Arab International University

Faculty of Informatics and Communication Engineering

Junior Project Report on

SQL Mentor

Submitted to

Department of Informatics Engineering

in partial fulfillment of the requirement for the Degree of Bachelor in

Informatics Engineering

Submitted by

Eyad Hassan Bargouth **Ahmad Refaat Kitaz**

Muhammad Zaid Al Nahhas

Under the Supervision of

Eng. Nisreen Al-Madi

February 2025

February 2025

Faculty of Informatics & Communication Engineering

CERTIFICATE OF APPROVAL

The undersigned certify that they have read and recommended to the Department of Informatics Engineering for acceptance, a project report entitled **SQL Mentor** Submitted by: **Eyad Hassan Bargouth, Ahmad Refaat Kitaz, Muhammad Zaid Al Nahhas** in partial fulfilment for the degree of Bachelor of Engineering in Informatics.

Supervisor	**Head of Department**
Eng. Nisreen Al-Madi	**Dr. Said Desouki**

Arab International University

The Arab International University (AIU) is a private Syrian university established in 2005. Its academic plans and the documents issued by it are approved and certified by the Ministry of Higher Education in the Syrian Arab Republic.

The university works to achieve the following goals:

- Preparing a distinguished generation of university graduates who are able to meet and advance the specific needs of society.
- Contributing to theoretical and applied scientific research that serves the purposes of national development. Work is being done to urge professors and academic staff to scientific research and participate in conferences and seminars that organize research.
- Achieving partnership with prestigious Arab and foreign universities with the aim of continuous development and modernization of academic work and conducting joint scientific research.
- Attracting distinguished academic and research competencies by providing the appropriate environment for their work.

The Arab International University is one of the first Syrian universities that have been established and inaugurated. It has been able to attract distinguished educational, research and administrative competencies, to create an integrated edifice from the academic, organizational and administrative aspects. It was able to graduate cadres of distinguished innovators by providing an educational environment based on unique qualitative and material ingredients, including:

- Modern and advanced study plans based on the credit hour system.
- Carefully selected educational cadres.
- Modern scientific laboratories and a laboratory for electronic libraries.
- Physical and moral incentives for students.
- Application of interactive teaching methods.
- Academic and educational guidance and counseling.

- A wide range of scientific cooperation agreements with local, regional and international universities of reputable reputation.
- Multiple agreements and memoranda of understanding with many civil society organizations.
- A proper campus with all the facilities of science, sports and entertainment, which we encourage you to visit and learn about their features.
- Student activities and clubs of all kinds: athletic, cultural, scientific and social.

The Arab International University life years are a time to invest in a student's future. The knowledge and experience that students acquire in the lecture hall and laboratories will help them in developing themselves. They will provide them with reasons for success in the chosen specialty. The student activities will help in expanding students' horizon. The activities of training, clubs and sports will enable students to develop their talents, and may even help in discovering new talents.

Students could invest time, mind and spirit in our university in order to reap the benefits of works and the time devoted in the coming years. We will be by our students in every step of their way.

Acknowledgements

It gives us immense pleasure to express our deepest sense of gratitude and sincere thanks to our highly respected and esteemed guide Eng. **Nisreen Al-Madi** for her valuable guidance, encouragement and help for completing this work, her useful suggestions for this whole work and cooperative behavior are sincerely acknowledged.

At the end we would like to express our sincere thanks to all our families, friends and others who helped us directly or indirectly during this project work.

Abstract

This project aims to design and develop an interactive SQL learning platform that focuses on enhancing user engagement and improving knowledge retention. The platform offers a comprehensive learning experience through structured levels, interactive quizzes, and an AI-powered error analysis system. These features are designed to bridge the gap between theoretical knowledge and practical application in SQL programming.

The platform organizes content into progressive levels, starting from the basics and advancing to more complex concepts, allowing users to learn SQL in a systematic way. Additionally, it includes interactive quizzes that help users assess their understanding and receive immediate feedback for improvement. The AI-based error analysis system identifies common mistakes and provides personalized suggestions, enabling users to learn from their errors and strengthen their skills.

By combining structured learning, practical exercises, and advanced feedback mechanisms, this platform aims to create an engaging and effective environment for mastering SQL, catering to both beginners and advanced learners.

Contents

Abbreviations

AIU	Arab International University
SQL	Structured Query Language
DB	Database
NoSQL	Not Only SQL
ORM	Object-Relational Mapping
API	Application Program Interface
UI	User Interface
ERD	Entity Relationship Diagram
CRUD	Create, Read, Update, Delete

Keywords

- **SQL Queries**
- **Database Design**
- **Data Integrity**
- **Database Management Systems**
- **ER Diagrams**

Introduction

With the growing demand for database management skills in today's data-driven world, SQL (Structured Query Language) has become an essential programming language across industries ranging from technology and finance to healthcare and retail. SQL allows professionals to efficiently manage and manipulate data in relational databases, making it a critical skill for anyone working with large datasets or building data-driven applications. As more organizations rely on SQL to manage their databases, the need for skilled professionals who are proficient in SQL continues to rise.

Traditional methods of learning SQL, such as textbooks and online tutorials, often fail to engage learners effectively. These methods typically offer a one-size-fits-all approach, providing static content without the opportunity for interaction or personalized feedback. Without the chance to practice and apply the knowledge in a real-time environment, learners may struggle to retain and fully understand key SQL concepts. This lack of interactivity and tailored instruction can lead to a frustrating learning experience, especially for beginners or those at an intermediate level who need more targeted guidance.

This project addresses these challenges by creating an innovative and interactive SQL learning platform that combines theoretical content with hands-on practice. By integrating instructional material, quizzes, and real-time error feedback, the platform provides learners with a dynamic and engaging experience. It offers an intuitive user interface where students can practice writing SQL queries, test their knowledge through quizzes, and receive instant feedback on their progress.

The platform is designed to cater to learners of all levels, with content and exercises that adapt to the user's individual skill set. For beginners, the platform introduces fundamental concepts and helps build a strong foundation in SQL. As learners advance, the platform provides more complex challenges, ensuring that users progress at their own pace while continuously refining their skills. The inclusion of real-time error feedback helps learners quickly identify and correct mistakes, fostering a deeper understanding of the language and its syntax.

Ultimately, this SQL learning platform aims to bridge the gap between traditional, passive learning methods and more effective, interactive education. By offering a comprehensive, personalized, and engaging learning experience, it empowers users to master SQL and apply their skills in real-world situations.

Chapter 1: Project Description

1.1 Background

SQL (Structured Query Language) is a fundamental skill for managing and querying databases. Current learning platforms often overlook the importance of hands-on practice and interactive feedback, leaving learners unprepared for real-world challenges.

1.2 Problem Statement

Many SQL learners struggle with applying theoretical knowledge to practical scenarios. A lack of immediate feedback and error analysis further hampers their learning process.

1.3 Project Objective

The primary objective of this project is to develop an interactive SQL learning platform. Key features include structured lessons, quizzes, an AI-driven error analysis tool, and a chatbot for syntax assistance.

1.4 Project Scope

The platform is designed for students, professionals, and anyone interested in learning SQL. It will include:

- A tiered learning system with progressively challenging content.
- Real-time feedback on SQL queries.
- A syntax assistance chatbot.
- Analytics to track user progress.

1.5 Project Features

- Interactive quizzes with instant feedback.
- AI-driven error detection and suggestions.
- A chatbot for syntax queries.

Chapter 2: Theoretical Study

2.1 Introduction

In today's digital age, databases are a fundamental component of various applications, including web platforms, mobile apps, and enterprise systems. With the growing demand for data analysis and big data management, learning Structured Query Language (SQL) has become essential for developers and analysts alike.

The SQL Mentor project aims to provide an interactive learning platform that helps users master SQL through quizzes, error analysis, and an expert system that guides them in understanding SQL syntax effectively.

2.2 Literature Review

Educational projects in SQL are based on various methodologies, including:

- Challenge-Based Learning: As seen in platforms like LeetCode SQL and HackerRank SQL, where users learn by solving practical exercises.
- Interactive Learning: Focused on live SQL execution and instant result analysis, similar to W3Schools SQL Editor.
- Error Feedback Mechanisms: These compare user queries with correct answers, using text parsing algorithms to provide feedback on mistakes.

2.3 Theoretical Foundations and Technologies Used

2.3.1 SQL Language and Its Applications

SQL (Structured Query Language) is the primary language used for managing Relational Database Management Systems (RDBMS). Key SQL operations include:

- SELECT: To retrieve data.
- INSERT, UPDATE, DELETE: For data manipulation.
- JOINs: To link multiple tables.
- Indexing & Optimization: To enhance query performance.

2.3.2 SQL Error Analysis

To analyze SQL query errors, the following techniques can be used:

- Syntax Analysis: Utilizing libraries like ANTLR to detect syntax errors in queries.
- Semantic Analysis: Comparing query outputs with expected results to identify common error patterns.
- Automated Error Correction: Implementing techniques like Levenshtein Distance to suggest possible corrections for SQL queries.

Automated Error Correction: Implementing techniques like Levenshtein Distance to suggest possible corrections for SQL queries.

2.3.3 Expert Systems for SQL Query Generation

Expert systems leverage artificial intelligence techniques to understand user requirements and generate SQL queries accordingly. These systems:

- Interact with users by asking structured questions about the data they want to retrieve.
- Generate SQL queries based on user inputs.
- Improve user queries and suggest optimizations.

2.3.4 Interactive User Interface Development

The platform will be developed using modern technologies to provide a seamless interactive experience, including:

- Next.js + TypeScript: For a dynamic front-end experience.
- Drizzle ORM: For efficient database management.
- Python (FastAPI or Flask): For the AI-driven backend that handles SQL query analysis.

2.4 Conclusion

SQL Mentor aims to provide an engaging and practical learning experience that mirrors real-world challenges in SQL query writing. Through quizzes, expert systems, and error analysis with automated corrections, the platform will leverage AI and database technologies to enhance SQL education and make learning more accessible and effective.

This project is a step toward making SQL learning more accessible and effective using artificial intelligence!

Chapter 3: Literature Review

The domain of this project revolves around the development of an interactive SQL learning platform that combines theoretical knowledge with practical application. SQL (Structured Query Language) is a fundamental tool for managing and manipulating relational databases, making it a critical skill for developers, data analysts, and database administrators. The goal of this platform is to provide users with a structured and engaging learning experience, incorporating features such as interactive quizzes, AI-based error analysis, and real-world scenarios to enhance understanding and retention.

This chapter explores existing works and applications in the field of SQL learning and database management. By analyzing similar platforms, we can identify best practices, innovative features, and gaps in the current market, which will inform the design and development of our proposed system.

3.1 Related Works

The system provides a facility to accept the requests from traders after they register. The admin will see the requests from the trader and check if it's a real information like the credit card or passport id, after that the admin will approve or reject the uncompleted requests, the trader can create stores after admin approval and add their categories and products.

3.2 Similar Applications

In the development of the SQL project for managing trader registrations and store creation, several similar applications have been widely used for learning and implementing SQL-based solutions. These applications offer valuable insights and features that contribute to understanding and improving various SQL functionalities.

3.3 Similar Applications:

3.3.1 HackerRank

HackerRank provides a wide range of SQL challenges and exercises for database. It allows users to practice and enhance their SQL skills through real-world scenarios, offering an interactive platform for learning and application of SQL queries (HackerRank, n.d.).

3.3.2 coddy.tech

coddy.tech is an online platform designed for SQL training and learning. It offers hands-on practice with SQL queries, covering diverse topics like joins, subqueries, data manipulation, and database design. This tool is useful for anyone seeking to deepen their understanding of SQL (Coddy.Tech, n.d.).

3.3.3 SQLTUTORIAL

SQLTUTORIAL offers a structured approach to learning SQL with detailed explanations and practical exercises. It serves as a comprehensive guide for beginners and intermediate learners, focusing on the fundamentals of SQL and gradually progressing to more advanced topics (SQLTutorial, n.d.).

3.3.4 Sql Zoo

Sql Zoo is an educational platform that helps users practice SQL queries through interactive exercises. It provides a variety of databases for users to interact with, enabling the practical application of SQL syntax in a controlled, learning environment (SQLZoo, n.d.).

3.3.5 W3Schools

W3Schools is a widely used online platform offering educational resources on web development and database technologies. It provides comprehensive SQL tutorials, interactive exercises, and step-by-step instructions to help users learn and apply SQL concepts effectively (W3Schools, n.d.).

3.4 Comparison

Feature	HackerRank	coddy.tech	SQLTOTORIAL	Sql zoo	W3Schools	Sql mentor
Chatbot						✓
Quiz generation					✓	✓
Score calculation						✓
Levels system	✓		✓	✓	✓	✓
Interactive compile	✓		✓		✓	✓
Self DB editor						✓
Practical mood	✓	✓	✓	✓	✓	✓
Friendly UI	✓				✓	✓
Real examples	✓	✓	✓		✓	✓
Real output	✓		✓	✓	✓	✓

3.5 State of the Art Studies

Reference	Year	Data	Preprocessing	Model	Result
(Khadija & Mustapha, 2024)	2024	Spider		RAT-SQL	62.7
				Bertrand-DR	57.9
				Photon	63.2
				Athena++	87.82
		M-SQL		TableQA	92.1
		IE-SQL		WikiSQL	94.2
(Gonzales, Ong, Cheng, Ong, & Azcarraga, 2023)	2023	Conversations from 227 students (aged 6-7), conducted during a pediatric health survey.	Entity Extraction	DialogFlowCX	only a one-second delay when the dataset size increased by a factor of 105
			Data Aggregation		
(Song, Wong, Zhao, & Jiang, 2024)	2024	WikiSQL	Extracted 96 log-scaled Mel-band energies from speech	CNN (Speech Encoder)	54.15
				GNN (Schema Encoder)	improves query match accuracy by 0.69% when handling multi-table queries compared to vanilla RNN
		Spider	Converted database schema into graphs with nodes and edges	Transformer (Relation-aware Encoder)	18.05
				LSTM (SQL-aware Decoder)	improves performance by 28.35% compared to vanilla RNN decoder for SemQL grammar decoding
(Dinu, Mihăescu, & Rebedea, 2023)	2023	Spider	Schema encoding and linking	RAT-SQL	80
				RoBERTa	Improved performance
(Wong, et al., 2024)	2024	spider		T5	73
				Graphix-T5	74
				RASAT	72

Chapter 4: System Analysis

4.1 Requirements Specification Overview

The requirements specification outlines the necessary features, constraints, and specifications for developing a software system. It serves as a foundation for the design, development, and testing phases of the project.

4.2. Functional Requirements

1. User Authentication

    - The system should allow users to register and log in using email and password.

2. CRUD Operations

    - The system should support creating, reading, updating, and deleting user data.

3. Data Management

    - Users should be able to manage databases, including creating new databases, modifying them, and deleting them.

4. Query Execution

    - Allow users to execute SQL queries with features for displaying results and handling errors.

5. Quiz Generation

    - Generate SQL quizzes with different levels of difficulty.
    - Provide feedback on quiz performance with explanations for correct/incorrect answers.

6. Chatbot EXP/ Code Generation

    - Provide a chatbot system to assist users with SQL-related queries and generate appropriate queries based on user questions.
    - Automatically generate SQL code based on textual input from users.

7. Post Management

    - Users should be able to add, edit, view, and delete posts.

8. Exercise Management

    - Support creation, modification, and deletion of exercises for users to practice SQL queries.

4.3. Non-Functional Requirements

1. Performance

- The system should support at least 100 concurrent users without significant performance degradation.
- SQL queries should execute within 2 seconds for basic operations.

2. Scalability

- The system should be scalable to support an increased number of users and data without performance loss.

3. Usability

- The system should have an intuitive interface that suits users of all skill levels.

4. Security

- The system protects against SQL injection attacks.
- User passwords are encrypted in the database using secure hashing algorithms.
- The website uses separate databases: one for the main site, one for exercises, and another for user data to prevent data interferences.

5. Reliability

- The system should have a 99.9% uptime with minimal downtime during maintenance.

6. Maintainability

- The code should be modular and well-documented for ease of maintenance and updates.

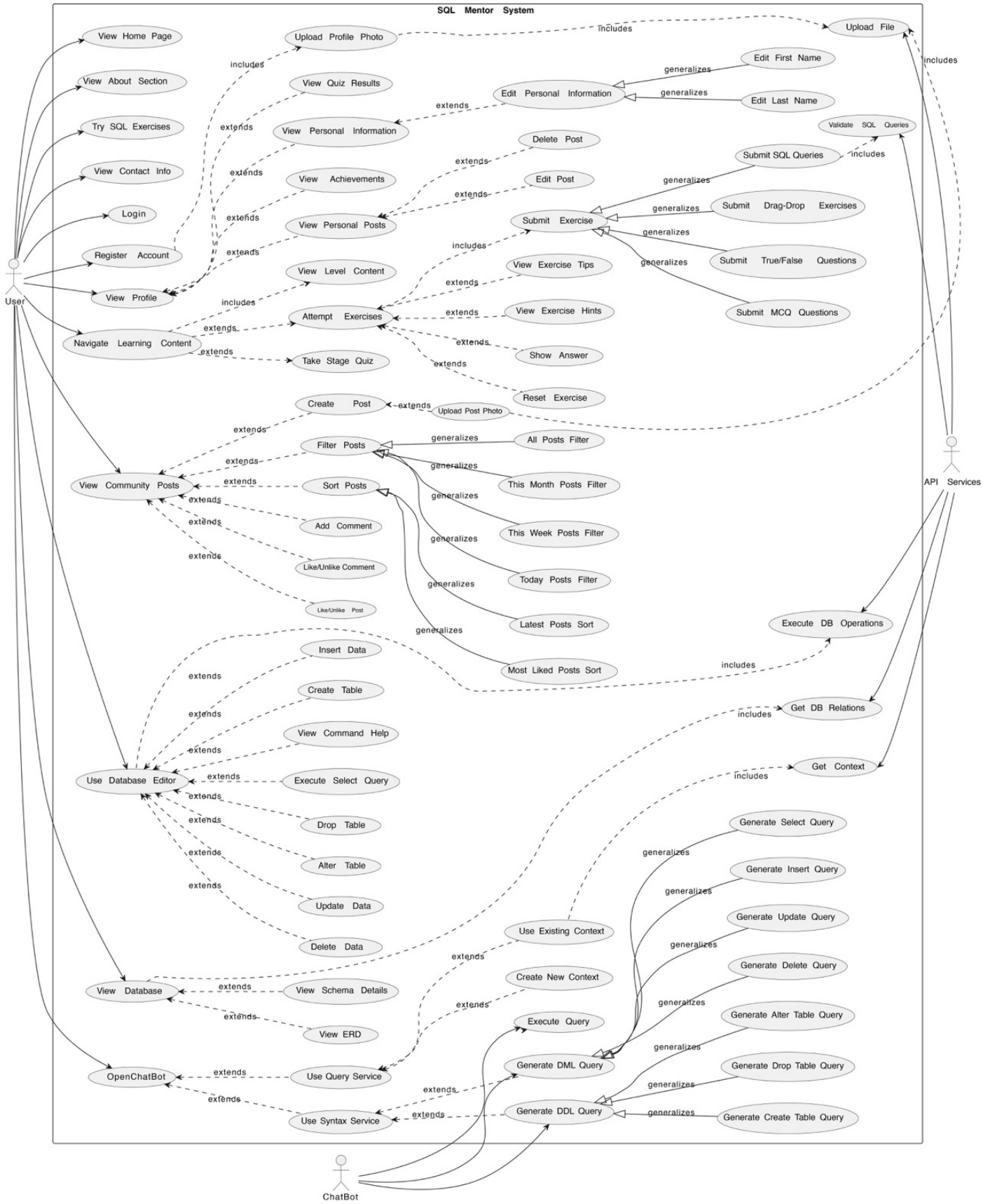
7. Compatibility

The system should be compatible with popular browsers and operating systems

4.4 Use cases

Use case is a list of actions or event steps typically defining the interactions between a role and a system to achieve a goal. The actor can be a human or other external system. In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in the Systems Modeling Language or as contractual statements.

After specifying the functional requirements of the application, we will represent the use case diagram that connects those functionalities together in order to show how the flow of information within the application is done.



use case diagram (Figure1)

4.4.2 Use case specification

- Login

Element	Description
Use Case Name	Login
Actors	- **Primary Actor:** User - **Supporting Actor:** Authentication Service
Pre-conditions	1. The user must have an existing account in the system. 2. The user must know their login credentials (username and password).
Post conditions	1. **Success:** The user is authenticated and granted access to the system with a valid session or token. 2. **Failure:** The system informs the user of incorrect credentials or errors, and no access is granted.
Basic Flow	1. The user navigates to the login page. 2. The system displays fields for username and password. 3. The user enters their credentials and clicks the **Login** button. 4. The system sends the credentials to the Authentication Service. 5. The Authentication Service validates the credentials by checking them against the database. 6. If the credentials are valid: - The system generates a session or token. - The user is granted access and redirected to the main page or dashboard. 7. The system displays a success message to the user.
Alternative Flows	**A1: Invalid Credentials** 1. The Authentication Service detects invalid credentials. 2. The system displays an error message: "Invalid username or password." 3. The user remains on the login page and can retry. **A2: Account Locked** 1. The user exceeds the maximum allowed login attempts. 2. The system locks the account temporarily and displays a message: "Account locked. Please try again after [time period] or reset your password.

- Registration

Element	Description
Use Case Name	Registration
Actors	- **Primary Actor:** User - **Supporting Actor:** Authentication Service
Pre-conditions	1. The user must provide valid registration details (e.g., username, password, email). 2. The user must not already have an account.
Post conditions	1. **Success:** The user's account is created and stored in the database. 2. **Failure:** The system informs the user of any errors (e.g., duplicate username).
Basic Flow	1. The user navigates to the registration page. 2. The system displays fields for the required details (username, password, email, etc.). 3. The user enters the details and submits the form. 4. The system checks if the username or email already exists in the database. 5. If the details are unique, the system saves the user information to the database. 6. The system displays a success message to the user.
Alternative Flows	**A1: Duplicate Username/Email** 1. The system detects that the username or email already exists. 2. The system displays an error message: "Username or email already in use." 3. The user is prompted to provide different details.
Exceptions	- **E1:** Network issues prevent the registration request from reaching the Authentication Service. - **Action:** Inform the user and suggest retrying later. - **E2:** Database query fails during account creation. - **Action:** Log the error and display a generic error message: "An unexpected error occurred. Please try again."

- Code Generation

Element	Description
Use Case Name	Code Generation
Actors	- **Primary Actor:** User - **Supporting Actor:** Code Generation Service, Database, AI Model
Pre-conditions	1. The user must provide a valid natural language query. 2. The database schema must be available to the system.
Post conditions	1. **Success:** The system generates and returns an accurate SQL query based on the user's input. 2. **Failure:** The system informs the user if the query cannot be generated.
Basic Flow	1. The user provides a natural language query describing the desired data retrieval or operation. 2. The system sends the query to the Code Generation Service. 3. The service retrieves the schema information from the database. 4. The system processes the user's query and schema using the AI model. 5. The AI model generates an SQL query. 6. The generated SQL query is returned to the user.
Alternative Flows	**A1: Missing Schema Information** 1. The system detects that the schema information is unavailable. 2. The system informs the user and suggests verifying the database connection. **A2: Ambiguous Query** 1. The system identifies ambiguity in the user's input. 2. The system requests clarification from the user and adjusts the process accordingly.
Exceptions	- **E1:** The AI model fails to process the user's query due to complexity or unexpected input. - **Action:** The system logs the error and informs the user with a generic message: "Unable to generate query, please simplify your input."

- Expert System Chatbot

Element	Description
Use Case Name	Expert System Chatbot
Actors	- **Primary Actor:** User - **Supporting Actor:** Chatbot Service, Knowledge Base, AI Model
Pre-conditions	1. The user must provide a valid SQL-related query (e.g., syntax question). 2. The system must have access to the Knowledge Base and AI Model.
Post conditions	1. **Success:** The chatbot provides a valid syntax explanation or suggested query. 2. **Failure:** The chatbot informs the user if it cannot process the query.
Basic Flow	1. The user inputs an SQL-related query into the chatbot interface. 2. The system sends the query to the Chatbot Service. 3. The Chatbot Service determines if the query is syntax-related or ambiguous. 4. If syntax-related: - The system retrieves the corresponding explanation from the Knowledge Base. 5. If further processing is needed: - The Chatbot Service forwards the query to the AI Model for advanced processing. 6. The system returns the answer to the user.
Alternative Flows	A1: Non-SQL Query 1. The chatbot identifies that the query is unrelated to SQL. 2. The chatbot informs the user that the query is unsupported.
Exceptions	- **E1:** The Knowledge Base does not contain relevant information. - **Action:** The system logs the issue and informs the user: "No relevant syntax found." - **E2:** The AI Model fails to process the query. - **Action:** The system logs the error and informs the user with a generic message: "Unable to process your query. Please try again."

- Score Calculation

Element	Description
Use Case Name	Score Calculation
Actors	- **Primary Actor:** User - **Supporting Actor:** Score Calculation Service, AI Model
Pre-conditions	1. The user must have completed a quiz or exercise. 2. The system must have access to the correct answers and user's submitted answers.
Post conditions	1. **Success:** The system calculates and displays the percentage score to the user. 2. **Failure:** The system informs the user if score calculation is not possible.
Basic Flow	1. The user submits their answers after completing a quiz or exercise. 2. The system sends the user's answers and the correct answers to the Score Calculation Service. 3. The service converts both sets of answers into vectors using the AI Model. 4. The system calculates the cosine similarity between the two vectors to determine accuracy. 5. The calculated similarity is converted into a percentage score. 6. The system displays the user's score (e.g., "Your score: 85%").
Alternative Flows	**A1: Partial Answers Submitted** 1. The user submits incomplete answers. 2. The system calculates the score for the submitted questions only and displays a partial score.
Exceptions	- **E1:** The AI Model fails to process the answers. - **Action:** Log the error and display a message: "Unable to calculate your score. Please try again."

- Quiz Generation

Element	Description
Use Case Name	Quiz Generation
Actors	- **Primary Actor:** User - **Supporting Actor:** Quiz Generation Service, Behavior Analysis Service, Database
Pre-conditions	1. The user must be logged in. 2. The system must have sufficient questions in the database for the user's level.
Post conditions	1. **Success:** The system generates a personalized quiz tailored to the user's level and behavior.
Basic Flow	1. The user requests a quiz. 2. The system sends the request to the Quiz Generation Service. 3. The service retrieves the user's level and performance history from the Behavior Analysis Service. 4. Based on the insights, the system fetches relevant questions from the database. 5. The service generates a quiz with a tailored number of questions and difficulty levels. 6. The system presents the quiz to the user.
Alternative Flows	**A1: Insufficient Questions** 1. The system detects a lack of questions for the user's level. 2. The system displays a message: "Not enough questions available for your level. Please try again later." **A2: User Behavior Unavailable** 1. The system cannot retrieve the user's performance data. 2. The system generates a generic quiz based on the user's level only.
Exceptions	- **E1:** The database query fails while retrieving questions. - **Action:** Log the error and display a generic error message: "An unexpected error occurred. Please try again."

- Solve Exercise

Element	Description
Use Case Name	Solve Exercise
Actors	**Primary Actor**: User **Supporting Actors:** Backend, Exercise Database, Main Database
Pre conditions	1. User must be logged into the system. 2. An exercise must be available for the user to solve.
Post conditions	1. **Success:** User receives the exercise result and accuracy percentage. 2. **Failure:** Error messages are displayed if issues occur during query validation or execution.
Basic Flow	1. User opens an exercise. 2. System starts tracking time and logs it in the main database. 3. User submits an answer (SQL query or statement). 4. Backend validates the query syntax. 5. If valid, the query is executed in the Exercise Database. 6. Backend calculates query accuracy and stores results in the main database. 7. System displays the result and accuracy to the user.
Alternative Flows	**A1: Invalid Syntax** 1. Query syntax is invalid. 2. System returns a syntax error with 0% accuracy. **A2: Show Answer** 1. Backend logs the event and updates the main database.
Exceptions	- **E1:** Database Error - **Action:** Log the error and display a generic error message: "An unexpected error occurred. Please try again."

- Add Post

Element	Description
Use Case Name	Add Post
Actors	**Primary Actor:** User **Supporting Actors:** Backend, Post Database
Pre conditions	1. User must be logged into the system.
Post conditions	1. **Success:** The post is successfully added to the Post Database. 2. **Failure:** An error message is displayed if the post cannot be added.
Basic Flow	1. User navigates to the "Add Post" section. 2. User fills in the post details (e.g., title, content, category). 3. User submits the post. 4. Backend validates the input. 5. If valid, the post is stored in the Post Database. 6. System confirms the successful addition of the post and displays it in the user's dashboard.
Alternative Flows	**A1: Missing Information** 1. User submits incomplete details. 2. System prompts the user to complete the required fields. **A2: Cancel Action** 1. User cancels the operation. 2. System discards any entered data and returns to the previous page.
Exceptions	- **E1: Database Error** Action: Log the error and display a generic error message: "An unexpected error occurred. Please try again."

- Edit Post

Element	Description
Use Case Name	Edit Post
Actors	**Primary Actor:** User **Supporting Actors:** Backend, Post Database
Pre conditions	1. User must be logged into the system. 2. The user must have permission to edit the post.
Post conditions	1. **Success:** The post is successfully updated in the Post Database. 2. **Failure:** An error message is displayed if the post cannot be updated.
Basic Flow	1. User navigates to the "Edit Post" section for a specific post. 2. System retrieves the existing post details. 3. User modifies the post details (e.g., title, content, category). 4. User submits the changes. 5. Backend validates the input. 6. If valid, the updated post is stored in the Post Database. 7. System confirms the successful update of the post and displays the updated version.
Alternative Flows	**A1: Cancel Action** 1. User cancels the operation. 2. System discards any changes and returns to the previous page.
Exceptions	**- E1: Database Error** Action: Log the error and display a generic error message: "An unexpected error occurred. Please try again."

- Delete Post

Element	Description
Use Case Name	Delete Post
Actors	**Primary Actor:** User **Supporting Actors:** Backend, Post Database
Pre conditions	1. User must be logged into the system. 2. The user must have permission to delete the post.
Post conditions	1. **Success:** The post is removed from the Post Database. 2. **Failure:** An error message is displayed if the post cannot be deleted.
Basic Flow	1. User navigates to the "Delete Post" option for a specific post. 2. System prompts the user for confirmation. 3. User confirms the deletion. 4. Backend verifies the user's permission. 5. The post is removed from the Post Database. 6. System confirms the successful deletion of the post.
Alternative Flows	**A1: Cancel Deletion** 1. User cancels the confirmation. 2. System retains the post and exits the deletion process.
Exceptions	- **E1: Database Error** Action: Log the error and display a generic error message: "An unexpected error occurred. Please try again."

- Like Post

Element	Description
Use Case Name	Like Post
Actors	**Primary Actor:** User **Supporting Actors:** Backend, Post Database
Pre conditions	1. User must be logged into the system. 2. The post must exist in the database.
Post conditions	1. **Success:** The like is added to the Post Database and displayed to the user. 2. **Failure:** An error message is displayed if the like cannot be recorded.
Basic Flow	1. User clicks the "Like" button on a specific post. 2. Backend verifies the user's identity and ensures the post exists. 3. Backend records the like in the Post Database. 4. System updates the like count and reflects the change to the user.
Alternative Flows	**A1: Unlike Post** 1. User clicks the "Unlike" button. 2. Backend removes the like from the Post Database. 3. System updates the like count and reflects the change.
Exceptions	- **E1: Database Error** Action: Log the error and display a generic error message: "An unexpected error occurred. Please try again."

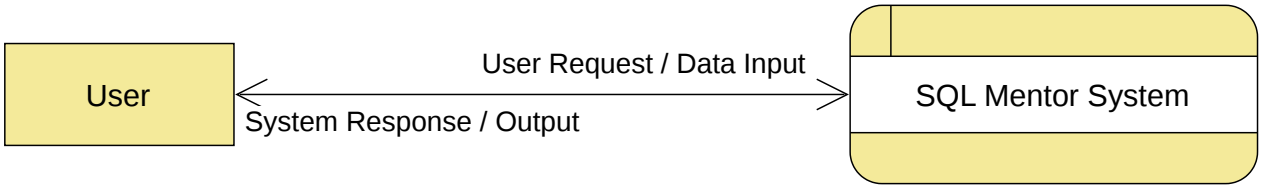
- Add Comment

Element	Description
Use Case Name	Add Comment
Actors	**Primary Actor:** User **Supporting Actors:** Backend, Comment Database
Pre conditions	1. User must be logged into the system. 2. The post must exist in the database.
Post conditions	1. **Success:** The comment is successfully added to the Comment Database. 2. **Failure:** An error message is displayed if the comment cannot be added.
Basic Flow	1. User navigates to the comments section of a specific post. 2. User enters the comment text. 3. User submits the comment. 4. Backend validates the input. 5. If valid, the comment is stored in the Comment Database. 6. System confirms the successful addition of the comment and displays it under the post.
Alternative Flows	**A1: Missing Content** 1. User submits an empty comment. 2. System prompts the user to enter content. **A2: Cancel Action** 1. User cancels the operation. 2. System discards any entered data and exits the comment section.
Exceptions	**- E1: Database Error** Action: Log the error and display a generic error message: "An unexpected error occurred. Please try again."

- Like Comment

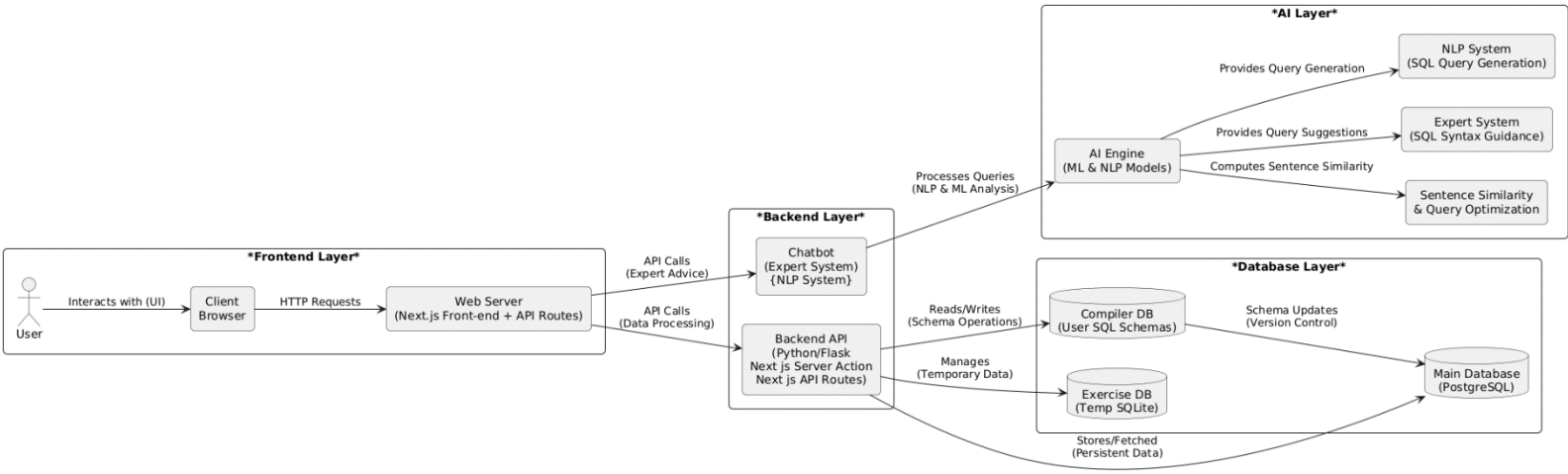
Element	Description
Use Case Name	Like Comment
Actors	**Primary Actor:** User **Supporting Actors:** Backend, Comment Database
Pre conditions	1. User must be logged into the system. 2. The post must exist in the database.
Post conditions	1. **Success:** The like is added to the Comment Database and displayed to the user. 2. **Failure:** An error message is displayed if the like cannot be recorded.
Basic Flow	1. User clicks the "Like" button on a specific comment. 2. Backend verifies the user's identity and ensures the comment exists. 3. Backend records the like in the Comment Database. 4. System updates the like count and reflects the change to the user.
Alternative Flows	**A1: Unlike Comment** 1. User clicks the "Unlike" button. 2. Backend removes the like from the Comment Database. 3. System updates the like count and reflects the change.
Exceptions	- **E1: Database Error** Action: Log the error and display a generic error message: "An unexpected error occurred. Please try again."
Priority	High

4.5 Context Diagram



Context Diagram (figure 2)

4.6 block diagram

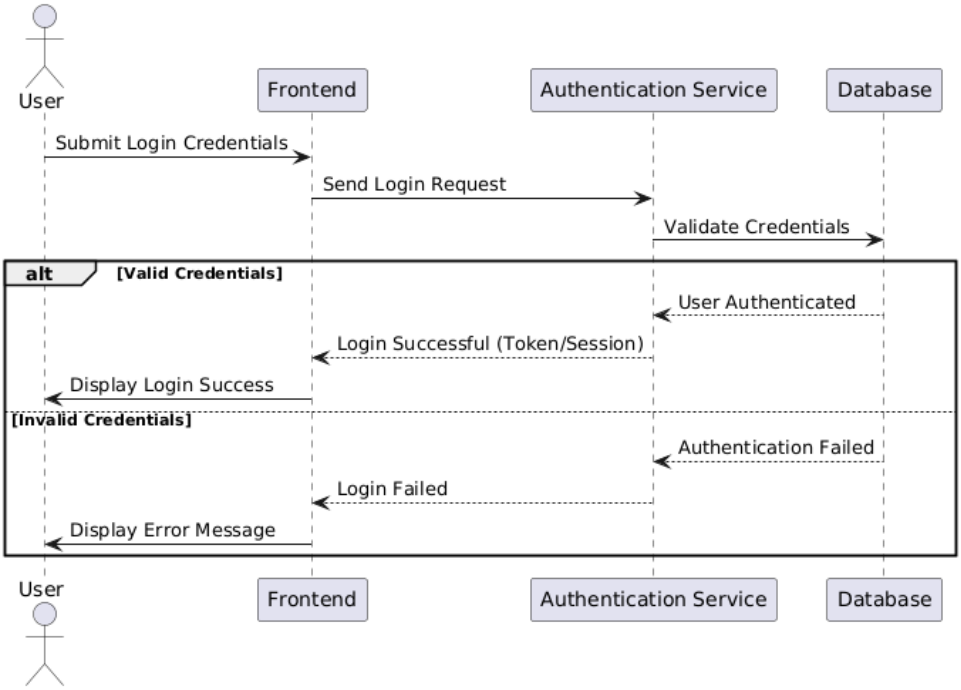


block Diagram (figure 3)

Chapter 5: System Design

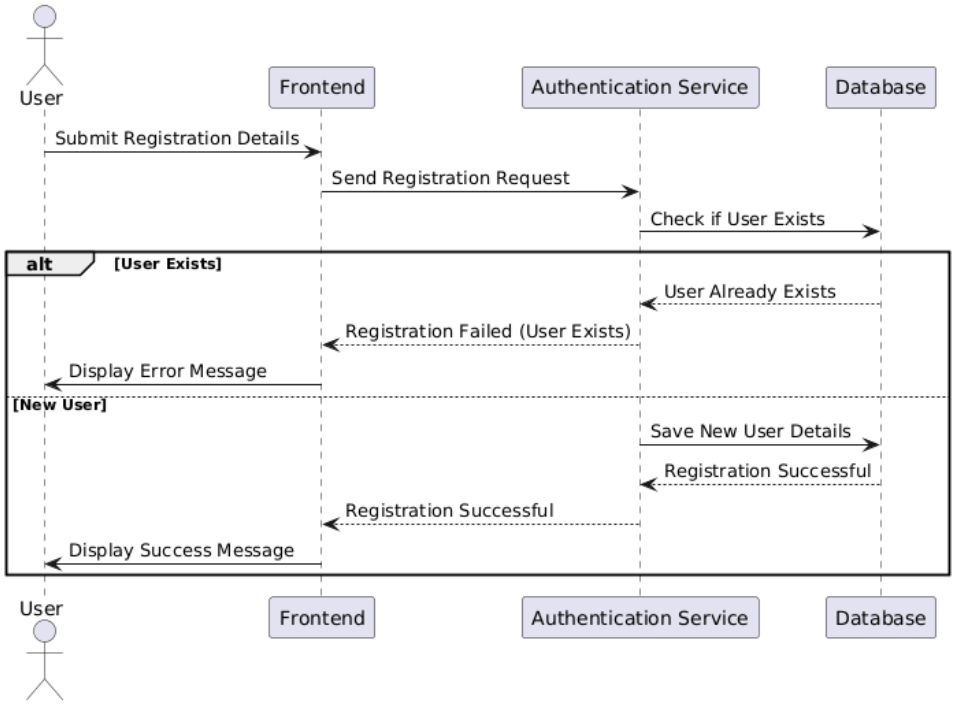
5.1 sequence diagram

- Login diagram:



Login (figure 1)

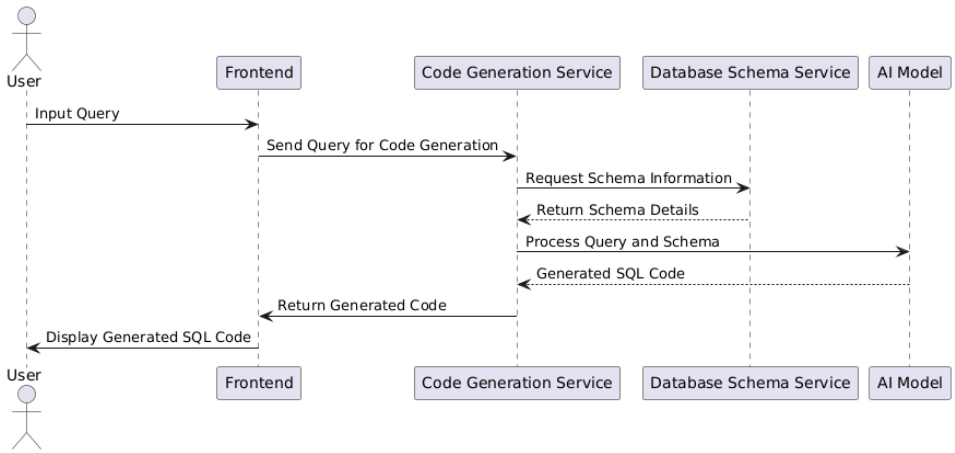
- Sign up diagram:



Signup

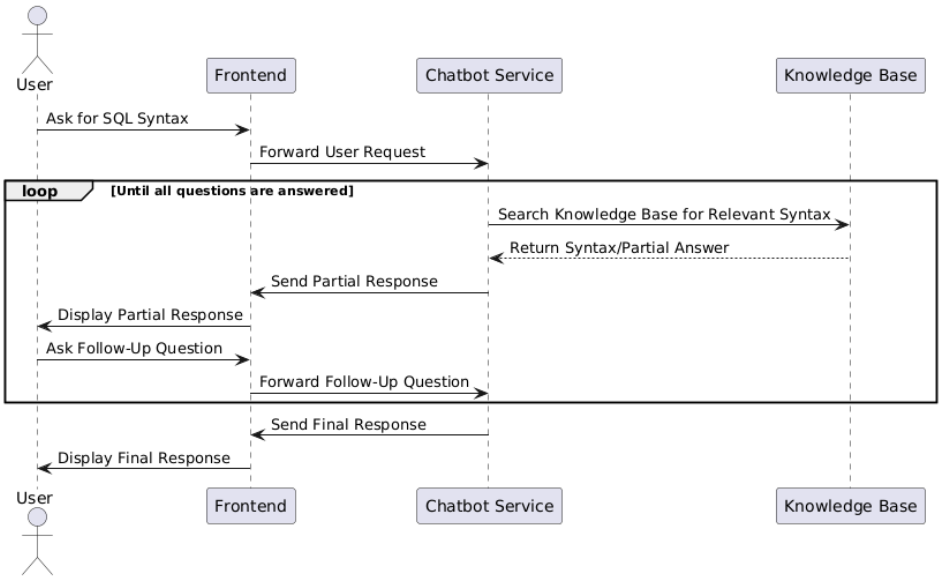
(figure 2)

- Code Generation diagram:



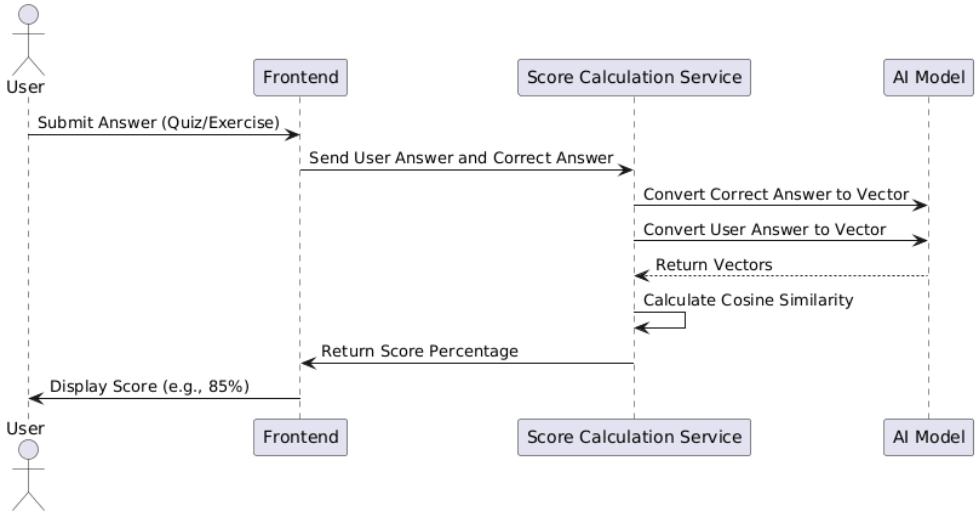
Code generation (figure 3)

- Expert System Chatbot



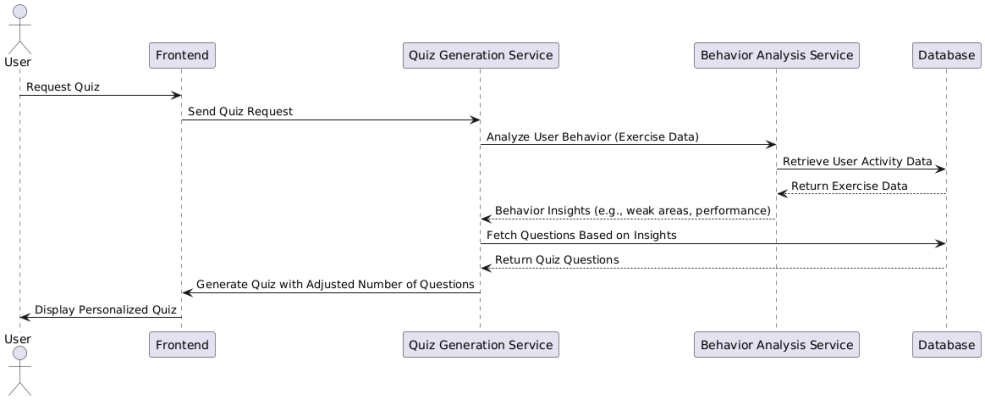
Expert system Chabot (figure 4)

- Score Calculation



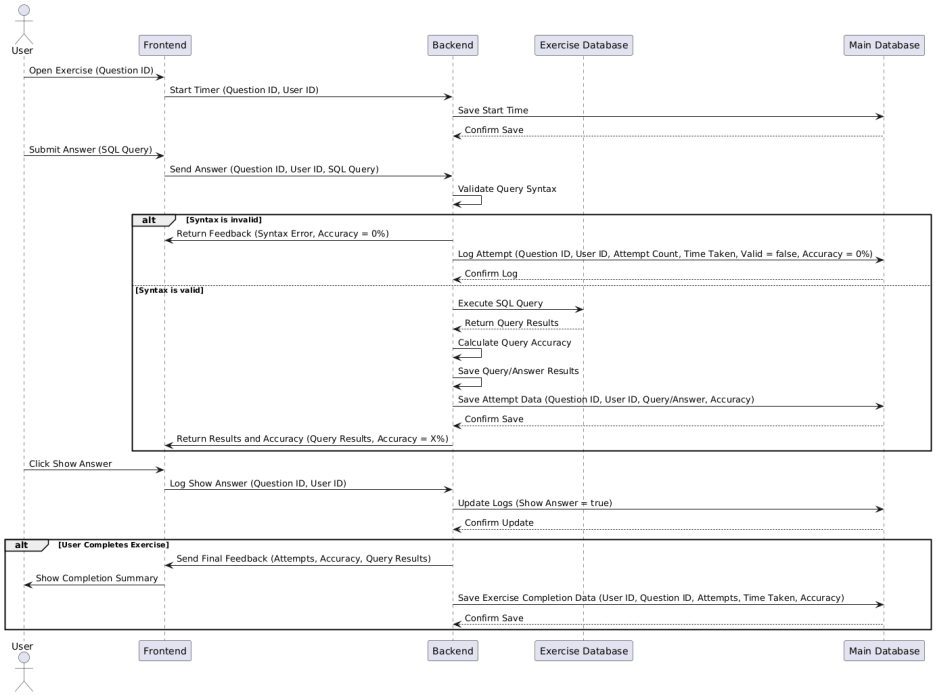
Score calculation (figure 5)

- Quiz Generation



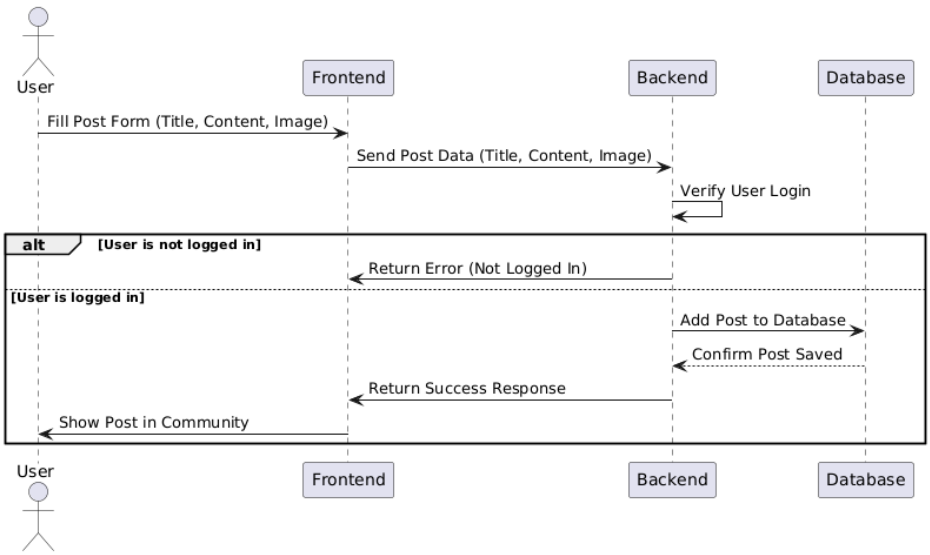
Quiz Generation (figure 6)

- Exercise



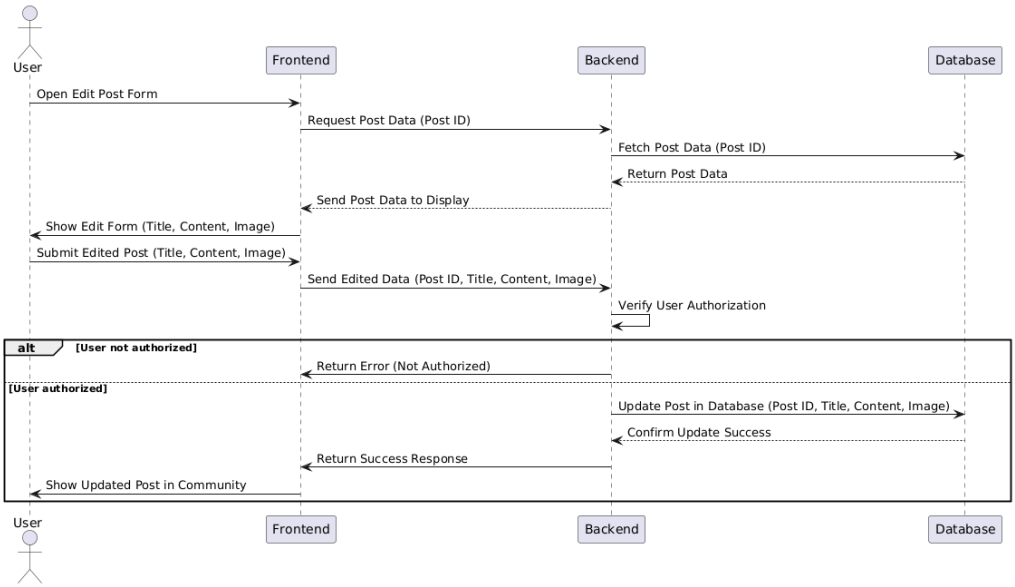
Exercise (figure 7)

- Add Post



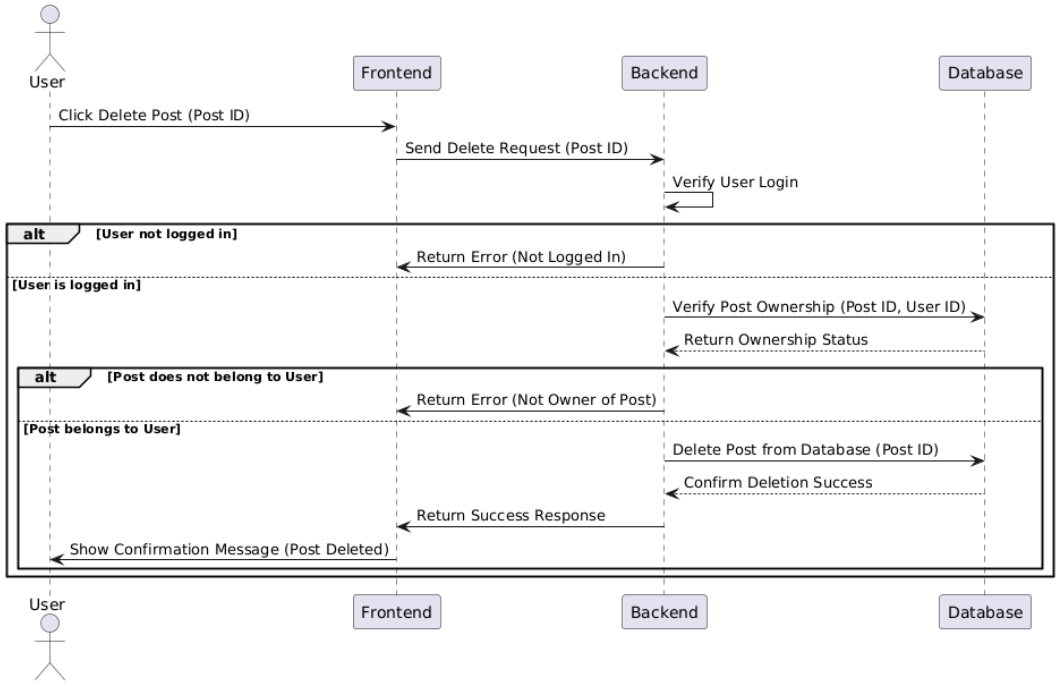
Add post (figure 8)

- Edit Post



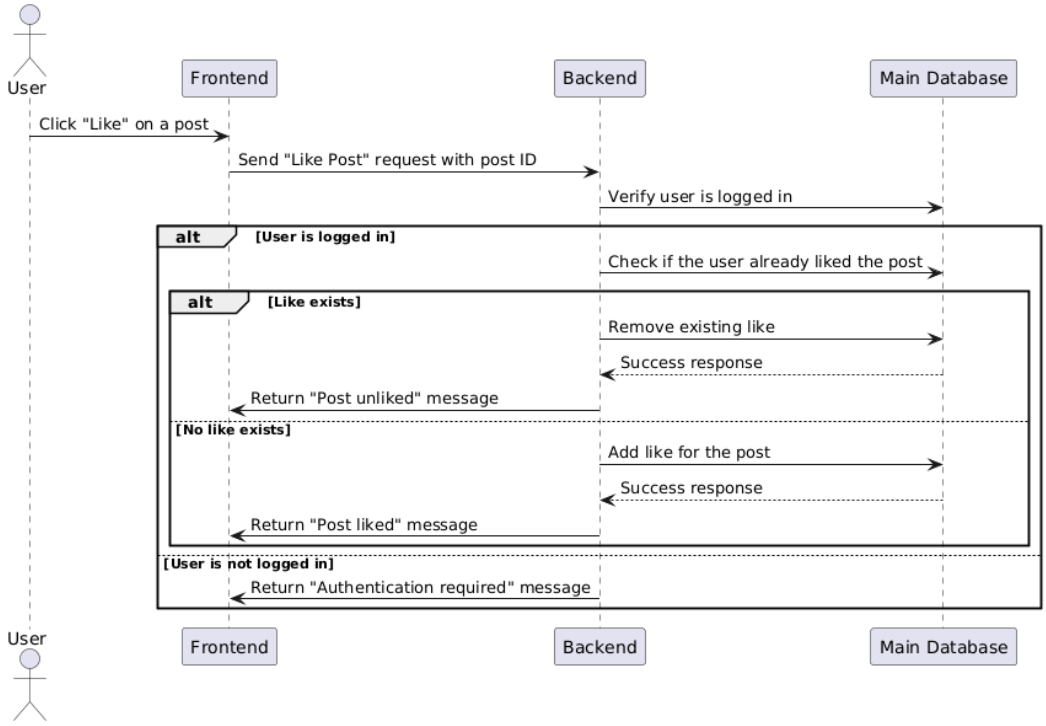
Edit post (figure 9)

- Delete Post



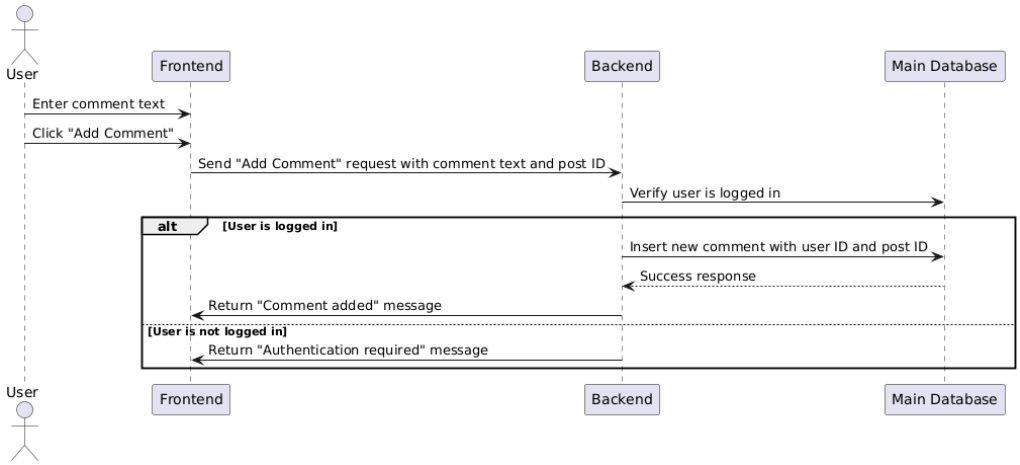
Delete post (figure 10)

- Like Post



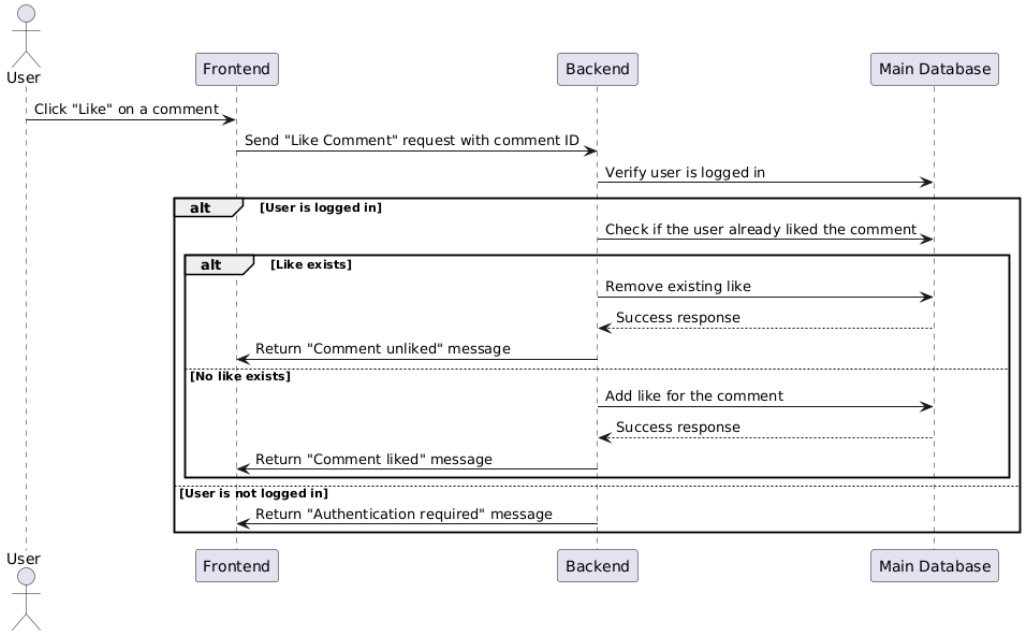
Like post (figure 11)

- Add Comment



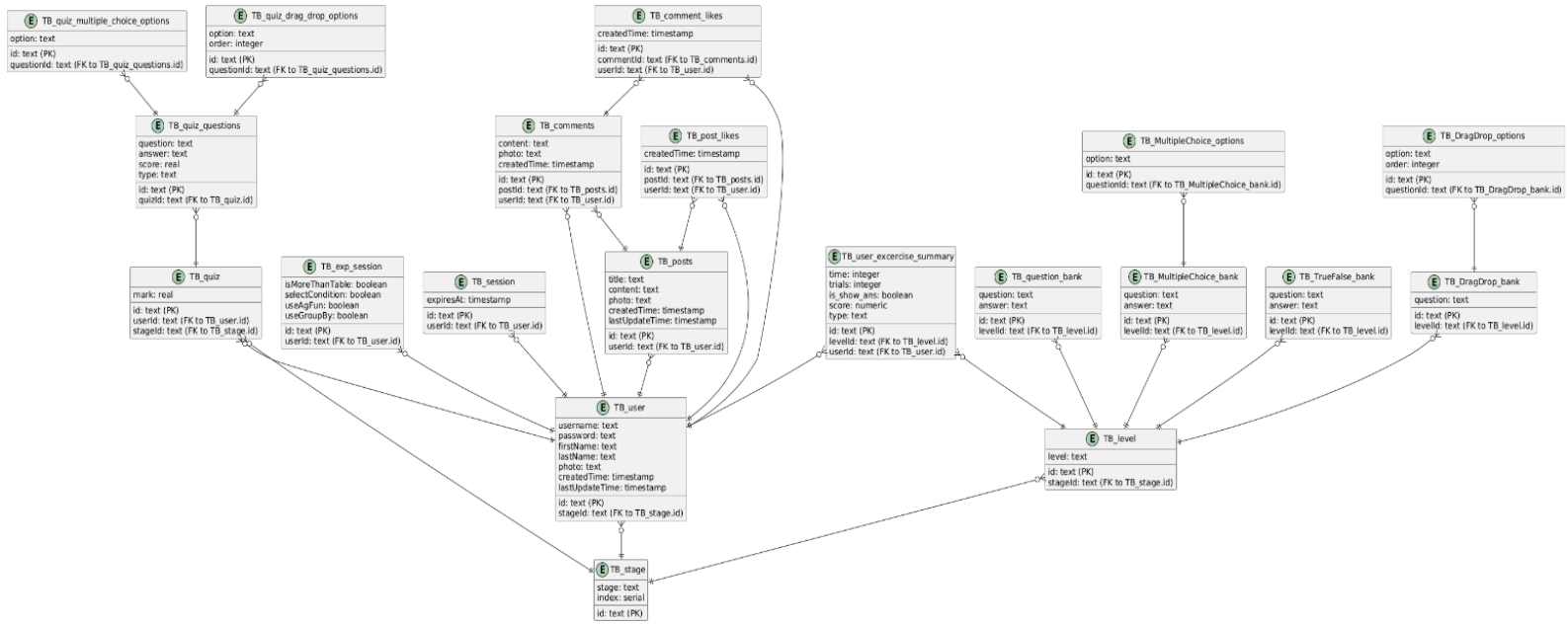
Add comment (figure 12)

- Like Comment



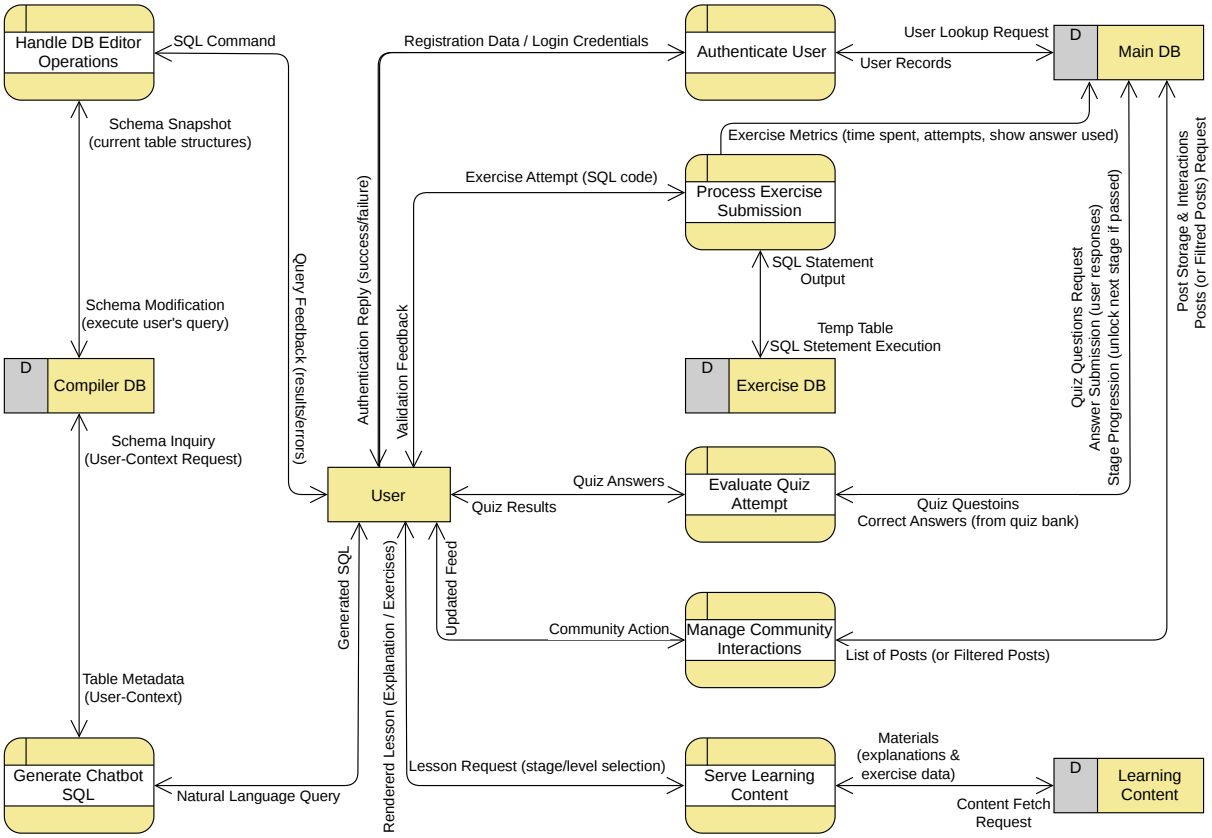
Like comment (figure 13)

5.2 Entity Relationship Diagram



ERD (figure 14)

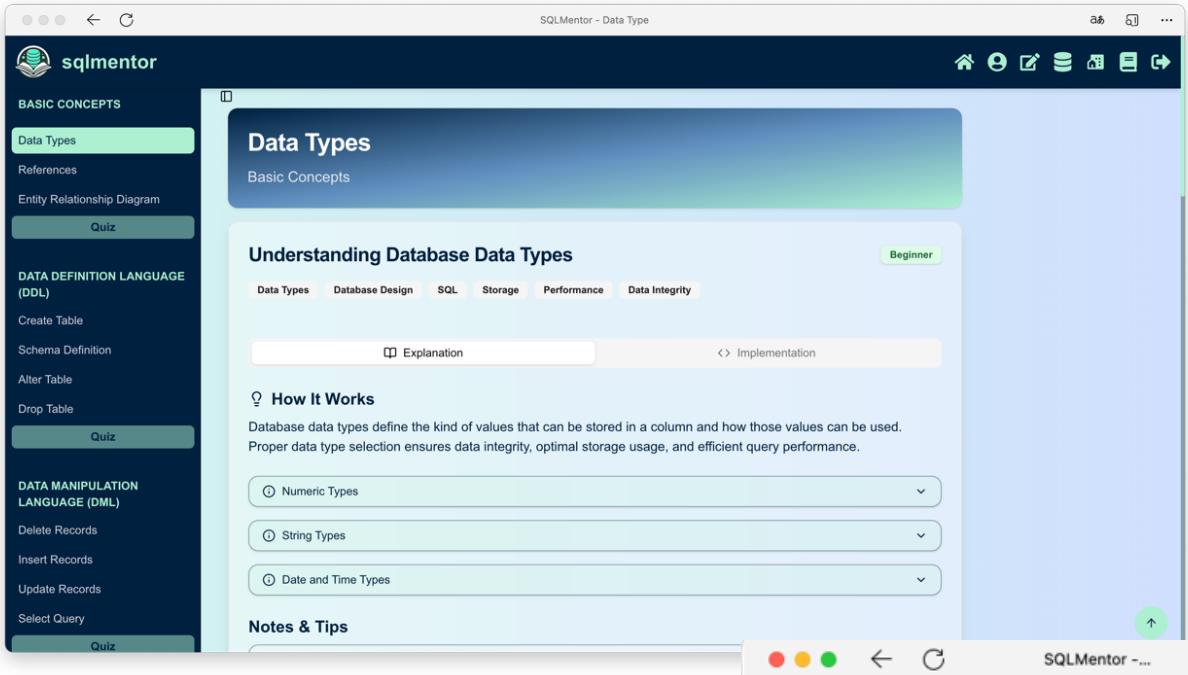
5.3 Data Flow Diagram (Level 1)

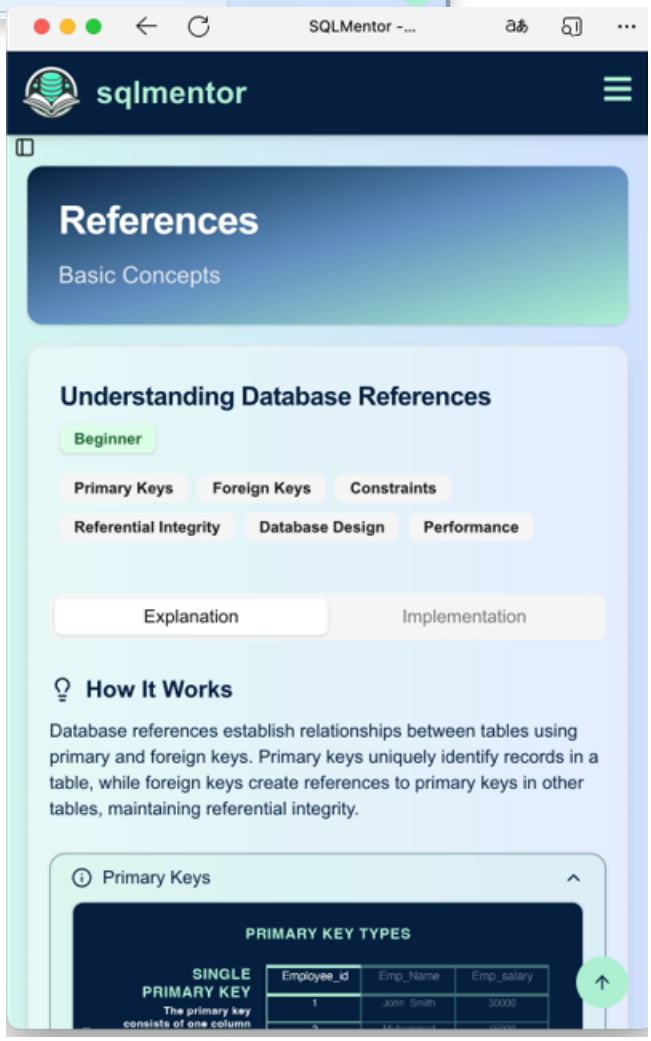


Level 1 DFD (figure 15)

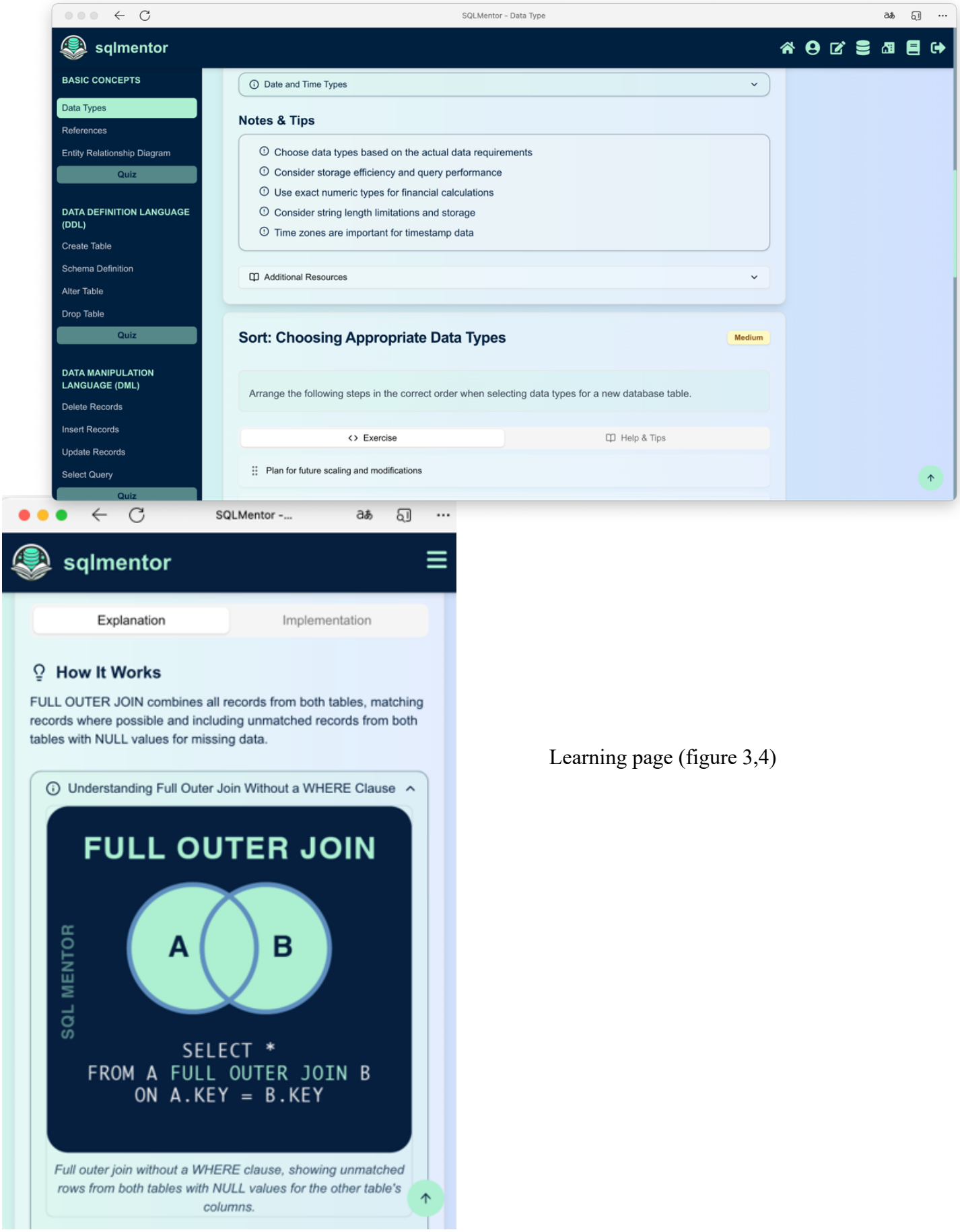
5.4 Web Pages Design

Learning Page

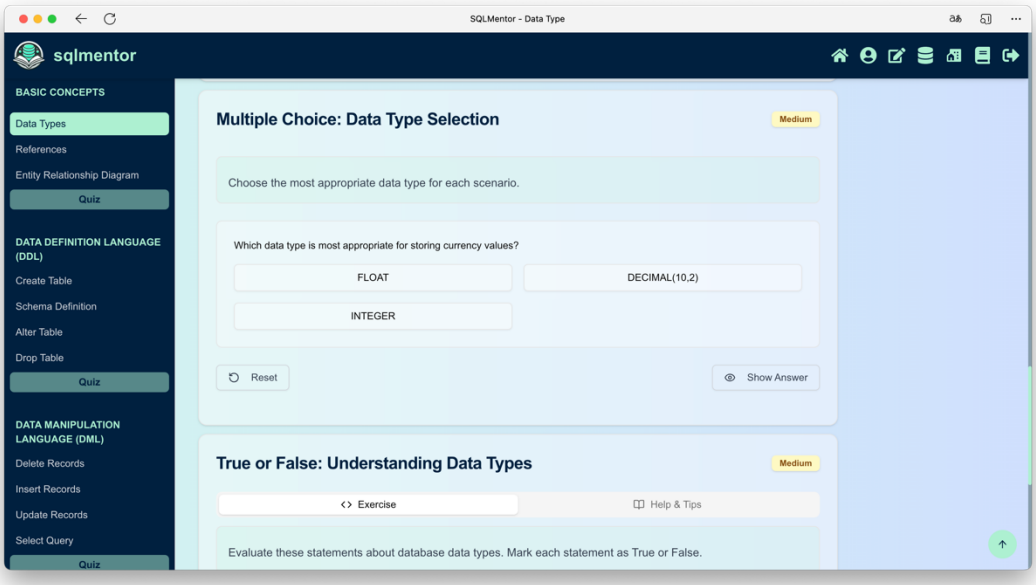


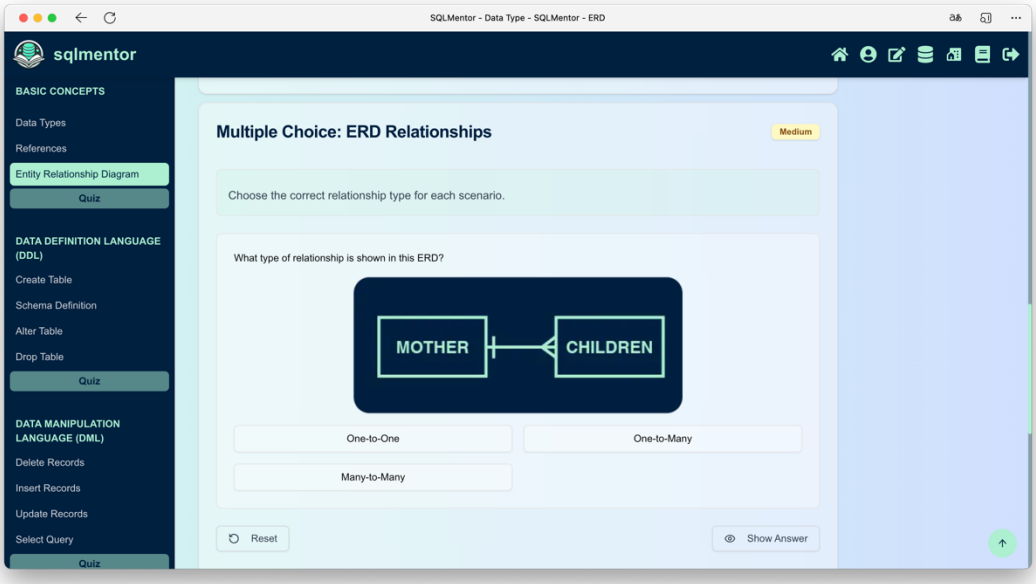


Learning page (figure 1,2)

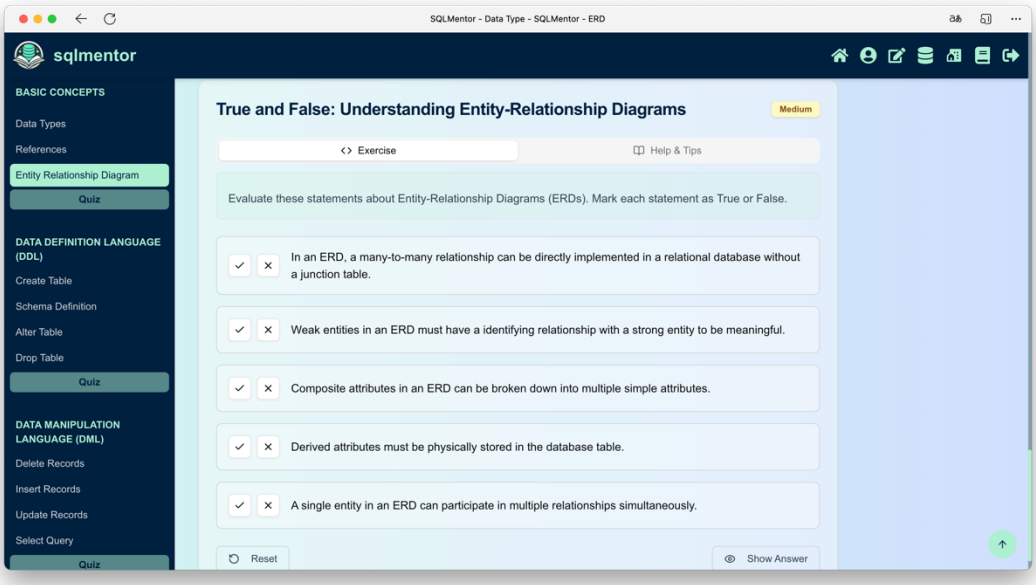


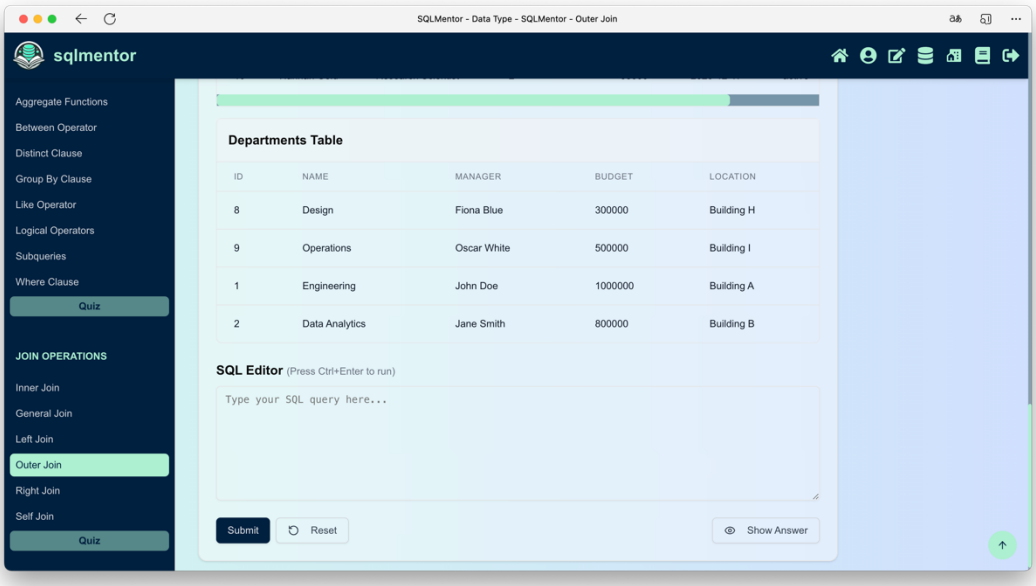
Learning page (figure 3,4)





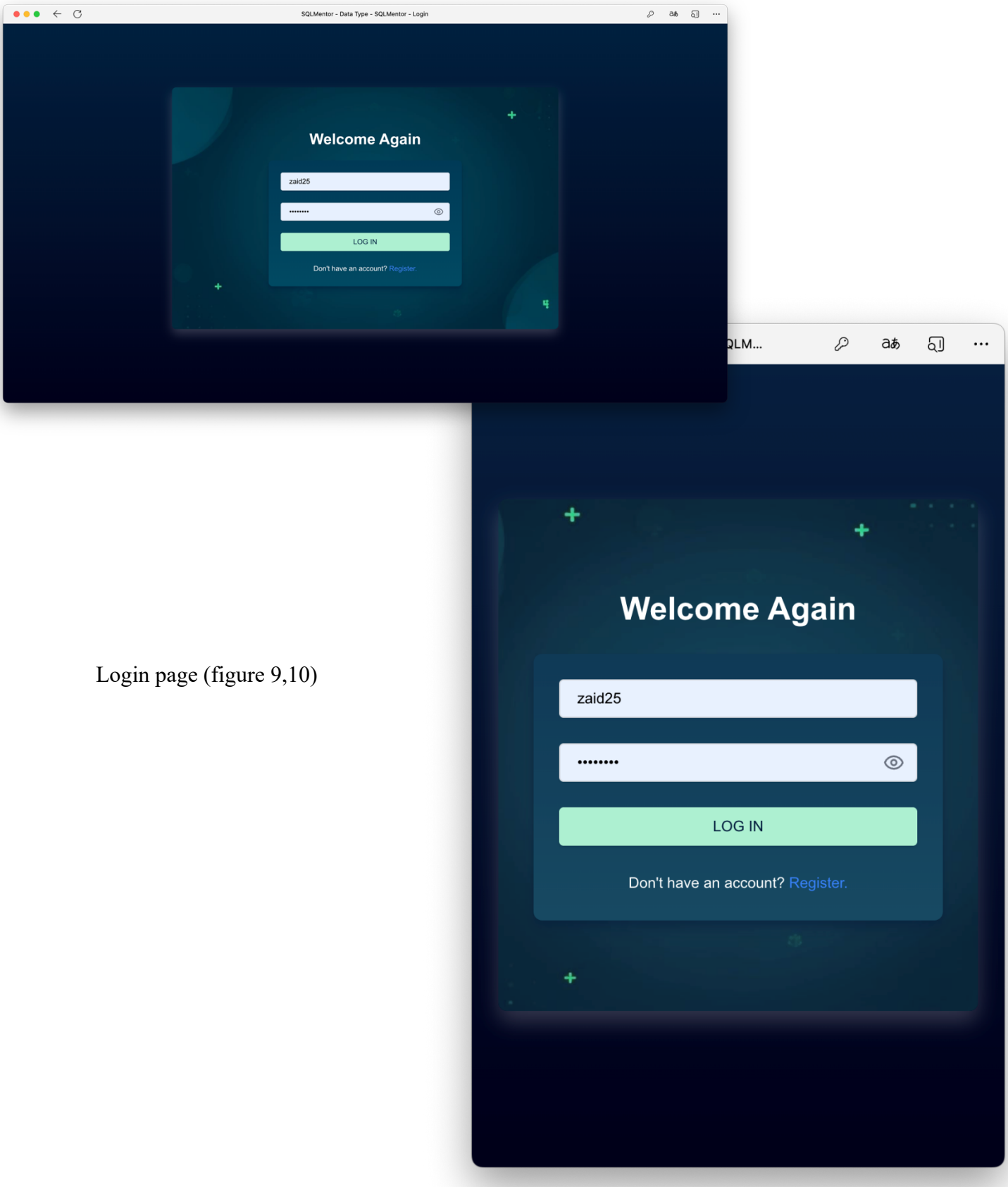
Learning page (figure 5,6)





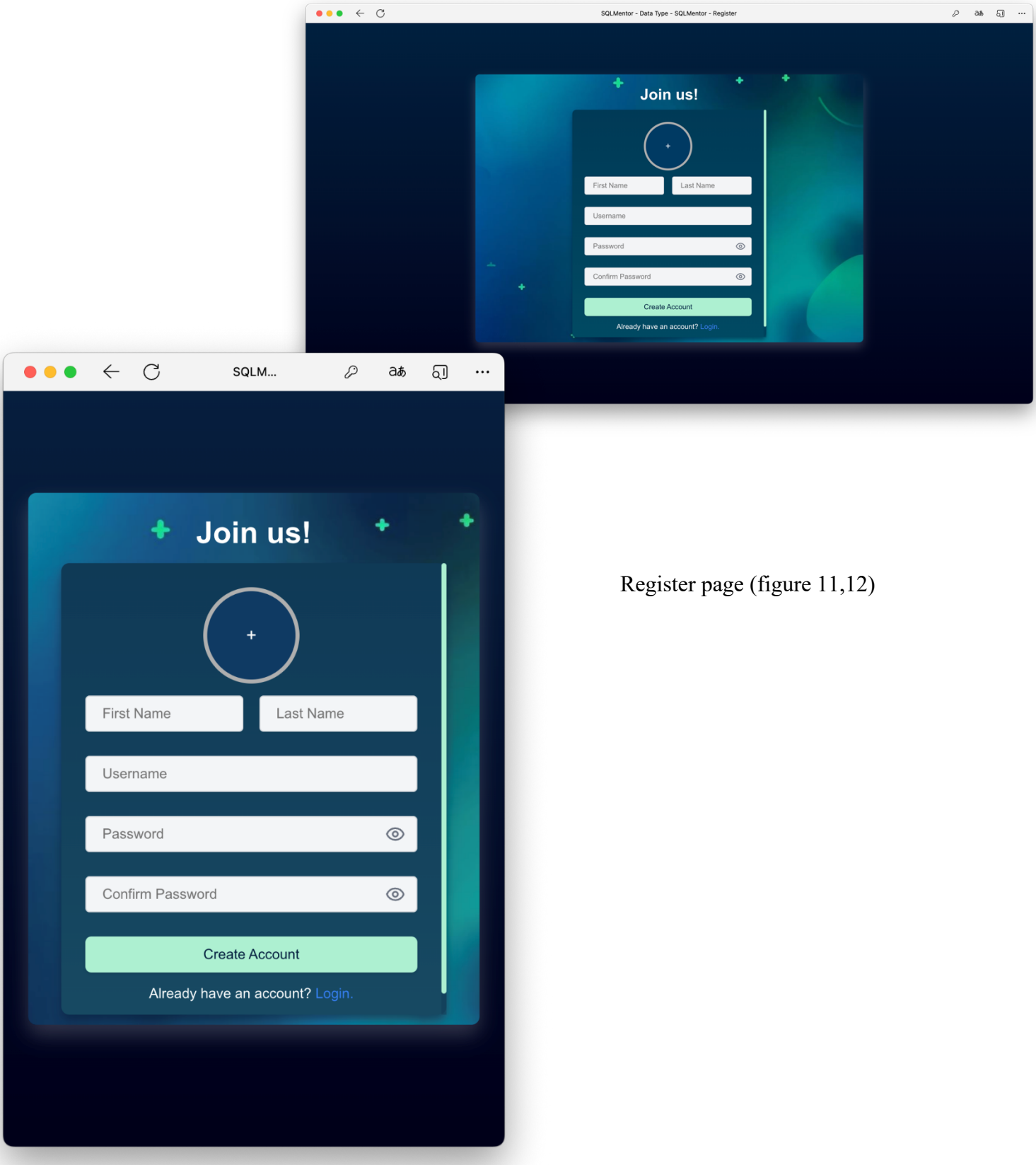
Learning page exercise (figure 7,8)

Login Page



Login page (figure 9,10)

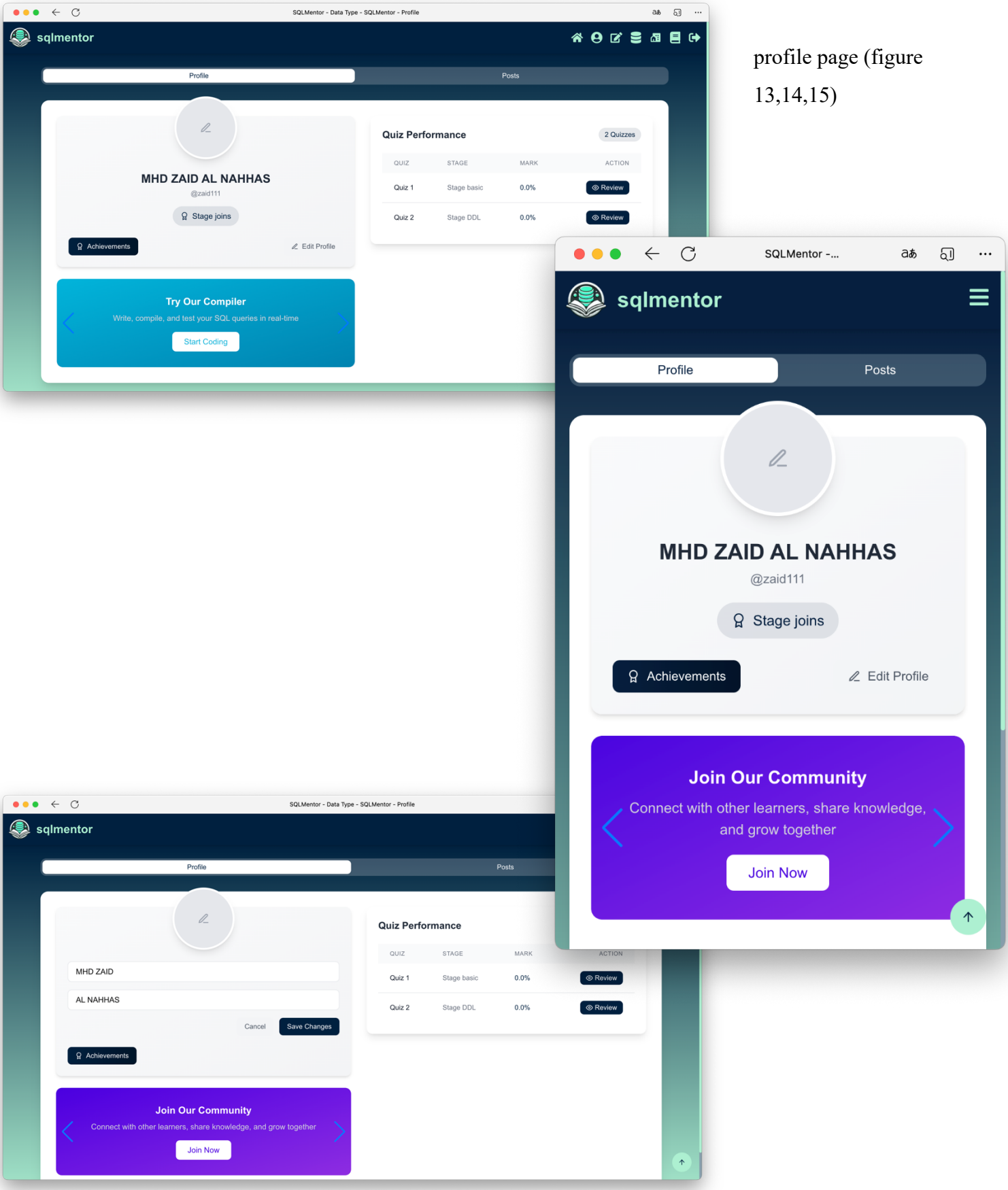
Register Page



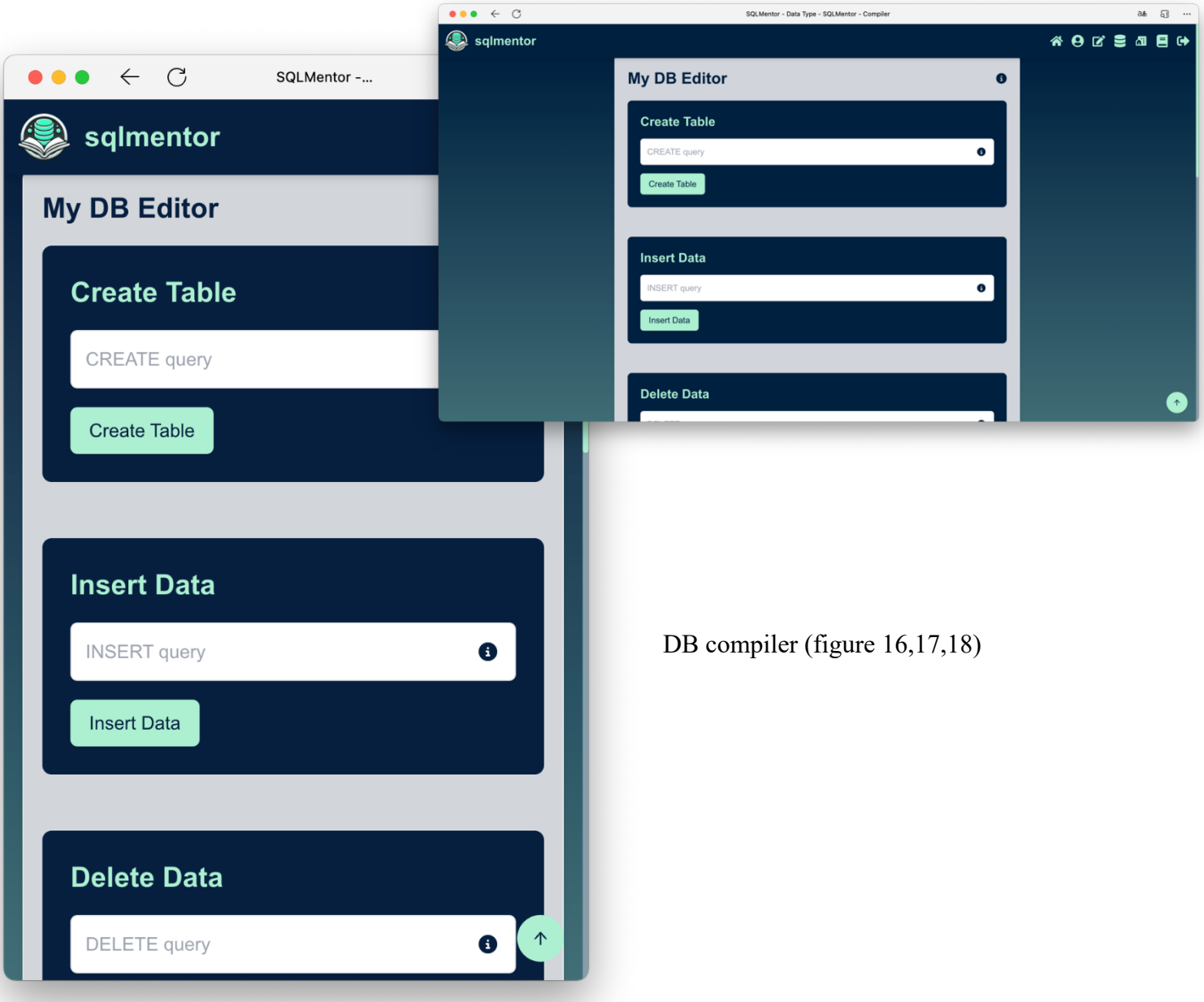
Register page (figure 11,12)

Profile page

profile page (figure 13,14,15)

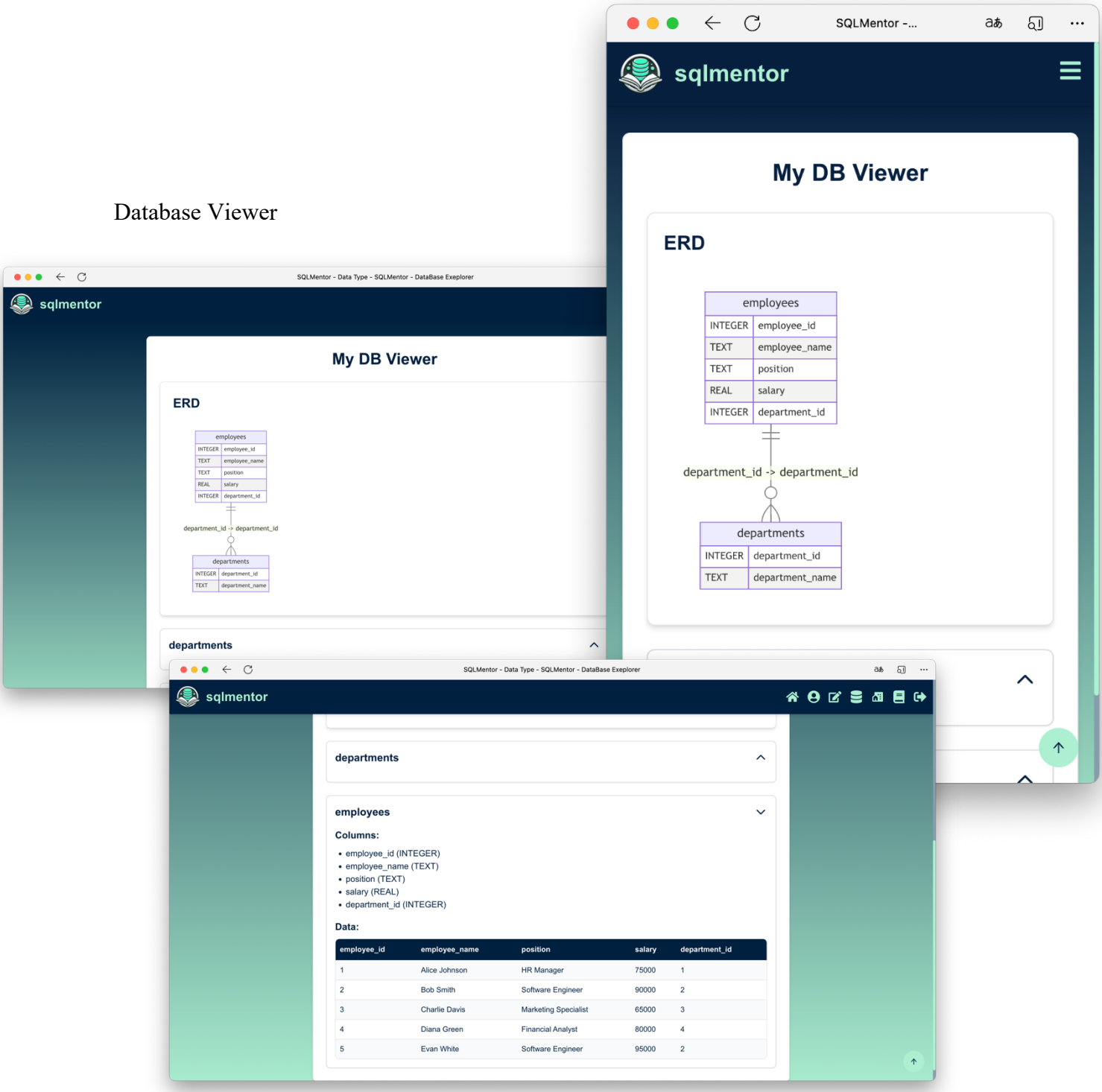


Database Compiler



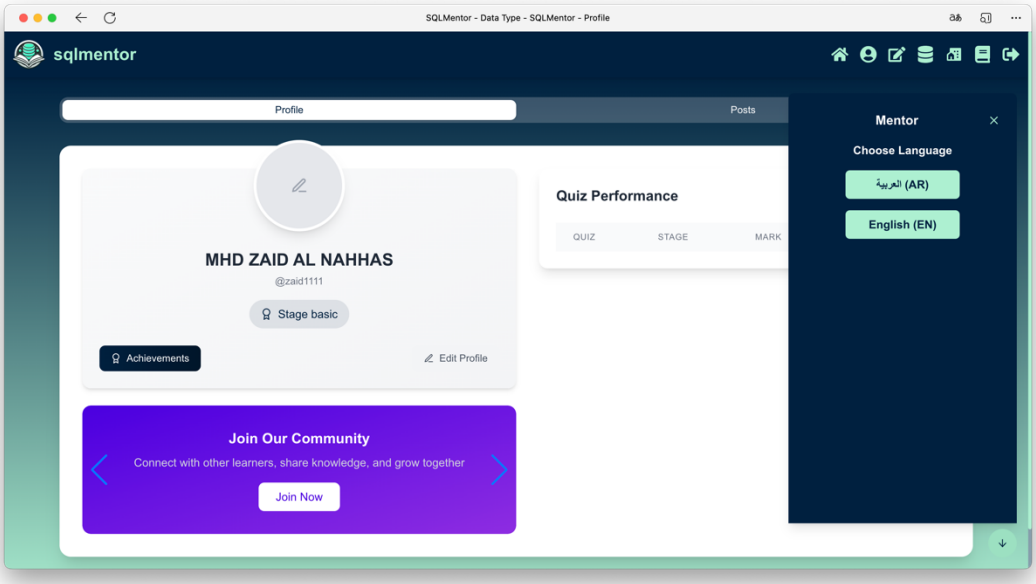
DB compiler (figure 16,17,18)

Database Viewer

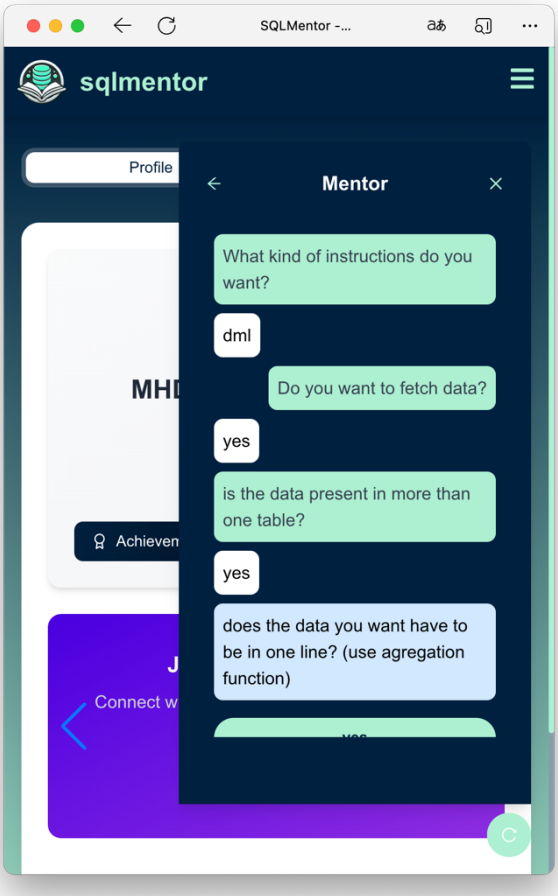


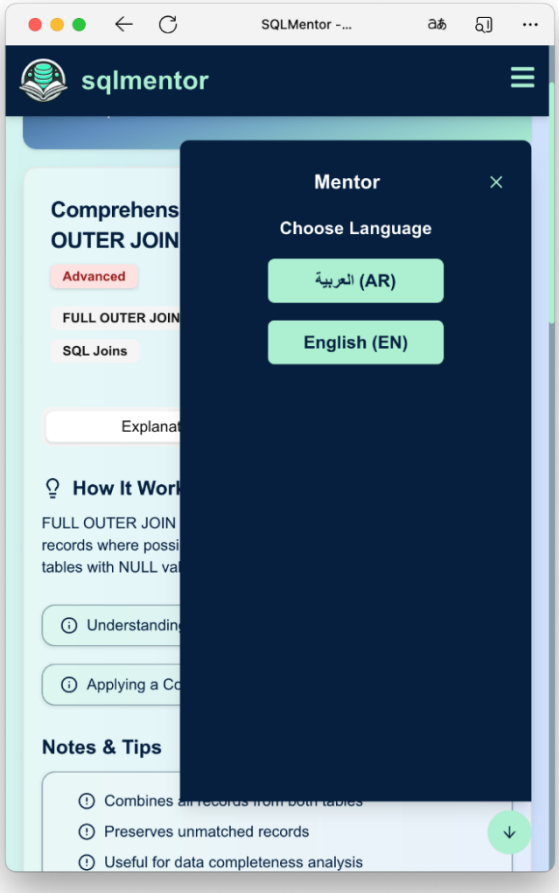
DB viewer (figure 19,20,21)

Chabot



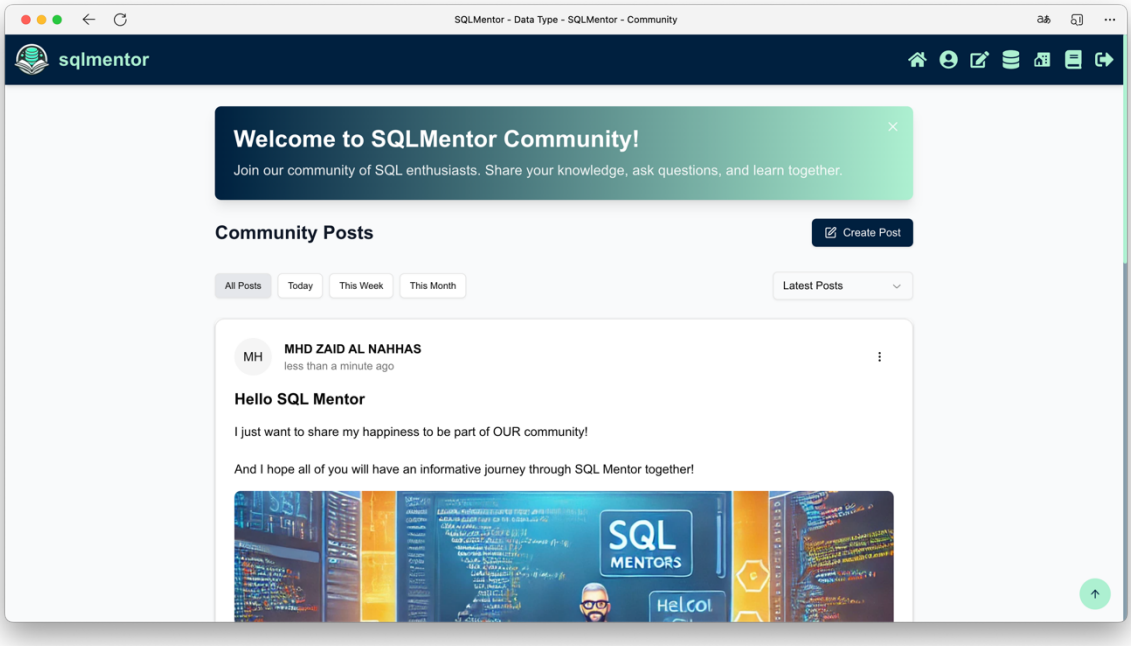
Chabot (figure 22,23,24)

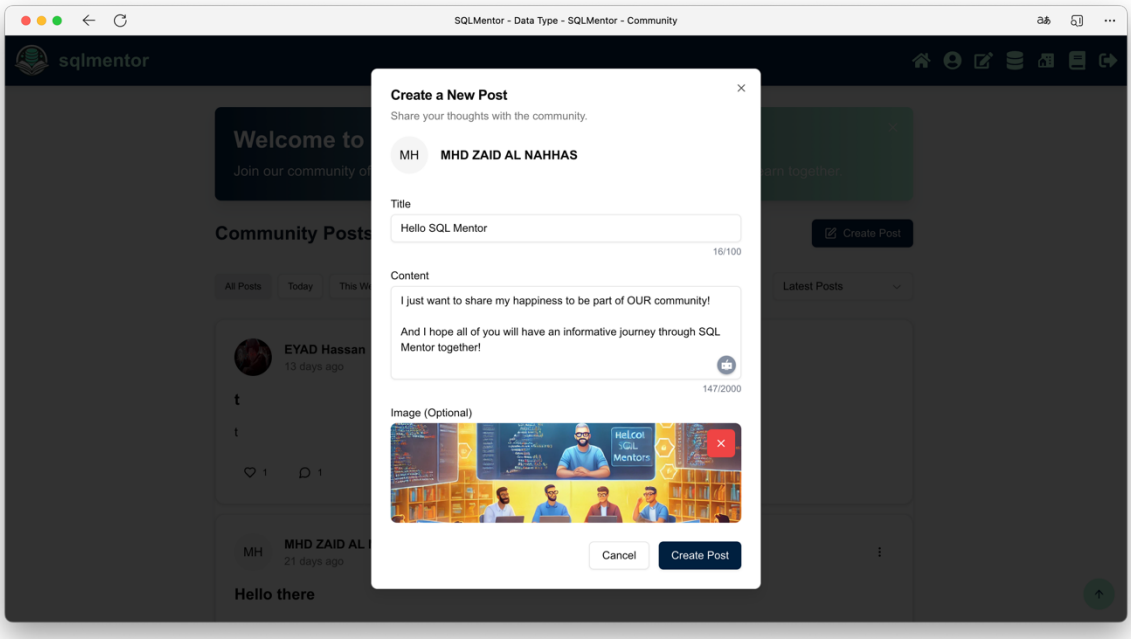




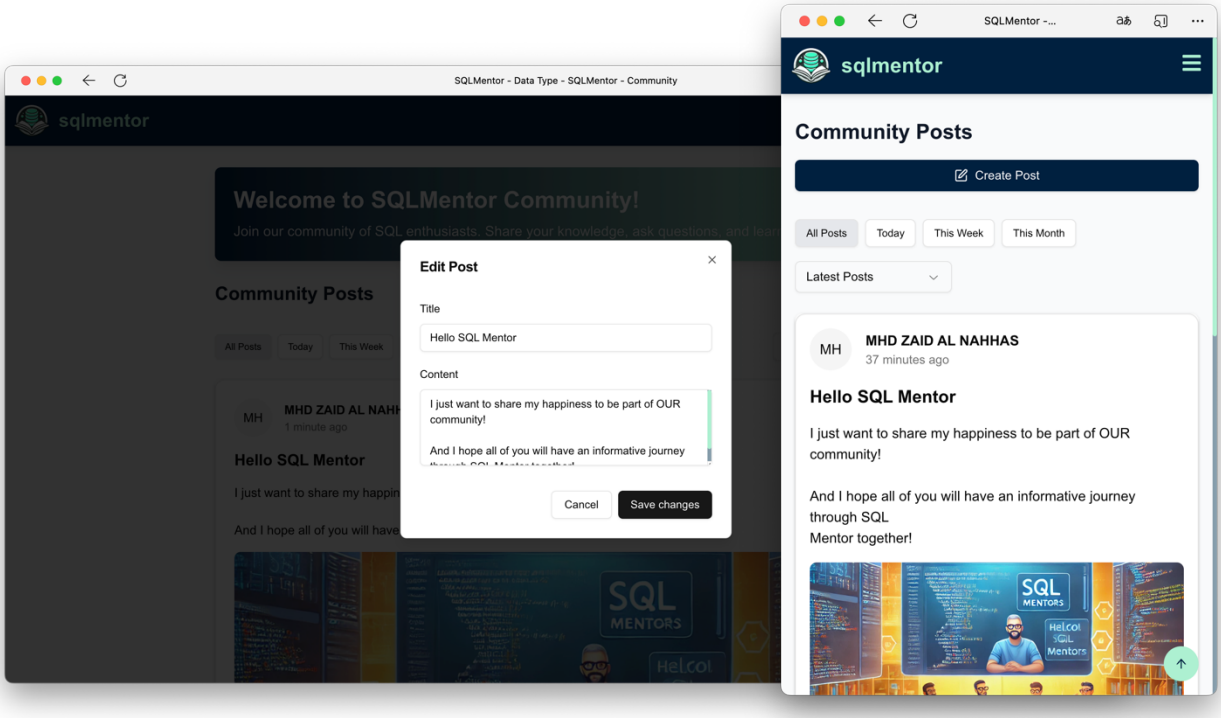
Community Page

community (figure 25,26)





community (figure 27,28)



5.5 Algorithms and Solutions

5.5.1 Chatbot Query Generation

Problem Description

This problem falls under Natural Language Processing (NLP), where we need to extract features from textual data such as the user's query and the context of tables in the database. Then, we use a trained model to generate the appropriate SQL query.

Algorithms Used

1. Fetching User Data
    - Upon receiving a query from the user, the table context is extracted from the user's database.
    - This information is used as input for the model.
2. Inputting Data into the Model
    - The query and table context are fed into the trained NLP model.
    - The model analyzes the inputs and predicts the appropriate SQL query.
3. Generating the Query Using the Model
    - The model has been trained on Text-to-SQL data and converts natural language questions into precise SQL queries.
    - It uses techniques such as sentence analysis and entity recognition to determine the relevant columns and tables in the database.
4. Returning the Results
    - The generated query is output to the user or executed on the database.

Technologies Used

- Pre-trained model**:** LoRA Fine-tuned Gemma 2B on Text-to-SQL data.

- Tokenization to convert words into numerical representations.
- Query prediction using AI.
- Semantic analysis to ensure accuracy.

Sample of code

```
# Limit the input sequence length to 512 (to control memory usage).
gemma_lm.preprocessor.sequence_length = 512
# Use AdamW (a common optimizer for transformer models).
optimizer = keras.optimizers.AdamW(
    learning_rate=5e-5,
    weight_decay=0.01,
)
# Exclude layernorm and bias terms from decay.
optimizer.exclude_from_weight_decay(var_names=["bias", "scale"])

gemma_lm.compile(
    loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    optimizer=optimizer,
    weighted_metrics=[keras.metrics.SparseCategoricalAccuracy()],
)
```

Start the training

```
gemma_lm.fit(data, epochs=1, batch_size=1)
```

```
1000/1000 ━━━━━━━━━━━━━━━━━━━━ 751s 729ms/step - loss: 0.2465 - sparse_categorical_accuracy: 0.6515

<keras.src.callbacks.history.History at 0x7b5d2807d4e0>
```

Training and preprocessor (figure 29,30)

```
def clean(text):
    return text.replace(u'\xa0', u' ').strip()

def prompt_fn(row):
    template = "Question:\n{question}\nContext:\n{context}\n\nAnswer:\n{answer}"
    prompt = template.format(
      question=clean(row['question']), context=clean(row['context']), answer=clean(row['answer'])
    )
    return prompt
```

We will take a subset of the dataset to use for training

```
LIMIT = 1000
data = raw_df[:LIMIT].apply(prompt_fn,axis=1).values.tolist()
```

5.5.2 Chatbot Syntax

Problem Description

This system functions as a Knowledge-Based System that helps users automatically generate SQL queries based on interactive responses. It supports DML (Data Manipulation Language) commands like SELECT, INSERT, UPDATE, DELETE, as well as DDL (Data Definition Language) commands like CREATE, ALTER, DROP, making it an interactive SQL compiler.

Algorithm Used

1. Starting the Conversation with the User
    - The system asks the user about the type of SQL command (DML or DDL)**.**
    - Based on the selection, it proceeds with a series of questions.
2. Collecting Information from the User
    - For DML Commands**:**
        - SELECT: Asks for columns (columns), table name (table), and conditions (WHERE conditions).
        - INSERT: Asks for table name (table) and values (values).
        - UPDATE: Asks for table name (table), updated values (SET values), and conditions (WHERE conditions).
        - DELETE: Asks for table name (table) and conditions (WHERE conditions).
    - For DDL Commands:
        - CREATE TABLE: Asks for table name (table_name), column names (columns), and data types (data types).
        - ALTER TABLE: Asks for table name (table_name) and modifications (add/drop columns, modify data types).
        - DROP TABLE: Asks for table name (table_name) to confirm before deletion.
3. Constructing the SQL Query

- After gathering the required information, the system dynamically generates a valid SQL query.
- It validates the syntax and ensures logical correctness.

4. Executing the Query in the User's Database
    - Each user has their own private database, ensuring that commands are executed only on their own data and tables.
    - The system ensures no interference with other users' data.

Technologies Used

- Knowledge-Based System utilizing a knowledge base for rule-based SQL generation.
- Interactive Dialogue System to analyze user requirements.
- Dynamic SQL Query Generation based on structured responses.
- Execution of DML and DDL queries within the user's environment, making the system function like a personal SQL Compiler.

Smaple of code

```
class lap(KnowledgeEngine):
    @DefFacts()
    def _initial_action(self):
        yield Fact(action="root")

    @Rule(Fact(action='root'))
    def Qroot(self):    "Qroot": Unknown word.
        self.declare(Fact(start=get_question({"q" : "What kind of instructions do you wants?" , "op1" : "dml" , "op2" : "ddl"})))

    @Rule(Fact(start='ddl'))
    def q1_1(self):
        self.declare(Fact(q1=get_question({"q" : "do you want to modify an old table?" ,  "op1" : "yes" , "op2" : "no"})))

    @Rule(Fact(q1='yes'))
    def q2_1(self):
        chose_table({"q" : "chose table"})
        self.declare(Fact(q2=get_question({"q" : "do you want to delete a table?" ,  "op1" : "yes" , "op2" : "no"})))

    @Rule(Fact(q2='yes'))
    def q3_1(self):
        self.declare(Fact(q3=send_syntax(f"drop table {table_name};")))

    @Rule(Fact(q2='no'))
    def q3_2(self):
        self.declare(Fact(q3=get_question({"q" : "do you want to delete or add a column?" ,  "op1" : "delete" , "op2" : "add"})))
```

Expert Chabot (figure 31)

5.5.3 Score Calculator

Problem Description

This system calculates the similarity between two sentences using Cosine Similarity between sentence embeddings. The primary goal is to evaluate user responses in quizzes and compare their answers in interactive exercises during learning.

Algorithm Used

1. Using SentenceTransformer Model
    - Based on all-mpnet-base-v2 to generate numerical representations (embeddings) of sentences.
2. Receiving API Requests
    - The user sends two sentences (sentence1 and sentence2) via a POST request to /compare.
3. Generating Sentence Embeddings
    - Each sentence is converted into a numerical vector representing its meaning.
4. Calculating Cosine Similarity
    - Cosine similarity is computed between the two vectors using the formula:

$$\cos(\theta) = \frac{A.B}{\|A\| \times \|B\|}$$

Where:

- $A.B$ is the dot product of vectors A and B.
- $\|A\|$ and $\|B\|$ are the magnitudes of vectors A and B, respectively.

The cosine similarity value ranges between -1 and 1:

- 1: Vectors are perfectly similar (parallel).
- 0: Vectors are orthogonal (no similarity).
- -1: Vectors are completely opposite.

5. Returning the Result
    - The similarity score is returned as a numerical value between 0 and 1.

Use Cases in the System

- Quiz Answer Evaluation
    - Compares the user's answer (userAns) with the correct answer (realAns).
    - The closer the score is to 1.0, the more similar the answers are.
- Evaluating Interactive Exercises
    - Used in learning exercises to verify correctness.

Smaple of code

```
try:
    data = request.get_json()
    sentence1 = data.get('sentence1')
    sentence2 = data.get('sentence2')
    if not sentence1 or not sentence2:
        return jsonify({'error': 'Please provide both sentences'}), 400
    embeddings = sentence_model.encode([sentence1, sentence2])
    cosine_sim = cosine_similarity([embeddings[0]], [embeddings[1]])
    cosine_sim_value = float(cosine_sim[0][0])
    return jsonify({'cosine_similarity': cosine_sim_value})

except Exception as e:
    return jsonify({'error': str(e)}), 500
```

Score calculation (figure 32,33)

```
from sentence_transformers import SentenceTransformer
from sklearn.metrics.pairwise import cosine_similarity

model = SentenceTransformer('sentence-transformers/all-mpnet-base-v2')

sentences = ['SELECT name FROM person WHERE age > 27', 'SELECT name FROM student WHERE age  < 27']
embeddings = model.encode(sentences)
cosine_sim = cosine_similarity([embeddings[0]], [embeddings[1]])

print(f"Cosine similarity between the sentences: {cosine_sim[0][0]}")
```

```
Cosine similarity between the sentences: 0.8661059141159058
```

5.5.4 Quiz Generation

Problem Description

This system generates quizzes based on user performance across different levels within a stage. Each stage consists of multiple levels, and at the end of the stage, a quiz is generated to evaluate the user's understanding.

During each level, the user learns specific content and completes exercises. The system verifies that the returned data matches the expected output from the ideal answer. If correct, the Score Calculator evaluates the accuracy.

Algorithm Used

1. Tracking User Performance
    - Four parameters are tracked for each level:
        - Time taken to solve the exercise.

- Number of attempts made.
- Whether the "Show Answer" button was used.
- Accuracy of the answer (calculated using the Score Calculator).

2. Calculating Quiz Questions
   - At the end of the stage, stored parameters are retrieved, and a formula determines the number of quiz questions:

$$Question\ Count = w_1 \times Accuracy + w_2 \times Time + w_3 \times Attempts + w_4 \times Show\ Answer\ Usage$$

Weights:

- $w_1 = -0.3\ (accuracy)$
- $w_2 = 0.2\ (time)$
- $w_3 = 0.2\ (attempts)$
- $w_4 = 1000\ (show\ answer\ usage)$

   - Values are normalized, and the final count is capped between 0 and 8 questions per level.
- Fetching Questions from the Question Bank
  - Questions are retrieved based on calculated counts and presented interactively.
- Evaluating User Answers
  - The Score Calculator computes the accuracy of each answer.
- Calculating the Quiz Score
  - Accuracy scores are averaged and scaled to a percentage (out of 100).
- Returning the Quiz Result
  - The final score is displayed along with feedback.

Use Cases in the System

- Adaptive Quiz Generation
- Performance Evaluation
- Balanced Question Distribution
- Answer Evaluation
- Quiz Scoring
- User Feedback

Sample of code

```
const user = await getUser();
if (!user) return { field: "root", message: "User not authenticated." };

const userParams = await db.query.TB_user_excercise_summary.findMany({
    where: (u, { eq }) => eq(u.userId, user.id),
});

const bestUserParams = userParams.reduce(
    (acc, curr) => {
        const existing = acc[curr.levelId];
        const currentScore = parseFloat(curr.score!);    non-null-expression
        if (
            !existing ||
            (currentScore && parseFloat(existing.score!) < currentScore)    no
        ) {
            acc[curr.levelId] = { ...curr, score: currentScore.toString() };
        }
        return acc;
    },
    {} as Record<string, (typeof userParams)[0]>,    record-utility-type
);
```

```
const maxTime = Math.max(...userParams.map((param) => param.time!));
const maxTrials = Math.max(...userParams.map((param) => param.trials!));

const w1 = -0.3;
const w2 = 0.2;
const w3 = 0.2;
const w4 = 1000;

const levelQuestionsCount = levels.map((level) => {
    const userLevelData = bestUserParams[level.id];
    if (!userLevelData) {
        return { levelId: level.id, questionCount: 8 };
    }

    const { score, time, trials, is_show_ans } = userLevelData;

    const normalizedScore =
        score === null ? 0 : Math.min(1, parseFloat(score) / 100);
    const normalizedTime = time! / maxTime;    non-null-expression
    const normalizedTrials = trials! / maxTrials;    non-null-expression
    const normalizedIsShowAns = is_show_ans ? 1 : 0;

    let questionCount =
        w1 * normalizedScore +
        w2 * normalizedTime +
        w3 * normalizedTrials +
        w4 * normalizedIsShowAns;

    questionCount = Math.min(Math.max(questionCount, 0), 8);
```

Quiz generation (figure 36,37)

5.5.5 User SQL Compiler

Problem Description

This system provides users with a personalized SQL compiler to create, execute, and save DML and DDL commands. Each user has a dedicated SQLite database, created upon first access, ensuring isolation from the main system database. The system integrates with NLP Chatbot and Chatbot Syntax features to enhance query generation.

Algorithm Used

1. Creating the User Database
    - A personal SQLite database is created and associated with the user's account.
2. Executing and Saving SQL Commands
    - Users can write and execute SQL commands in their database.
    - Commands are saved in history for reference.
3. Integrating with Chatbot Features
    - NLP Chatbot: Extracts table context for SQL query generation.
    - Chatbot Syntax: Guides interactive SQL query generation.
4. Displaying Tables and Data
    - Users can view tables and stored data.
5. Generating ERD Diagrams
    - The Mermaid library dynamically generates an ERD diagram from the user's database schema.

Use Cases in the System

- Personal SQL Compiler
- Table Context Integration
- Data and Schema Visualization
- Isolated Execution Environment

Example Workflow

1. A user accesses their SQL compiler.
2. Creates a table:
3. CREATE TABLE employees (id INT, name TEXT, department TEXT);

4. Inserts data:
5. INSERT INTO employees (id, name, department) VALUES (1, 'John Doe', 'HR');
6. Queries the table:
7. SELECT * FROM employees;
8. In NLP Chatbot, the user asks: "Show all employees in HR," and the system generates:
9. SELECT * FROM employees WHERE department = 'HR';
10. The Mermaid library generates an ERD diagram displaying the employees table.

This feature provides an isolated SQL environment, integrates with chatbot features, and enhances learning through interactive query execution and visualization.

Sample of code

```
export async function GET(req: NextRequest) {    typing-function-parameters
    try {
        const db = await getDb();
        const tables = await db.all(
            `SELECT name FROM sqlite_master WHERE type='table' AND name NOT LIKE 'sqlite_%';`,
        );
        const tablesWithColumnsAndData = await Promise.all(
            tables.map(async (table: { name: string }) => {    typing-function-parameters
                const columns = await db.all(`PRAGMA table_info(${table.name});`);
                const data = await db.all(`SELECT * FROM ${table.name}`);
                return {
                    tableName: table.name,
                    columns: columns.map((col: any) => ({    typing-function-parameters
                        columnName: col.name,
                        columnType: col.type,
                    })),
                    data,
                };
            }),
        );
        return NextResponse.json(tablesWithColumnsAndData, { status: 200 });
    } catch (error) {
        console.error("Database error:", error);
        return NextResponse.json(
            {
                error: "حدث خطأ أثناء جلب الجداول والبيانات.",
                details: error instanceof Error ? error.message : String(error),
            },
            { status: 500 },
        );
    }
}
```

User compiler (figure 34,35)

```
switch (queryType) {
    case "select": {
        try {
            const rows = await db.all(query);
            return NextResponse.json({ data: rows }, { status: 200 });
        } catch (error) {
            console.error("SELECT query error:", error);
            return NextResponse.json(
                {
                    error: "فشل استعلام SELECT. تأكد من صحة الجدول أو الأعمدة.",
                    originalError:
                        error instanceof Error ? error.message : String(error),
                },
                { status: 500 },
            );
        }
    }
```

5.5.6 translation

Problem Description

translation is a tool that converts user questions from Arabic to English. This is important because AI models perform better when questions are in English. When a user types a question in Arabic, the translation API takes that question and translates it into English. The AI model then processes the English question and provides a response. This way, Arabic-speaking users can easily interact with the AI, ensuring better performance and understanding.

Algorithm Used

Load Model and Tokenizer:

Load MarianTokenizer and MarianMTModel for Arabic-to-English translation.

Tokenize Input:

Tokenize Arabic text (text_ar) into tensors with padding and truncation.

Generate Translation:

Use the model to generate English token IDs with beam search (num_beams=4) and early stopping.

Decode Output:

Decode token IDs into English text, skipping special tokens.

Key Parameters:

max_length=512: Limits translation length.

num_beams=4: Improves translation quality.

early_stopping=True: Stops generation early if needed.

skip_special_tokens=True: Cleans up output text.

Sample of code

```
ar_to_en_model_name = 'Helsinki-NLP/opus-mt-ar-en'
ar_to_en_tokenizer = MarianTokenizer.from_pretrained(ar_to_en_model_name)
ar_to_en_model = MarianMTModel.from_pretrained(ar_to_en_model_name)
        inputs = ar_to_en_tokenizer.encode(text_ar, return_tensors="pt", padding=True,
truncation=True)
        translated = ar_to_en_model.generate(inputs, max_length=512, num_beams=4,
early_stopping=True)
translated_text = ar_to_en_tokenizer.decode(translated[0], skip_special_tokens=True)
```

5.6 Technical Stack Overview

Frontend Development

1. Next.js
    - Serves as the primary framework for server-side rendering (SSR) and static site generation (SSG).
    - Enables seamless routing, API integration, and optimized performance.
    - Manages authentication, state, and dynamic content rendering.
2. React.js
    - Powers the component-based UI architecture for reusable and modular design.
    - Handles interactive features like quizzes, database editors, and real-time updates.
    - Integrates with Next.js for hybrid static/dynamic applications.
3. TypeScript
    - Enhances code reliability with static type checking.
    - Reduces runtime errors in complex features (e.g., SQL query builders, chatbots).
    - Improves collaboration through clear type definitions.

Backend Development

1. Next.js API Routes
    - Handles lightweight backend logic (e.g., user authentication, exercise validation).
    - Connects frontend components to databases and external services.
2. Next.js Server Actions
    - Manages secure server-side operations (e.g., form submissions, profile edits).
    - Directly interacts with databases without exposing client-side logic.
3. Flask API (Python)
    - Runs advanced Python-based tasks (e.g., AI-driven SQL query, data analysis).
    - Communicates with the Next.js frontend via RESTful endpoints.

Python Development Tools

1. Jupyter Notebook & Google Colab
    - Used for prototyping AI/ML algorithms and data analysis.
    - Tests SQL-related logic (e.g., query optimization, schema design).
    - Integrates with Flask to deploy trained models into production.

Deployment & Infrastructure

1. Docker
    - Containerizes the entire application (frontend, backend, Flask services).
    - Ensures consistent environments across development, staging, and production.
    - Simplifies scalability using orchestration tools like Kubernetes.
2. GitHub Desktop
    - Manages version control for collaborative development.
    - Tracks changes across frontend (Next.js/React), backend (Flask), and infrastructure (Docker) codebases.
    - Facilitates CI/CD pipelines for automated testing and deployment.

Integration Workflow

- Frontend: Next.js + React + TypeScript creates a responsive UI, fetching data from Next.js API routes and Server Actions.

- Backend: Next.js handles basic CRUD operations, while Flask manages compute-heavy tasks (e.g., AI/ML).
- Python Tools: Jupyter/Colab prototypes are deployed via Flask, with Docker ensuring seamless execution.
- DevOps: GitHub Desktop coordinates code changes, while Docker containers ensure reliable deployments.

Chapter 6: Testing

6.1 Data Sets

Data Source

The dataset was obtained from Hugging Face. It includes SQL queries designed to convert natural language questions into executable database queries , which can be accessed at the following link:
knowrohit07/know_sql.

Importance of the Dataset in Training

This dataset was used to train the Gemma-2B model to generate SQL queries based on table schemas and user questions.

How the Data Was Used in the Project

- Converting Questions to SQL:
  - A natural language question is given, the table schema is analyzed, and the appropriate SQL query is generated.
- Analyzing Table Schema Structures:
  - The model was trained to understand table schemas and generate correct queries based on field relationships.
- Enhancing Model Performance:
  - The LoRA (Low-Rank Adaptation) technique was applied to improve result quality while optimizing computational resource consumption.

Dataset Structure

- The dataset consists of 3 columns:
  - answer: Contains the generated SQL query.
  - context: Contains the table schema used in the query.
  - question: Contains the natural language question.
- The dataset includes 49,456 rows, providing a large set of examples for training and evaluation.

This dataset includes SQL queries designed to convert natural language questions into executable database queries. It plays a fundamental role in training the **Gemma-2B** model to generate SQL queries based on table schemas and user queries.

```
print(f'Total number of samples: {len(raw_df)}')
```

Python

```
Total number of samples: 49456
```

```
raw_df.head()
```

Python

	answer	context	question
0	SELECT COUNT(district) FROM table_1341586_19 WHERE incumbent = "Lindy Boggs"	CREATE TABLE table_1341586_19 (district VARCHAR, incumbent VARCHAR)	how many district with incumbent being lindy boggs
1	SELECT result FROM table_1341586_19 WHERE candidates = "Billy Tauzin (D) Unopposed"	CREATE TABLE table_1341586_19 (result VARCHAR, candidates VARCHAR)	what's the result with candidates being billy tauzin (d) unopposed
2	SELECT COUNT(candidates) FROM table_1341586_19 WHERE result = "Retired to run for U. S. Senate Republican hold"	CREATE TABLE table_1341586_19 (candidates VARCHAR, result VARCHAR)	how many candidates with result being retired to run for u. s. senate republican hold
3	SELECT result FROM table_1341586_19 WHERE district = "Louisiana 2"	CREATE TABLE table_1341586_19 (result VARCHAR, district VARCHAR)	what's the result with district being louisiana 2
4	SELECT candidates FROM table_1341586_19 WHERE first_elected = 1977	CREATE TABLE table_1341586_19 (candidates VARCHAR, first_elected VARCHAR)	who is the the candidates with first elected being 1977

6.2 Testing

Test Case ID	Test Condition	Expected Results	Actual Results	Status (Pass/Fail)
TC_001	Launching the system for the first time	The home page is displayed, and clicking "Get Started" redirects to the login page	The home page was displayed correctly, and clicking "Get Started" redirected to the login page	Pass
TC_002	Logging in for the first time	After logging in, the user is redirected to the first level of the learning pages	After logging in, the user was successfully redirected to the first level	Pass

			of the learning pages	
TC_003	Opening the website later	If the user was previously logged in, the home page opens while keeping the login session active via cookies unless the user had logged out before	The home page opened with the user still logged in due to cookies, and logging out removed the session	Pass
TC_004	Attempting to access a stage level that the user has not reached yet	The page will not open, and a message will inform the user that they have not reached this level yet	The page was blocked, and the user was notified that they had not reached this level yet	Pass
TC_005	Attempting to access a quiz for a stage the user has not reached yet	The quiz page will not open, and a message will inform the user that they need to reach this level first	The quiz page was blocked, and the user was notified they must reach the level first	Pass

TC_006	Attempting to access the login or sign-up page while already logged in	The user is redirected to the home page since cookies indicate an active login session	The user was redirected to the home page due to stored login cookies	Pass
TC_007	Attempting to access the profile, compiler, a learning stage, or the user's database page without logging in	The user is redirected to the login page	The user was redirected to the login page	Pass
TC_008	Updating account information (name or profile picture) from the profile page	The information is successfully updated	The information was successfully updated	Pass
TC_009	Opening a previously taken quiz from the profile page and viewing the result	The quiz page opens successfully, displaying the result	The quiz page opened successfully, displaying the result	Pass

TC_010	Editing, adding, or deleting user posts from the community page	Posts are successfully edited, added, or deleted	Posts were successfully edited, added, or deleted	Pass
TC_011	Liking a post, adding a comment, or liking a comment on a post	The like, comment, or like on a comment is successfully registered	The like, comment, and like on a comment were successfully registered	Pass
TC_011	Liking a post, adding a comment, or liking a comment on a post	The like, comment, or like on a comment is successfully registered	The like, comment, and like on a comment were successfully registered	Pass
TC_012	Attempting an exercise in a learning stage (SQL query type)	The query executes successfully, and the answer with its accuracy is displayed	The query executed successfully, and the answer with its accuracy was displayed	Pass
TC_013	Attempting an exercise in a learning stage (True/False, MCQ, Drag & Drop)	The exercise executes successfully, and the answer is displayed	The exercise executed successfully, and the answer was displayed	Pass

TC_014	Submitting a quiz and sending answers	The answers are collected, accuracy is calculated, and the quiz result is displayed successfully	The answers were collected, accuracy was calculated, and the quiz result was displayed successfully	Pass
TC_015	Writing a table creation statement in the compiler page	The table is created successfully in the user's database	The table was created successfully in the user's database	Pass
TC_016	Executing other DDL (CREATE, ALTER, DROP) and DML (INSERT, UPDATE, DELETE) statements in the compiler	The statements execute successfully in the user's database	The statements executed successfully in the user's database	Pass
TC_017	Executing a SELECT statement in the compiler page	The query executes successfully, and the result is displayed	The query executed successfully, and the result was displayed	Pass

TC_018	Opening the user's database browser page	The ERD schema of the user's database is displayed, along with tables, columns, data types, and stored data	The ERD schema, tables, columns, and stored data were displayed successfully	Pass
TC_019	Testing chatbot syntax for CREATE statement	The chatbot asks relevant questions and generates the CREATE statement	The chatbot asked questions and generated the CREATE statement	Pass
TC_020	Testing chatbot syntax for ALTER statement	The chatbot asks relevant questions and generates the ALTER statement based on existing tables	The chatbot asked questions and generated the ALTER statement	Pass
TC_021	Testing chatbot syntax for DROP statement	The chatbot asks relevant questions and generates the DROP statement based on existing tables	The chatbot asked questions and generated the DROP statement	Pass

TC_022	Testing chatbot syntax for TRUNCATE statement	The chatbot asks relevant questions and generates the TRUNCATE statement based on existing tables	The chatbot asked questions and generated the TRUNCATE statement	Pass
TC_023	Testing chatbot syntax for INSERT statement	The chatbot asks relevant questions and generates the INSERT statement with specified values	The chatbot asked questions and generated the INSERT statement	Pass
TC_024	Testing chatbot syntax for DELETE statement	The chatbot asks relevant questions and generates the DELETE statement, applying conditions if specified	The chatbot asked questions and generated the DELETE statement	Pass
TC_025	Testing chatbot syntax for UPDATE statement	The chatbot asks relevant questions and generates the UPDATE statement, applying	The chatbot asked questions and generated the UPDATE statement	Pass

		conditions if specified		
TC_026	Testing chatbot syntax for SELECT statement with WHERE clause	The chatbot asks relevant questions and generates a SELECT statement with a WHERE condition	The chatbot asked questions and generated the SELECT statement with WHERE	Pass
TC_027	Testing chatbot syntax for SELECT statement with ORDER BY	The chatbot asks relevant questions and generates a SELECT statement with ORDER BY	The chatbot asked questions and generated the SELECT statement with ORDER BY	Pass
TC_028	Testing chatbot syntax for SELECT statement with GROUP BY	The chatbot asks relevant questions and generates a SELECT statement with GROUP BY	The chatbot asked questions and generated the SELECT statement with GROUP BY	Pass
TC_029	Testing chatbot syntax for SELECT statement with aggregate functions (MIN, MAX,	The chatbot asks relevant questions and generates a SELECT statement using	The chatbot asked questions and generated the SELECT statement with aggregate functions	Pass

	COUNT, SUM, AVG)	aggregate functions		
TC_030	Testing chatbot syntax for SELECT statement with JOIN	The chatbot asks relevant questions and generates a SELECT statement using JOIN	The chatbot asked questions and generated the SELECT statement with JOIN	Pass
TC_031	Testing chatbot syntax for CREATE statement in Arabic	The chatbot asks relevant questions in Arabic and generates the CREATE statement	The chatbot asked questions in Arabic and generated the CREATE statement	Pass
TC_032	Testing chatbot syntax for ALTER statement in Arabic	The chatbot asks relevant questions in Arabic and generates the ALTER statement	The chatbot asked questions in Arabic and generated the ALTER statement	Pass
TC_033	Testing chatbot syntax for DROP statement in Arabic	The chatbot asks relevant questions in Arabic and generates the DROP statement	The chatbot asked questions in Arabic and generated the DROP statement	Pass

TC_034	Testing chatbot syntax for TRUNCATE statement in Arabic	The chatbot asks relevant questions in Arabic and generates the TRUNCATE statement	The chatbot asked questions in Arabic and generated the TRUNCATE statement	Pass
TC_035	Testing chatbot syntax for INSERT statement in Arabic	The chatbot asks relevant questions in Arabic and generates the INSERT statement	The chatbot asked questions in Arabic and generated the INSERT statement	Pass
TC_036	Testing chatbot syntax for DELETE statement in Arabic	The chatbot asks relevant questions in Arabic and generates the DELETE statement	The chatbot asked questions in Arabic and generated the DELETE statement	Pass
TC_037	Testing chatbot syntax for UPDATE statement in Arabic	The chatbot asks relevant questions in Arabic and generates the UPDATE statement	The chatbot asked questions in Arabic and generated the UPDATE statement	Pass
TC_038	Testing chatbot syntax for SELECT statement	The chatbot asks relevant questions in Arabic and	The chatbot asked questions in Arabic and	Pass

	with WHERE in Arabic	generates the SELECT statement with WHERE condition	generated the SELECT statement with WHERE	
TC_039	Testing chatbot syntax for SELECT statement with ORDER BY in Arabic	The chatbot asks relevant questions in Arabic and generates the SELECT statement with ORDER BY	The chatbot asked questions in Arabic and generated the SELECT statement with ORDER BY	Pass
TC_040	Testing chatbot syntax for SELECT statement with GROUP BY in Arabic	The chatbot asks relevant questions in Arabic and generates the SELECT statement with GROUP BY	The chatbot asked questions in Arabic and generated the SELECT statement with GROUP BY	Pass
TC_041	Testing chatbot syntax for SELECT statement with aggregate functions (MIN, MAX, COUNT, SUM, AVG) in Arabic	The chatbot asks relevant questions in Arabic and generates the SELECT statement with aggregate functions	The chatbot asked questions in Arabic and generated the SELECT statement with aggregate functions	Pass

TC_042	Testing chatbot syntax for SELECT statement with JOIN in Arabic	The chatbot asks relevant questions in Arabic and generates the SELECT statement with JOIN	The chatbot asked questions in Arabic and generated the SELECT statement with JOIN	Pass
TC_043	Executing the generated SQL statement from chatbot syntax	The system successfully executes the generated SQL statement	The system executed the SQL statement successfully	Pass
TC_044	Testing chatbot code generation using a question and context from the user's database	The chatbot generates the correct SQL query based on the given context, with an option to execute it	The chatbot generated the correct SQL query with an execution option	Pass
TC_045	Executing the SQL query generated by the chatbot code generation	The system successfully executes the SQL query and returns the correct results	The system executed the query and returned the correct results	Pass
TC_046	Testing chatbot code generation	The chatbot generates an SQL query	The chatbot generated the SQL query	Pass

	with a new context provided by the user	but does not provide an execution option since the tables do not exist in the database	without an execution option	
TC_047	Executing the query generated in the previous test case	Execution is not allowed, and the system informs the user that the tables do not exist	The system displayed an error message indicating that the tables do not exist	Pass
TC_048	Testing all previous chatbot code generation scenarios in Arabic, with the user inputting questions in Arabic	The chatbot generates the correct SQL query in response to the Arabic question, considering the database context	The chatbot generated the correct SQL query based on the Arabic question	Pass
TC_049	Executing the SQL query generated by the chatbot in Arabic	The system successfully executes the SQL query and returns the correct results	The system executed the query and returned the correct results	Pass

TC_050	Testing chatbot code generation with a new user-provided context in Arabic	The chatbot generates an SQL query but does not provide an execution option since the tables do not exist in the database	The chatbot generated the SQL query without an execution option	Pass
TC_051	Executing the query generated in the previous Arabic test case	Execution is not allowed, and the system informs the user that the tables do not exist	The system displayed an error message indicating that the tables do not exist	Pass

6.3 Results

All test cases were executed successfully, and the system performed as expected. Each feature was tested under various conditions to ensure functionality, usability, and reliability. The results indicate that:

- The system correctly handles user authentication, navigation, and session management.
- Access control mechanisms work as intended, preventing users from accessing unauthorized content.
- The learning modules, exercises, and quizzes function correctly, displaying accurate results and feedback.
- The SQL compiler executes DDL, DML, and query statements correctly.

- The chatbot successfully generates SQL statements based on user input, both in English and Arabic, and provides appropriate execution options where applicable.
- The community interaction features (posts, comments, and likes) operate smoothly.
- Error handling and system responses are appropriately displayed when users attempt unsupported actions.

Overall, all test cases passed, confirming that the system meets the expected requirements.

Chapter 7: Results & Discussion

We were able to build this system with the features mentioned above and the non-functional requirements were met.

In this chapter, we present the results and discuss the performance of the SQL learning platform, focusing on the algorithms and solutions implemented in the system. The platform leverages advanced techniques such as Natural Language Processing (NLP), Knowledge-Based Systems, and Cosine Similarity to provide an interactive and effective learning experience. Below, we discuss the outcomes of each key feature and its impact on the overall system.

7.1 Chatbot Query Generation

The Chatbot Query Generation feature utilizes a pre-trained NLP model (LoRA Fine-tuned Gemma 2B) to convert natural language queries into SQL commands. This feature was tested with a variety of user inputs, ranging from simple SELECT statements to complex JOIN operations.

Results:

The model achieved a SparseCategoricalAccuracy of 0.6515 and a loss of 0.2465 in generating correct SQL queries

The system successfully extracted table context and relevant columns, ensuring that the generated queries were syntactically correct.

Discussion:

While the model performs well for basic queries, there is room for improvement in handling complex and ambiguous inputs. Future work could involve fine-tuning the model with a larger dataset of Text-to-SQL examples and incorporating user feedback to enhance accuracy.

7.2 Chatbot Syntax

The Chatbot Syntax feature acts as an interactive SQL compiler, guiding users through the process of constructing SQL queries using a Knowledge-Based System.

Results:

- Users can create valid SQL queries for full DML and DDL commands.
- The system effectively handled edge cases, such as missing or incorrect inputs, by prompting users for clarification.
- The interactive dialogue system reduced the learning curve for beginners, as they could construct queries step-by-step without prior knowledge of SQL syntax.

Discussion:
This feature proved to be highly effective for novice users, providing a structured approach to learning SQL. However, advanced users found the step-by-step process time-consuming. Future iterations could include an "advanced mode" for experienced users, allowing them to skip intermediate steps.

7.3 Score Calculator
The Score Calculator uses Cosine Similarity to evaluate the accuracy of user answers by comparing them with the correct answers.

Results:

- The system relies on Cosine Similarity to compare SQL queries, allowing for the evaluation of answers based on their similarity to the correct response.
- The model demonstrated moderate to good accuracy in distinguishing between different answers, but it struggles with recognizing semantically equivalent but syntactically different queries.
- The similarity score is computed using the all-mpnet-base-v2 model, which converts text into numerical embeddings for comparison.
- Cosine Similarity results indicate a clear difference between distinct queries, but they do not always directly reflect the correctness of an answer. This suggests the need for additional processing to classify answers as fully correct, partially correct, or incorrect.
- The system can be improved by implementing flexible similarity thresholds, where answers are categorized based on their computed similarity score.

Discussion:
The Score Calculator is a critical component of the platform, ensuring fair and accurate evaluation of user performance. However, the system occasionally struggled with

semantically similar but syntactically different answers. Enhancing the model with semantic understanding capabilities could address this limitation.

7.4 User SQL Compiler

The User SQL Compiler provides a personalized environment for users to practice SQL commands in their own database.

Results:

Users were able to create, execute, and save SQL commands with 100% accuracy.

The integration of the user's tables with the NLP Chatbot and Chatbot Syntax features enhanced the overall learning experience.

The dynamically generated ERD diagrams using the Mermaid library were well-received, as they provided a clear visual representation of the database schema.

Discussion:

The User SQL Compiler is a standout feature, offering users a safe and isolated environment to practice SQL. The integration with other features ensures a seamless learning experience. Future enhancements could include support for additional database management systems (e.g., MySQL, PostgreSQL) to broaden the platform's applicability.

7.5 Quiz Generation

The Quiz Generation feature dynamically creates quizzes based on user performance, ensuring a personalized learning experience.

Results:

The system successfully generated quizzes tailored to each user's strengths and weaknesses.

The mathematical formula used to calculate the number of questions per level provided a balanced distribution of question types.

Users reported that the quizzes were challenging yet fair, helping them reinforce their understanding of SQL concepts.

Discussion:

The adaptive nature of the quiz generation system ensures that users are consistently challenged at an appropriate level. However, the formula's weights ($w1, w2, w3, w4$) may need adjustment based on user feedback to better reflect individual learning patterns.

7.6 Non-Functional Requirements

- Performance

Optimized database queries and implemented caching to handle 100+ concurrent users efficiently.

- Scalability

Used a microservices architecture and cloud-based hosting for seamless scaling.

- Usability

Designed an intuitive UI with clear navigation and interactive elements for all skill levels.

- Security

Implemented input validation, hashed passwords with bcrypt, and separated databases for isolation.

- Reliability

Deployed on a reliable cloud provider with redundancy and automated monitoring for 99.9% uptime.

- Maintainability

Followed modular coding practices with clear documentation for easy updates.

- Compatibility

Ensured cross-browser and OS compatibility through responsive design and extensive testing.

Conclusion and Future Works

Conclusion

The system provides a comprehensive and user-friendly environment for managing databases, executing SQL queries, and facilitating SQL learning through quizzes and exercises. With features like user authentication, CRUD operations, data management, and interactive chatbot assistance, the system aims to enhance the overall database management experience. Additionally, the system supports automatic SQL code generation, which simplifies the process of creating complex queries, thus increasing productivity and efficiency. By integrating these features, the system ensures that users can efficiently manage their databases and improve their SQL proficiency.

As the demand for more advanced and tailored database management solutions grows, future developments will focus on incorporating AI-driven capabilities. Features such as converting ERD to code, auto-completion for SQL queries, and support for NoSQL databases will further enhance the system's versatility and adaptability to different user needs. These improvements will provide a more dynamic and efficient platform, enabling users to manage databases with ease while keeping up with evolving technologies. Through these advancements, the system aims to continuously meet the growing demands of database management and education.

Future Works

To enhance SQL Mentor's capabilities and user experience, we plan to implement the following advanced features:

1. ERD-to-Code Generator

Description:
A tool to automatically convert Entity-Relationship Diagrams (ERDs) into SQL code, streamlining database design and schema creation.
Key Features:

- Drag-and-drop interface for designing ERDs.
- Auto-generation of SQL DDL (Data Definition Language) statements (e.g., CREATE TABLE, FOREIGN KEY).
- Support for reverse-engineering existing databases into visual ERDs.
  Benefits:
- Simplifies learning database design principles.
- Reduces manual coding errors during schema creation.
- Integrates with the Database Editor for real-time code testing.

2. Intelligent SQL Auto-Complete

Description:
A context-aware code completion system to assist users while writing SQL queries.
Key Features:

- Real-time syntax suggestions based on database schema (tables, columns, constraints).
- Predictive text for SQL keywords (e.g., SELECT, JOIN, GROUP BY).
- Error highlighting and corrective recommendations.
  Benefits:
- Accelerates query writing for beginners and experts.
- Minimizes syntax errors during exercises.
- Adaptive learning: Suggests relevant functions/clauses based on user progress.

3. PL/SQL Support

Description:
Expanding beyond basic SQL to include Procedural Language/SQL (PL/SQL) for advanced database programming.
Key Features:

- Interactive tutorials on writing stored procedures, functions, and triggers.
- Code validation for PL/SQL logic (e.g., loops, conditional statements).
- Sandbox environment to test PL/SQL scripts against user-created databases.
  Benefits:
- Prepares users for real-world database administration and automation tasks.
- Enables complex data operations (e.g., transaction handling, error logging).
- Integrates with the Chatbot to guide users through PL/SQL debugging.

Appendix 1: Method Specification

Appendices provide supplementary information that, if included in the main text, may cause distraction and cloud the central theme. Appendices should be numbered using Arabic numerals, e.g., Appendix 1, Appendix 2.

1. Overview

This appendix outlines the methodology used in the development and evaluation of the SQL Mentor system. It describes the techniques, models, and frameworks employed to enhance SQL learning through interactive exercises, quizzes, and AI-driven assessments.

2. Sentence Similarity Calculation

The system utilizes Cosine Similarity to compare user responses with correct answers. The SentenceTransformer Model (based on all-mpnet-base-v2) is used to generate numerical embeddings of sentences. The similarity between two vectors is calculated using:

Where:

- is the dot product of vectors A and B.
- and are the magnitudes of vectors A and B.

The resulting similarity score ranges from 0 to 1, indicating the degree of similarity between user responses and correct answers.

3. Quiz Generation Algorithm

A dynamic quiz generation algorithm determines the number and difficulty of questions based on user performance. The parameters considered include:

- Time taken to solve exercises
- Number of attempts made
- Usage of the 'Show Answer' feature
- Accuracy of previous answers (calculated using the Score Calculator)

The formula applied is:

where the weight values are:

- (accuracy weight)
- (time weight)
- (attempts weight)
- (show answer penalty weight)

The final question count is capped between 0 and 8 per level.

4. SQL Compiler Implementation

The system provides a personal SQL compiler where users can:

- Execute DML (SELECT, INSERT, UPDATE, DELETE) and DDL (CREATE, ALTER, DROP) commands.
- Store and retrieve table structures.
- Integrate their database schema into the AI Chatbot for query generation.

To visualize database relationships, an Entity-Relationship Diagram (ERD) is dynamically generated using the Mermaid library.

5. NLP-Driven Chatbot for SQL Query Generation

The chatbot leverages LoRA Fine-tuned Gemma 2B for Text-to-SQL conversion. It interacts with users through structured queries and dynamically generates optimized SQL statements by:

1. Extracting table context from the user database.
2. Parsing user input for intent and required fields.
3. Constructing valid SQL queries with semantic and syntactic correctness.
4. Providing execution options where applicable.

6. Performance Metrics & Evaluation

The effectiveness of the system is assessed using the following metrics:

- Quiz Score Accuracy: Measures how well the Score Calculator aligns with human evaluation.
- Execution Time: Assesses the speed of query execution in the SQL compiler.
- User Engagement: Evaluates time spent on exercises and number of attempts before success.
- Chatbot Precision: Determines the correctness of SQL queries generated by the chatbot.

7. Future Improvements

Planned enhancements to the methodology include:

- Integration with PL/SQL to support stored procedures and triggers.
- Support for multiple database engines beyond SQLite (e.g., MySQL, PostgreSQL).
- Improved error analysis algorithms for better SQL syntax suggestions.

This methodology ensures an efficient, adaptive, and user-friendly learning environment for mastering SQL.

References

Khadija, M., & Mustapha, M. (2024). CHAT-SQL: NATURAL LANGUAGE TEXT TO SQL QUERIES BASED ON DEEP LEARNING TECHNIQUES . *Journal of Theoretical and Applied Information Technology*, 10.

Gonzales, M. E., Ong, E. B., Cheng, C. K., Ong, E. C., & Azcarraga, J. J. (2023). From Unstructured to Structured: Transforming Chatbot Dialogues into Data Mart Schema for Visualization. *Philippine Computing Science Congress (PCSC2023)*, 1-8.

Song, Y., Wong, R. C.-W., Zhao, X., & Jiang, D. (2024). Speech-to-SQL: Towards Speech-driven SQL Query Generation From Natural Language Question. *The VLDB Journal*, 1-14.

Dinu, R. M., Mihăescu, M. C., & Rebedea, T. E. (2023). Machine Learning System for Natural Language to SQL Translation. *Proceedings of RoCHI 2023*, (pp. 1-8).

Wong, A., Pham, L., Lee, Y., Chan, S., Sadaya, R., Khmelevsky, Y., . . . Ferri, M. (2024). Translating Natural Language Queries to SQL Using the T5 Model. *2024 IEEE International Systems Conference (SysCon)* (pp. 1-6). Montreal: IEEE.

HackerRank. (n.d.). *Solve SQL | HackerRank*. Retrieved from HackerRank: https://www.hackerrank.com/domains/sql

Coddy.Tech. (n.d.). *Coddy - Learn SQL*. Retrieved from Coddy.Tech: https://coddy.tech/landing/sql

SQLTutorial. (n.d.). *SQL Tutorial: Learn SQL from Scratch for Begginers*. Retrieved from SQLTutorial: https://www.sqltutorial.org/

SQLZoo. (n.d.). *SQLZoo* . Retrieved from SQLZoo : https://sqlzoo.net/wiki/SQL_Tutorial

W3Schools. (n.d.). *SQL Tutorial*. Retrieved from W3Schools: https://www.w3schools.com/sql/

ملخص

النظام يوفر بيئة شاملة لإدارة قواعد البيانات وتنفيذ استعلاماتSQL ، مما يعزز من كفاءة الاستخدام ويسهم في تحسين مهارات المستخدمين في التعامل مع قواعد البيانات. يتضمن النظام ميزات متعددة مثل تسجيل الدخول للمستخدم، الذي يسمح للمستخدمين بإنشاء حسابات جديدة وتسجيل الدخول باستخدام البريد الإلكتروني وكلمة المرور، مما يوفر تجربة آمنة وموثوقة.

تم تصميم النظام باستخدام Next.js مع Python ، مما يضمن أداءً عاليًا وسرعة في تنفيذ العمليات المختلفة. يدعم النظام العمليات الأساسية CRUD ، مما يمكن المستخدمين من إنشاء وحذف وتحديث البيانات والاطلاع عليها بفعالية. كما يوفر إدارة شاملة لقواعد البيانات، حيث يمكن للمستخدمين إنشاء قواعد بيانات جديدة وإجراء التعديلات عليها وحذفها عند الحاجة، مما يتيح إدارة مرنة وفعالة للبيانات.

إحدى الميزات الرئيسية هي القدرة على تنفيذ استعلامات SQL بدقة وسرعة، مع تقديم دعم متقدم لمعالجة الأخطاء وعرض النتائج بشكل واضح وسهل الفهم. المساعد الذكي يوفر وظيفة لإنشاء استعلامات SQL تلقائيًا بناءً على المدخلات النصية من المستخدمين، مما يجعل العملية أكثر سلاسة ويساعد في تحسين إنتاجية العمل مع الاستعلامات.

علاوة على ذلك، يتم تشفير كلمات المرور في قواعد البيانات لضمان الأمان وحماية بيانات المستخدمين من التهديدات الخارجية.

باختصار، يقدم النظام تجربة متكاملة وآمنة لإدارة قواعد البيانات، مع التركيز على تحسين تجربة المستخدم والأمان، مما يسهم في تعزيز الإنتاجية وسهولة استخدام النظام بشكل عام.

الجامعة العربية الدولية

الجامعة العربية الدولية AIU جامعة سورية خاصة أُحدثت عام 2005، خططها الدراسية والوثائق الصادرة عنها معتمدة ومصدقة من قبل وزارة التعليم العالي في الجمهورية العربية السورية.

تعمل الجامعة على تحقيق الأهداف الآتية:

- إعداد جيل متميز من الخريجين الجامعيين القادرين على تلبية الحاجات النوعية للمجتمع والنهوض به.
- الإسهام في البحوث العلمية النظرية والتطبيقية التي تخدم أغراض التنمية الوطنية، ويتم العمل على حث الأساتذة والعاملين الأكاديميين على البحث العلمي والمشاركة في المؤتمرات والندوات التي تنظم الأبحاث.
- تحقيق الشراكة مع الجامعات العربية والأجنبية المرموقة بهدف التطوير والتحديث المستمرين للعمل الأكاديمي والقيام ببحوث علمية مشتركة.
- استقطاب الكفاءات الأكاديمية والبحثية المتميزة عن طريق توفير البيئة المناسبة لعملها.

الجامعة العربية الدولية من الجامعات السورية الأولى التي جرى تأسيسها وافتتاحها، وقد تمكنت من اجتذاب الكفاءات التعليمية والبحثية والإدارية المتميزة، لإنشاء صرح متكامل من النواحي الأكاديمية والتنظيمية والإدارية. وتمكنت من تخريج كوادر من المبدعين والمتميزين من خلال توفير بيئة تعليمية ترتكز إلى مقومات نوعية ومادية فريدة منها:

- الخطط الدراسية الحديثة والمتطورة المستندة إلى نظام الساعات المعتمدة.
- الأطر التعليمية المنتقاة بعناية كبيرة.
- المختبرات العلمية الحديثة، ومختبر للمكتبات الإلكترونية.
- المحفزات المادية والمعنوية للطلبة.
- تطبيق طرائق التدريس التفاعلي.
- التوجيه والإرشاد الأكاديمي والتربوي.
- مجموعة كبيرة من اتفاقيات التعاون العلمي مع جامعات محلية وإقليمية ودولية ذات سمعة مرموقة.
- اتفاقيات ومذكرات تفاهم متعددة مع العديد من مؤسسات المجتمع المدني.

- الحرم الجامعي اللائق والمزود بكافة المرافق العلمية والرياضية والترفيهية، والذي نشجعك على زيارته والتعرف على مزاياه.
- الأنشطة والأندية الطلابية بمختلف أنواعها: الرياضية والثقافية والعلمية والاجتماعية.

في الجامعة العربية الدولية سنوات الحياة الجامعية هي وقت للاستثمار في مستقبل الطالب. فالمعارف والخبرات التي يحصلها في قاعة المحاضرات والمختبرات ستساعده في تطوير ذاته، وستمنحه أسباب النجاح في التخصص الذي اختاره، والنشاط الطلابي الذي يمارسه سيساعده في توسيع أفقه، وفعاليات التدريب والأندية والرياضة ستمكنه من تطوير مواهبه، ولربما تساعده في اكتشاف مواهب جديدة.

ليستثمر وقته وذهنه وروحه في جامعتنا كي يجني فوائد عمله والوقت الذي كرسه في السنين القادمة. ونحن سوف نكون بجانب طلبتنا في كل خطوة على دربهم.

الجامعة العربية الدولية

كلية الهندسة المعلوماتية والاتصالات

مشروع ما قبل التخرج

مرشد SQL

تم تقديمه إلى

قسم الهندسة المعلوماتية

تقديم

إياد حسن برغوث **أحمد رفعت قيطاز**

محمد زيد النحاس

بإشراف

المهندسة نسرين الماضي

شباط 2025