachers to assess the completed tests and provides analytics and performance metrics to educators and the students. Collaborates with the Test Management Subsystem to retrieve test data and with the User Management Subsystem to authenticate educators. Analyzes student responses, generates progress reports, and presents detailed analytics to educators and the student for assessment purposes
- **Class Diagram:**

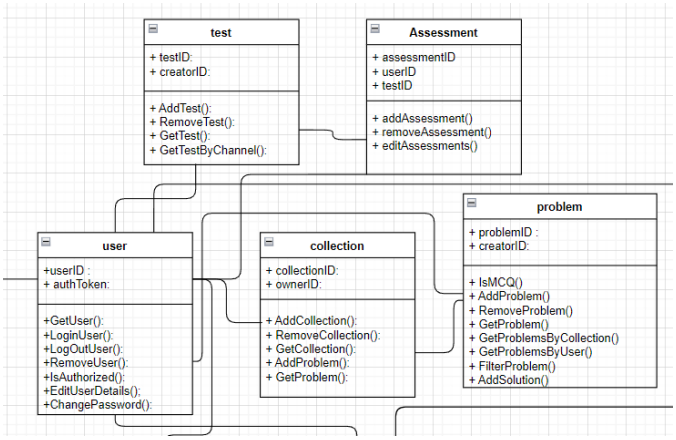

Figure 9: Assessment and Analysis Subsystem

- **Subsystem Interface:**
  - Open a specific test and see the answers of the students
  - Check answerscripts and give an assessment
  - Analayze the test results

6 Data Design

6.1 Data Description

As we are using **MySQL**, a relational database, we created relational tables, and implemented the database functionalities using various functions and procedures. Each of the modules are capable of querying the database via the classes to store and gather information and execute the logical operations. Data are stored in the data tier or the database as follows:

- **Relational Database:** The database should be created using the principles and best practices of a relational database. Each entity consists of atomic attributes. Following the principles of normalization, various foreign key and other constraints are made. Some relational tables are made to create relations among various entities to give the application the necessary functionalities.

- **Unique ID:** Each entity has its own data table in the database. Each record in each entity table, has a unique id. This id is used to create relational data tables and other functionalities in the database following the principles of normalization. All the data tables should follow the principles of data integrity by reducing data redundancy.

- **Creation of Data:** When a new instance of an entity is requested to be created by the client in the frontend server, the frontend sends the request to the backend. The backend then calls the necessary functions defined in the modules and classes. Each module can query the database using necessary procedure or function or SQL statement and extract the necessary results.

  For example: When a new user account is opened, the frontend sends a request with the new user information to the backend. The backend receives the request in the User Management Subsystem. The subsystem interacts with necessary modules and classes that can communicate with the database server. The user module then sends a request to the Database server by calling the adduser procedure that adds a new user to the database using the provided information.

  Each of the entities should have their necessary procedures, and functionalities defined in the database

- **Retrieval of Data:** The client requests through the frontend a list of the necessary data. This request is similarly received by one of the

Subsystems which then executes the logic using the modules to get the data.

For example: When a client requests all the physics problem of a collection, the Collection Management Subsystem receives it. Then it interacts with the User Management Subsystem to check if the user is authorized to see the collection. When confirmed, the Subsystem then uses the modules to execute the query to the server and then processes the results to send to the frontend server.

Necessary relationships should be formed among the entities using necessary relational tables. And to perform a query according to the need of the frontend, the database should implement necessary structured or view queries.

- **Creating User Functionality by Interaction between Entities:** The entities interact through relational data tables. The database should implement necessary operations in the procedures to keep data integrity and ease of access. And functionalities such as user accessibility should be implemented with careful thought.
- **Authentication:** This is a special functionality which executes when a new user is created or the user logs in.
    - When a new user account is created, the user only needs email, phone, and a username to create a user. Other personal details are to be added later.
    - When the new user record is being created in the database, the user's password is to be saved after hashing using a powerful hashing algorithm.
    - When a user logs in using their password, the database should be able to rehash the provided password to match the previously saved hash to check if they are authorized
    - Upon authorization, the database should be able to send the userID and a unique authentication token to the frontend.
    - Each request from the frontend should contain the credentials
    - The backend should be able to confirm the validity of the credentials before it proceeds to execute an operation
- **Removal of Data:** The server should be able to maintain data integrity when an entity, say user, or channel, has been deleted.

The principles and best practices of database management are expected in the process of storing and management of the data to avoid any security failure or integrity loss. To ensure scalability, separate relational tables are made to keep the records of the interaction between the entities.

6.2 Data Dictionary

- **Advertisement:** The advertisement entity gives the advertisement details of the channel
  - **Attributes:**
    - **id:** char(36)
    - **headline:** varchar(50)
    - **subtitle:** varchar(50)
    - **background:** varchar(255)
    - **channelID:** char(36), foreign key
    - **creationTime:** datetime
    - **lastUpdateTime:** datetime
  - **Methods:**
    - **AddAdvertisement:** headline, subtitle, background, channelID
    - **DeleteAdvertisement:** channelID
    - **EditAdvertisement:** channelID, updateArray
- **Announcement:** Announcements can be made inside a channel that describes an important announcement or a contains a test
  - **Attributes:**
    - **id:** char(36)
    - **title:** varchar(100)
    - **postdesc:** longtext
    - **isTest:** boolean
    - **creatorID:** char(36), foreign key
    - **channelID:** char(36), foreign key
    - **groupID:** char(36), foreign key
    - **creationTime:** datetime
    - **lastEdit:** datetime
  - **Methods:**

        - **AddAnnouncement:** title, postdesc, isTest, creatorID, channelID, groupID
        - **DeleteAnnouncement:** channelID
        - **EditAnnouncement:** channelID, updateArray

- **Channel:** Channels can be created by the users. They can do various tasks in a channel such as giving announcements, test announcements, inviting others etc.
    - **Attributes:**
        - **id:** char(36)
        - **channelName:** varchar(20)
        - **channelOwner:** char(36), foreign key
        - **dateCreated:** datetime
    - **Methods:**
        - **AddChannel:** channelName, channelOwner
        - **DeleteAnnouncement:** channelID
        - **AddUserToChannelAs:** channelID, userID, role
        - **RemoveUserFromChannel:** channelID, userID

- **Collections:** Users can create collection and add problems to the collection, share with others
    - **Attributes:**
        - **id:** char(36)
        - **collectionName:** varchar(20)
        - **ownerID:** char(36), foreign key
        - **dateCreated:** datetime
        - **lastUpdate:** datetime
    - **Methods:**
        - **AddCollection:** collectionName, ownerID
        - **DeleteCollection:** collectionID
        - **ShareWithUser:** collectionID, userID
        - **RemoveUserShare:** collectionID, userID
        - **AddProblem:** collectionID, problemID
        - **RemoveProblem:** collectionID, problemID

- **Groups:** Groups can be created inside a channel. Groups have the similar functionalities as a channel
    - **Attributes:**
        - **id:** char(36)
        - **groupName:** varchar(20)
        - **GroupPhoto:** varchar(255)
        - **channelID:** char(36), foreign key
        - **creatorID:** char(36), foreign key
        - **dateCreated:** datetime
    - **Methods:**
        - **AddGroup:** groupName, groupPhoto, channelID, creatorID
        - **DeleteGroup:** groupID
        - **AddUserToGroup:** groupID, userID
        - **RemoveUserFromGroup:** groupID, userID
- **Problem:** Problems can be MCQ or descriptive
    - **Attributes:**
        - **id:** char(36)
        - **subj:** varchar(20)
        - **topics:** varchar(255)
        - **probDesc:** longtext
        - **solution:** longtext
        - **creatorID:** char(36)
        - **creationTime:** datetime
        - **lastEdit:** datetime
    - **Methods:**
        - **AddProblem:** subj, topics, probDesc, solution, creatorID
        - **DeleteProblem:** problemID
        - **EditProblem:** problemID, editArray
- **Test:** Test can be created using either existing problems or new problems
    - **Attributes:**
        - **id:** char(36)

- **title:** varchar(20)
- **subj:** char(20)
- **topics:** varchar(255)
- **startTime:** datetime
- **endTime:** datetime
- **totalMarks** int
- **creatorID** char(36), foreign key
- **channelID** char(36), foreign key
- **groupID** char(36), foreign key
- **announcementID** char(36), foreign key
- **creationTime** datetime

- **Methods:**
  - **AddTest:** title, subj, topics, startTime, endTime, totalMarks, creatorID, channelID, groupID, announcementID
  - **DeleteTest:** testID
  - **AddProblem:** testID, problemID
  - **RemoveProblem:** testID, problemID

- **User:** Users are the entities to provide clients a means to interact with the application as an identifiable user.
  - **Attributes:**
    - **id:** char(36)
    - **username:** varchar(50)
    - **psswrd:** varchar(255)
    - **email:** varchar(100)
    - **phone:** varchar(20)
    - **dateCreated:** datetime
  - **Methods:**
    - **AddUser:** username, psswrd, email, phone
    - **DeleteUser:** userID
    - **EditDetails:** userID, details

6.3 Entity Relationship Diagram

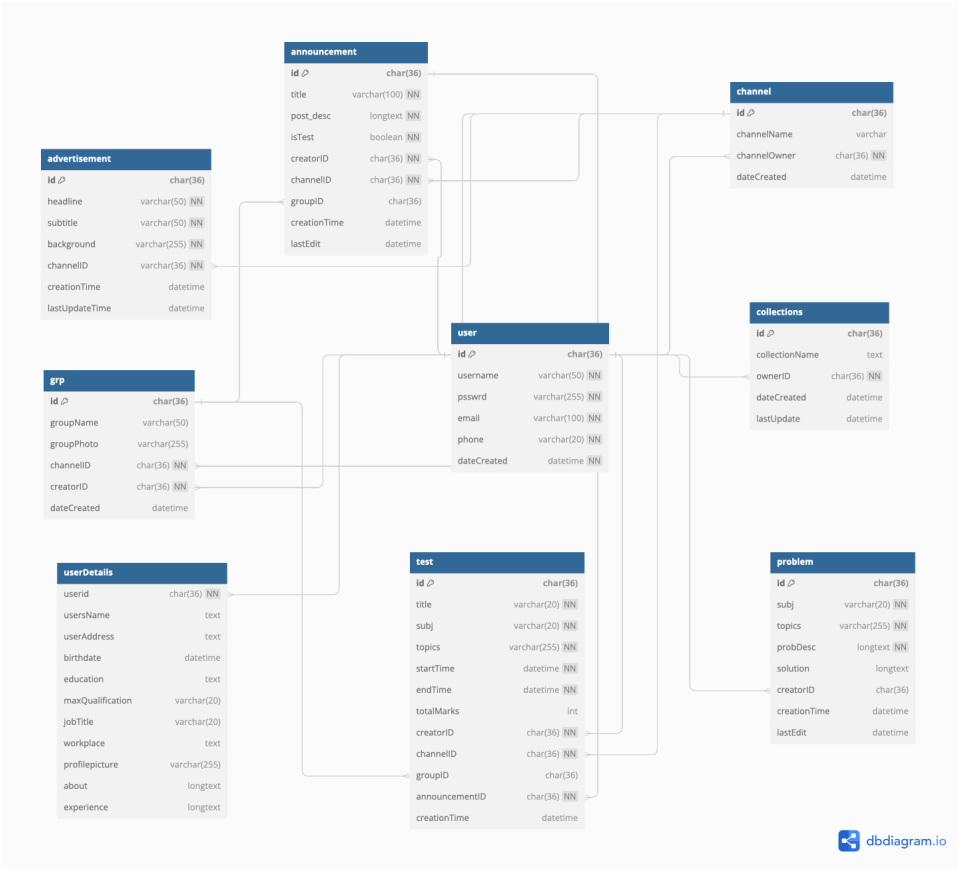

Figure 10: ER Diagram

7 User Requirement and Component Traceability Matrix

	User Management	Collection Management	Channel Management	Test Creation	Test Taking	Assessment and Analysis
Authentication	-	-	-	-	-	-
Personal Problem Database	-	-	X	-	-	-
Channel	-	X	-	-	-	-
Group	-	X	-	-	-	
Test Crafting	-	-	-	-	-	X
Test Giving	-	-	-	-	-	-
Test Taking	-	-	-	-	-	-
Assessment	-	-	-	-	-	-
Self Analysis	-	-	-	-	-	-