**CODING PLATFORM MANAGEMENT**

A MINI PROJECT REPORT

Submitted By

**LOKESH SINGH .N - (230701166)**

**KUMARAN .S - (230701159)**

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI -602105

2023-2024

**BONAFIDE CERTIFICATE**

Certified that this project report “**CODING PLATFORM MANAGEMENT**”

Is the Bonafide work of

**“LOKESH SINGH . N - (230701166) & KUMARAN .S - (230701159)**

Who carried out the project work under my supervision

Submitted for practical examination held on

INTERNAL EXAMINER EXTERNAL EXAMINER

**ACKNOWLEDGEMENT**

I would like to extend my sincere gratitude to everyone who has contributed to the successful completion of this mini project.

First and foremost, I am deeply thankful to my Professor

**Mrs. K .Maheshmeena** my project advisor, for their invaluable guidance, insightful feedback, and continuous support throughout the duration of this mini project,. Their expertise and encouragement have been instrumental in shaping my research and bringing this mini project to completion.

I would also like to express my appreciation to the faculty and staff of the **Computer Science and Engineering Department** at Rajalakshmi engineering college for providing the necessary resources and a conductive learning environment. We express our sincere thanks to **Dr. P. Kumar, M.E., Ph. D**., Professor and Head of the Department Computer Science and Engineering for his guidance and encouragement throughout the project work.

My heartfelt thanks go to my peers and friends for their collaboration, constructive criticism, and moral support.

Thank you all for your contributions, both direct and indirect, to the success of this project.

**Abstract:**

In the digital age, coding platforms serve as essential hubs for software development, education, and collaboration among programmers of varying skill levels. Effective management of these platforms is crucial for fostering an environment that enhances user engagement, facilitates learning, and supports project development. This paper explores the key elements of coding platform management, including user experience optimization, content curation, community building, and performance analytics. We analyze strategies for maintaining platform security and integrity, ensuring that users can collaborate safely and effectively. Additionally, we discuss the role of feedback mechanisms in iterating and improving platform features. By examining case studies from leading coding platforms, we provide insights into best practices for managing coding environments that cater to diverse user needs, ultimately contributing to the advancement of software development and the programming community. Our findings reveal that a robust management approach not only enhances platform functionality but also fosters creativity, knowledge sharing, and innovation among users.

Table of contents

**1.INTRODUCTION**

1.1 INTRODUCTION

1.2 OBJECTIVES

1.3 MODULES

**2. SURVEY OF TECHNOLOGIES**

2.1 SOFTWARE DESCRIPTION

2.2 LANGUAGES

2.2.1 SQL

2.2.2 JAVA

**3.REQUIREMENTS AND ANALYSIS**

3.1 REQUIREMENT SPECIFICATION

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

3.3 ARCHITECTURE DIAGRAM

3.4 ER DIAGRAM

3.5 NORMALIZATION

**4.PROGRAM CODE**

**5.RESULTS AND DISCUSSION**

**6.CONCLUSION**

**1.INTRODUCTION**

**1.1 INTRODUCTION**

**Coding Platform Management** refers to the comprehensive administration and optimization of integrated development environments (IDEs) that support the software development lifecycle. With the increasing complexity of modern software projects and the rise of distributed development teams, the need for effective management of coding platforms has become crucial. These platforms provide a centralized environment for writing, testing, and deploying code, facilitating collaboration, version control, and automated workflows. Effective platform management ensures that developers have access to the right tools, resources, and performance optimizations, while maintaining secure and scalable infrastructure. As development practices evolve, including the adoption of DevOps and cloud-based architectures, coding platform management has shifted from simple tool integration to a strategic approach that encompasses security, performance monitoring, continuous integration, and dynamic resource allocation. The goal is to create a seamless, efficient, and secure environment that accelerates development cycles and enhances team productivity.

**1.2 OBJECTIVES**

* The objective of **Coding Platform Management** is to provide a secure, efficient, and scalable environment that supports the entire software development lifecycle. This includes enabling seamless collaboration among developers through integrated tools for version control, real-time code sharing, and communication.
* The platform aims to optimize performance by continuously monitoring system resources, uptime, and performance metrics, while ensuring secure code handling through robust access controls and encryption.
* It also focuses on automating development workflows with CI/CD pipelines to streamline code testing, building, and deployment.
* Additionally, the platform aims to maintain high code quality by incorporating automated testing, code analysis, and reviews, while allowing for dynamic resource allocation and scaling to meet varying workload demands.
* Compliance with industry standards and regulatory requirements is also a priority, alongside providing comprehensive documentation management to support efficient project development and knowledge sharing.
* Ultimately, the goal is to enhance productivity, security, and collaboration across development teams.

**1.3 MODULES**

* **User Management Module**: Handles authentication, role-based access control, and user permissions to ensure secure platform access.

**Version Control Integration Module**: Integrates with systems like Git to manage code repositories, branching, and collaboration.

* **CI/CD Module**: Automates code testing, building, and deployment processes to streamline the development pipeline.
* **Performance Monitoring Module**: Tracks platform performance metrics, such as CPU, memory usage, and system uptime.

**Security Management Module**: Implements encryption, authentication protocols, and vulnerability assessments for secure code handling.

* **Collaboration Tools Module**: Enables real-time code sharing, chat, and issue tracking for team collaboration.

**Resource Allocation & Scaling Module**: Manages platform resources and supports dynamic scaling for cloud-based platforms.

* **Code Quality & Testing Module**: Automates static analysis, unit testing, and quality checks to ensure high-quality code.
* **Audit & Compliance Module**: Tracks system changes and ensures compliance with industry standards and regulations.
* **Documentation Management Module**: Facilitates the creation, management, and sharing of project documentation for easy access

**2. SURVEY OF TECHNOLOGIES**

**2.1 SOFTWARE DESCRIPTION**

Software Description for Coding Platform Management refers to the suite of tools and systems designed to facilitate the development, collaboration, and deployment of software projects within a centralized environment. This software integrates a variety of essential features such as project management, version control, code execution, testing, and deployment, streamlining the entire software development lifecycle (SDLC). At its core, it offers an integrated development environment (IDE) that supports multiple programming languages and provides features like syntax highlighting, code completion, debugging tools, and version control system integration, typically with Git. These platforms enable seamless collaboration among developers by allowing them to share code, track changes, and manage multiple versions of a project efficiently.

A key aspect of coding platform management is the automation of CI/CD pipelines, ensuring that code is continuously integrated, tested, and deployed across various environments. This helps in reducing manual errors, speeding up development cycles, and ensuring consistency in deployments. The platform typically integrates with Docker for containerization and Kubernetes for orchestration, making it easier to manage scalable applications and ensure consistent environments across development, staging, and production.

Security is a critical component, and platform management software often includes features like role-based access control (RBAC), multi-factor authentication (MFA), and encryption for sensitive data to ensure that only authorized users can access or modify critical components. Additionally, performance monitoring tools are embedded to help developers track the health and performance of their applications and infrastructure. In cloud-based setups, coding platform management software can dynamically allocate resources, enabling scalable infrastructure that adapts to varying workloads. Overall, this software serves as a comprehensive solution for improving collaboration, optimizing development workflows, ensuring security, and automating key processes, making it an indispensable tool for modern software development teams.

**2.2 LANGUAGES**

2.2.1 SQL

SQL for Coding Platform Management is essential for handling the storage, retrieval, and manipulation of various types of data within a coding platform. It facilitates the management of key platform components such as user accounts, project repositories, permissions, and activity logs. SQL databases are often used to store and organize structured data, including user profiles, where tables contain information such as usernames, email addresses, roles (admin, developer, etc.), and authentication credentials. SQL queries are used to manage user access, ensuring secure logins and role-based access control to different features within the platform.

In addition to user management, SQL is heavily involved in managing project repositories and associated metadata. Each project may have tables that store information about the project's name, description, contributors, repository links, and code versioning data. With SQL, version control systems like Git are integrated, allowing for tracking and querying project changes, commits, and branches. For team collaboration, SQL is used to manage tables related to comments, issues, tasks, and discussions within the project.

SQL also plays a critical role in managing Continuous Integration/Continuous Deployment (CI/CD) pipelines. It stores results from automated tests, build logs, and deployment histories, enabling platform administrators to track the status of various projects and environments. Performance optimization is essential, particularly for platforms with large user bases, requiring efficient indexing and query optimization to ensure fast data retrieval without affecting system performance. SQL also provides essential tools for data backup, recovery, and audit logging, ensuring data integrity and security in the platform’s database. Overall, SQL is fundamental to the organization, scalability, and functionality of coding platform management, supporting everything from user and project data management to automation and performance monitoring

**2.2.2 JAVA**

Java for Coding Platform Management plays a crucial role in building and managing the various components of a coding platform, providing a robust, scalable, and secure backend infrastructure. Java is widely used in the development of platform features such as user authentication, project management, code versioning, and the integration of Continuous Integration/Continuous Deployment (CI/CD) pipelines. Through Java-based frameworks like Spring Boot and Spring MVC, developers can build RESTful APIs that enable interaction between different modules of the platform, ensuring smooth communication between the frontend, backend, and third-party services. Java is also used to implement business logic for user management, permission control, and role-based access, which are essential for ensuring secure and seamless collaboration among development teams.

For the management of code repositories, Java can integrate with version control systems like Git, allowing the platform to track code commits, branches, merges, and changes across multiple repositories. Additionally, Java can handle the orchestration of build and deployment processes, supporting integration with tools like Jenkins or GitLab CI to automate testing, building, and deploying code. Java’s strong support for multi-threading and concurrent processing allows the platform to efficiently handle multiple user requests and large-scale code executions, ensuring a smooth experience even under heavy load.

Java-based technologies like Hibernate or JPA (Java Persistence API) are often employed for database management within the coding platform, enabling smooth interaction with relational databases that store user data, project metadata, and logs. Furthermore, Java's integration with containerization tools like Docker allows for the creation of isolated environments for running applications, ensuring consistency across different development environments. With Java’s rich ecosystem of libraries, frameworks, and tools, it remains an ideal choice for building and managing the infrastructure and backend services of a modern, scalable coding platform

**3. REQUIREMENTS AND ANALYSIS**

**1. REQUIREMENT SPECIFICATION**

Functional Requirements:

* The **functional requirements** for a **Coding Platform Management** system include user authentication and role-based access control to manage permissions.
* The platform should support project and repository management with version control integration, enabling teams to track code changes. It must facilitate Continuous Integration/Continuous Deployment (CI/CD) for automated testing and deployment.
* Collaboration features like task tracking, code reviews, and real-time communication are essential for team coordination.
* Additionally, the platform should ensure security, offer cloud integration for deployment, and provide analytics and reporting tools for performance monitoring.

Non Functional Requirements:

* **Scalability**: The platform must be able to scale horizontally and vertically to handle increasing numbers of users, projects, and code repositories without significant performance degradation.
* **Availability and Reliability**: The system should have high availability, ensuring minimal downtime with redundancy and failover mechanisms to provide continuous service even in the event of failures.
* **Security**: The platform must ensure data protection through encryption, secure user authentication, role-based access control, and compliance with industry-standard security protocols to safeguard user and project data.
* **Performance**: The platform must deliver fast response times for tasks such as code execution, testing, and deployment, even under heavy load, ensuring a smooth and efficient user experience.
* **Usability**: The system should be intuitive and user-friendly, with a well-designed interface that allows users to easily navigate, manage projects, and collaborate effectively, reducing the learning curve for new users.

**3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

**Hardware Requirements:**

**Server and Processor**:  
A multi-core processor (e.g., Intel Xeon or AMD EPYC) with at least 4 cores and a clock speed of 2.5 GHz or higher, along with a minimum of 16 GB RAM (32 GB recommended for larger teams), to handle concurrent code execution, builds, version control, and user management tasks.

**Storage**:  
High-speed SSD storage with a minimum of 500 GB capacity for storing code repositories, project data, version history, and logs. Scalable storage options should be available for growing data, with backup solutions in place to ensure data integrity.

**Network and Redundancy**:  
High-speed network connectivity (1 Gbps or higher) for smooth collaboration and remote access. Redundant power systems (UPS) and backup servers should be in place to ensure continuous platform availability and disaster recovery.

**Software Requirements:**

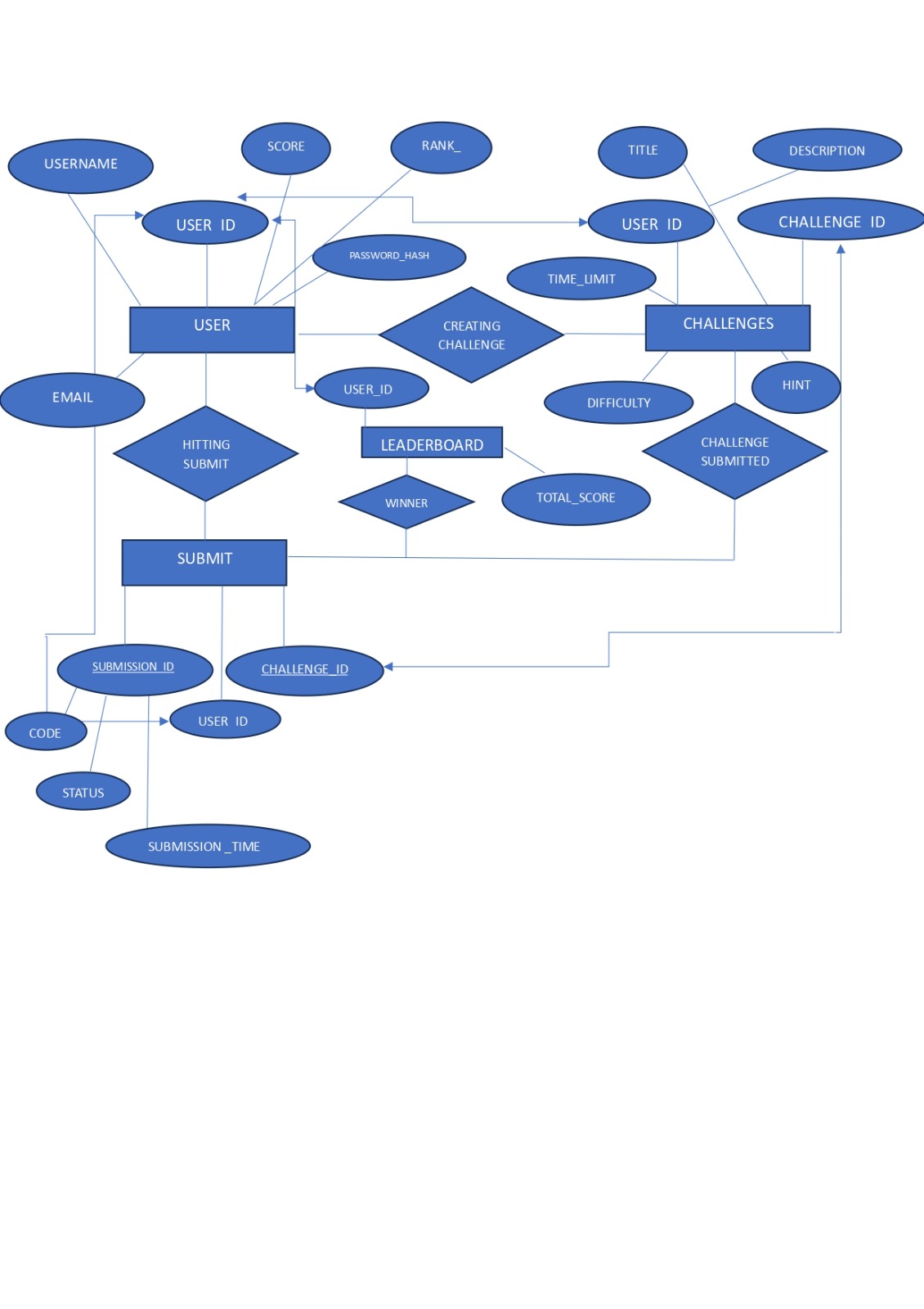
**Operating System**:

A stable, server-grade operating system such as **Linux (Ubuntu, CentOS, or Red Hat)** or **Windows Server**, depending on the platform’s architecture and development stack, to ensure compatibility with development tools and backend services.

**Version Control System**:  
A robust **Version Control System (VCS)** like **Git**, integrated with platforms such as **GitHub**, **GitLab**, or **Bitbucket**, to manage code repositories, versioning, branching, and collaboration among developers.

**Development Tools and Frameworks**:  
Support for integrated development environments (IDEs)

**3.4 ER DIAGRAM**

****

**3.5 NORMALIZATION**

Normalization for the coding platform's database schema ensures data efficiency, integrity, and maintainability.

1. First Normal Form (1NF): Eliminate Repeating Groups

- Each table must have a primary key, and every field should contain atomic (indivisible) values.

Example: Potential Issues in Raw Data

Submissions:

| submission\_id | user\_id | challenge\_id | code | status | submission\_time |

|---------------|---------|--------------|---------------|----------|-----------------|

| 1 | 10 | 5 | print("Hello")| Passed | 2024-11-20 |

| 2 | 10 | 5 | print("Hello")| Passed | 2024-11-20 |

Here, duplicate submissions exist. Submissions should be uniquely identified.

Normalized Table:

- Add a unique identifier (submission\_id) for each submission.

Normalized Submissions Table:

| submission\_id | user\_id | challenge\_id | code | status | submission\_time |

|---------------|---------|--------------|---------------|----------|-----------------|

| 1 | 10 | 5 | print("Hello")| Passed | 2024-11-20 |

Similarly, apply the same principle to other tables.

2. Second Normal Form (2NF): Remove Partial Dependencies

-Non-key attributes should depend on the whole primary key, not just part of it.

Submissions Table

Problem: user\_id, challenge\_id, and status are partially dependent on submission\_id.

Solution: Keep submission details in the Submissions table and create separate tables for Users and Challenges to hold their details. Use foreign keys to link these tables.

Normalized Tables:

1. Submissions Table:

| submission\_id | user\_id | challenge\_id | code | status | submission\_time |

|---------------|---------|--------------|---------------|----------|-----------------|

| 1 | 10 | 5 | print("Hello")| Passed | 2024-11-20 |

2. Users Table:

| user\_id | username | password\_hash | email | score | rank |

|---------|----------|---------------|---------------|-------|------|

| 10 | Alice | [hashed\_pass] | alice@mail.com| 1500 | 2 |

3. Challenges Table:

| challenge\_id | title | description | difficulty | time\_limit | hint |

|--------------|------------|---------------------|------------|------------|--------------|

| 5 | "Basics" | "Intro to Python..."| Easy | 2s | "Use print()"|

3. Third Normal Form (3NF): Remove Transitive Dependencies

- Non-key attributes should not depend on other non-key attributes.

Users Table

Problem: score depends on user\_id, but rank is derived from score.

Solution: Remove rank from the Users table. Instead, compute rank dynamically or store it in a separate Leaderboard table.

Normalized Tables:

1. Users Table:

| user\_id | username | password\_hash | email | score |

|---------|----------|---------------|---------------|-------|

| 10 | Alice | [hashed\_pass] | alice@mail.com| 1500 |

2. Leaderboard Table:

| user\_id | total\_score |

|---------|-------------|

| 10 | 1500 |

Challenges Table

No transitive dependency issues here since all attributes directly describe the challenge\_id.

Final Normalized Schema

After normalization, the schema ensures data integrity and avoids redundancy:

1. Submissions Table:

| submission\_id | user\_id | challenge\_id | code | status | submission\_time |

|---------------|---------|--------------|---------------|----------|-----------------|

2. Users Table:

| user\_id | username | password\_hash | email | score |

|---------|----------|---------------|---------------|-------|

3. Leaderboard Table:

| user\_id | total\_score |

|---------|-------------|

4. Challenges Table:

| challenge\_id | title | description | difficulty | time\_limit | hint |

|--------------|------------|---------------------|------------|------------|--------------|

**Benefits of Normalization for the Coding Platform**

1. Data Integrity: Eliminates duplicate or inconsistent data (e.g., user scores are updated consistently across tables).

2. Reduced Redundancy: Saves storage space and minimizes repetitive data (e.g., challenge details are stored once).

3. Simplified Queries: Improves the clarity of SQL queries by separating concerns into dedicated tables.

4. Scalability: Makes it easier to add features like tags for challenges or a submission review system without altering existing tables.

1. **PROGRAM CODE:**

**SAMPLE CODE**

* 1. **MAIN PAGE DESIGN**

import javax.swing.\*;

public class Main {

public static void main(String[] args) {

try {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());

} catch (Exception e) { e.printStackTrace();

}

SwingUtilities.invokeLater(() -> {

LoginFrame loginFrame = new LoginFrame(); loginFrame.setVisible(true);

});

}

}

**4.2 REGISTRATION PAGE DESIGN**

**import** javax.swing.\*;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**public class** RegistrationFrame **extends** JFrame {

**private** JTextField usernameField;

**private** JTextField emailField;

**private** JPasswordField passwordField;

**private** JButton registerButton;

**public** RegistrationFrame() { setTitle("Register"); setSize(400, 350);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***); setLocationRelativeTo(**null**);

JPanel panel = **new** JPanel(); panel.setLayout(**null**);

JLabel usernameLabel = **new** JLabel("Username:"); usernameLabel.setBounds(50, 50, 100, 30); panel.add(usernameLabel);

usernameField = **new** JTextField(); usernameField.setBounds(150, 50, 150, 30); panel.add(usernameField);

JLabel emailLabel = **new** JLabel("Email:")

emailLabel.setBounds(50, 100, 100, 30);

panel.add(emailLabel);

emailField = **new** JTextField(); emailField.setBounds(150, 100, 150, 30); panel.add(emailField);

JLabel passwordLabel = **new** JLabel("Password:"); passwordLabel.setBounds(50, 150, 100, 30); panel.add(passwordLabel);

passwordField = **new** JPasswordField(); passwordField.setBounds(150, 150, 150, 30); panel.add(passwordField);

registerButton = **new** JButton("Register"); registerButton.setBounds(150, 200, 100, 30); panel.add(registerButton);

registerButton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) { String username = usernameField.getText(); String email = emailField.getText();

String password = **new** String(passwordField.getPassword()); Database.*registerUser*(username, password, email); JOptionPane.*showMessageDialog*(**null**, "Registration successful!"); **new** LoginFrame().setVisible(**true**);

dispose();

}

});

add(panel);

}

**public static void** main(String[] args) { SwingUtilities.*invokeLater*(() -> {

**new** RegistrationFrame().setVisible(**true**);

});

}

}

* 1. **LOGIN PAGE DESIGN:**

**import** javax.swing.\*;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**public class** LoginFrame **extends** JFrame { **private** JTextField usernameField; **private** JPasswordField passwordField; **private** JButton loginButton;

**private** JButton registerButton;

**public** LoginFrame() { setTitle("Login"); setSize(400, 300);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***); setLocationRelativeTo(**null**);

JPanel panel = **new** JPanel(); panel.setLayout(**null**);

JLabel usernameLabel = **new** JLabel("Username:"); usernameLabel.setBounds(50, 50, 100, 30); panel.add(usernameLabel);

usernameField = **new** JTextField(); usernameField.setBounds(150, 50, 150, 30); panel.add(usernameField);

JLabel passwordLabel = **new** JLabel("Password:"); passwordLabel.setBounds(50, 100, 100, 30); panel.add(passwordLabel);

passwordField = **new** JPasswordField(); passwordField.setBounds(150, 100, 150, 30); panel.add(passwordField);

loginButton = **new** JButton("Login"); loginButton.setBounds(50, 150, 100, 30); panel.add(loginButton);

registerButton = **new** JButton("Register"); registerButton.setBounds(200, 150, 100, 30); panel.add(registerButton);

loginButton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) { String username = usernameField.getText();

String password = **new** String(passwordField.getPassword());

**int** userId = Database.*isLoginValid*(username, password);

**if** (userId != -1) {

**new** MainDashboard(username, userId).setVisible(**true**); dispose();

} **else** {

JOptionPane.*showMessageDialog*(**null**, "Invalid credentials, please try again.");

}

}

});

registerButton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) { **new** RegistrationFrame().setVisible(**true**); dispose();

}

});

add(panel);

}

**public static void** main(String[] args) { SwingUtilities.*invokeLater*(() -> {

**new** LoginFrame().setVisible(**true**);

});

}

}

* 1. **DASHBOARD DESIGN**

**import** javax.swing.\*;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**public class** MainDashboard **extends** JFrame {

**private int** userId;

**private** String username;

**public** MainDashboard(String username, **int** userId) {

**this**.username = username;

**this**.userId = userId;

setTitle("Dashboard"); setSize(400, 300);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***); setLocationRelativeTo(**null**);

JPanel panel = **new** JPanel(); panel.setLayout(**null**);

JLabel welcomeLabel = **new** JLabel("Welcome, " + username + "!"); welcomeLabel.setBounds(50, 30, 300, 30); panel.add(welcomeLabel);

JButton challengesButton = **new** JButton("Challenges"); challengesButton.setBounds(100, 80, 150, 30); panel.add(challengesButton);

JButton createChallengeButton = **new** JButton("Create Challenge"); createChallengeButton.setBounds(100, 120, 150, 30); panel.add(createChallengeButton);

JButton leaderboardButton = **new** JButton("Leaderboard"); leaderboardButton.setBounds(100, 160, 150, 30); panel.add(leaderboardButton);

challengesButton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) {

**new** ChallengeFrame(userId).setVisible(**true**);

}

});

createChallengeButton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) {

**new** CreateChallengeFame(userId).setVisible(**true**);

}

});

leaderboardButton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) {

**new** LeaderboardFrame().setVisible(**true**);

}

});

add(panel);

}

}

* 1. **CHALLENGE FRAME DESIGN**

**import** javax.swing.\*;

**import** java.awt.event.ActionEvent; **import** java.awt.event.ActionListener; **import** java.util.List;

**public class** ChallengeFrame **extends** JFrame {

**private int** userId;

**public** ChallengeFrame(**int** userId) {

**this**.userId = userId;

setTitle("Challenges"); setSize(600, 500);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***); setLocationRelativeTo(**null**);

JPanel panel = **new** JPanel(); panel.setLayout(**null**);

JLabel selectChallengeLabel = **new** JLabel("Select a challenge:"); selectChallengeLabel.setBounds(50, 30, 300, 30); panel.add(selectChallengeLabel);

List<String> challenges = Database.*getChallenges*();

**int** yPosition = 80;

**for** (String challenge : challenges) {

JButton challengeButton = **new** JButton(challenge); challengeButton.setBounds(100, yPosition, 300, 50);

yPosition += 60;

challengeButton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) {

**int** challengeId = Database.*getChallengeId*(challenge); JTextArea codeArea = **new** JTextArea(10, 40); codeArea.setLineWrap(**true**); codeArea.setWrapStyleWord(**true**);

JScrollPane scrollPane = **new** JScrollPane(codeArea);

**int** result = JOptionPane.*showConfirmDialog*(

**null**, scrollPane,

"Enter solution code:", JOptionPane.***OK\_CANCEL\_OPTION***, JOptionPane.***PLAIN\_MESSAGE***

);

**if** (result == JOptionPane.***OK\_OPTION***) { String code = codeArea.getText();

Database.*submitSolution*(userId, challengeId, code); JOptionPane.*showMessageDialog*(**null**, "Challenge submitted!");

}

}

});

panel.add(challengeButton);

}

JScrollPane scrollPane = **new** JScrollPane(panel); add(scrollPane);

}

}

* 1. **CREATE CHALLENGE FRAME DESIGN**

**import** javax.swing.\*;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**public class** CreateChallengeFame **extends** JFrame {

**private int** userId;

**public** CreateChallengeFame(**int** userId) {

**this**.userId = userId;

setTitle("Create Challenge"); setSize(400, 300);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***); setLocationRelativeTo(**null**);

JPanel panel = **new** JPanel(); panel.setLayout(**null**);

JLabel titleLabel = **new** JLabel("Challenge Title:"); titleLabel.setBounds(50, 30, 100, 30); panel.add(titleLabel);

JTextField titleField = **new** JTextField(); titleField.setBounds(150, 30, 150, 30); panel.add(titleField);

JLabel descLabel = **new** JLabel("Description:"); descLabel.setBounds(50, 80, 100, 30); panel.add(descLabel);

JTextArea descField = **new** JTextArea(); descField.setBounds(150, 80, 150, 70); panel.add(descField);

JButton createButton = **new** JButton("Create Challenge"); createButton.setBounds(100, 200, 150, 30); panel.add(createButton);

createButton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) { String title = titleField.getText();

String description = descField.getText();

**if** (title.isEmpty() || description.isEmpty()) { JOptionPane.*showMessageDialog*(**null**, "Please fill out all fields.");

} **else** {

Database.*createChallenge*(title, description, userId); JOptionPane.*showMessageDialog*(**null**, "Challenge created!"); dispose();

}

}

});

add(panel);

}

}

* 1. **SUBMISSION FRAME DESIGN**

**import** javax.swing.\*;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**public class** SubmissionFrame **extends** JFrame {

**private int** userId;

**private int** challengeId;

**public** SubmissionFrame(**int** userId, **int** challengeId) {

**this**.userId = userId;

**this**.challengeId = challengeId;

setTitle("Submit Solution"); setSize(400, 300);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***); setLocationRelativeTo(**null**);

JPanel panel = **new** JPanel(); panel.setLayout(**null**);

JLabel codeLabel = **new** JLabel("Enter your solution code:"); codeLabel.setBounds(50, 30, 300, 30); panel.add(codeLabel);

JTextArea codeArea = **new** JTextArea(); codeArea.setBounds(50, 70, 300, 100); panel.add(codeArea);

JButton submitButton = **new** JButton("Submit"); submitButton.setBounds(150, 200, 100, 30); panel.add(submitButton);

submitButton.addActionListener(**new** ActionListener() { @Override

**public void** actionPerformed(ActionEvent e) { String code = codeArea.getText();

Database.*submitSolution*(userId, challengeId, code); JOptionPane.*showMessageDialog*(**null**, "Solution submitted!"

dispose();

}

});

add(panel);

}

}

* 1. **LEADERBOARD FRAME DESIGN**

**import** javax.swing.\*;

**import** javax.swing.table.DefaultTableModel;

**import** java.awt.\*;

**import** java.util.List;

**public class** LeaderboardFrame **extends** JFrame {

**public** LeaderboardFrame() { setTitle("Leaderboard"); setSize(400, 300);

setDefaultCloseOperation(JFrame.***DISPOSE\_ON\_CLOSE***); setLocationRelativeTo(**null**);

JPanel panel = **new** JPanel(); panel.setLayout(**new** BorderLayout());

JLabel titleLabel = **new** JLabel("Leaderboard", JLabel.***CENTER***); titleLabel.setFont(**new** Font("Arial", Font.***BOLD***, 20)); panel.add(titleLabel, BorderLayout.***NORTH***);

String[] columnNames = {"Rank", "Username", "Score"}; DefaultTableModel model = **new** DefaultTableModel(columnNames, 0);

List<UserScore> scores = Database.*getTopScores*();

**int** rank = 1;

**for** (UserScore score : scores) {

model.addRow(**new** Object[]{rank++, score.getUsername(), score.getScore()});

}

JTable table = **new** JTable(model);

panel.add(**new** JScrollPane(table), BorderLayout.***CENTER***);

add(panel);

}

}

* 1. **USERS SCORE**

**public class** UserScore { **private** String username; **private int** score;

**public** UserScore(String username, **int** score) {

**this**.username = username;

**this**.score = score;

}

**public** String getUsername() {

**return** username;

}

**public int** getScore() {

**return** score;

}

}

* 1. **DATABASE CONNECTIVITY**

**import** java.sql.\*;

**import** java.util.ArrayList;

**import** java.util.List;

**public class** Database {

**private static final** String ***URL*** = "jdbc:mysql://localhost:3306/coding\_platform";

**private static final** String ***USER*** = "root";

**private static final** String ***PASSWORD*** = "Sakk29@2006";

**public static** Connection connect() {

**try** {

**return** DriverManager.*getConnection*(***URL***, ***USER***, ***PASSWORD***);

} **catch** (SQLException e) { e.printStackTrace(); **return null**;

}

}

**public static int** isLoginValid(String username, String password) {

String query = "SELECT user\_id FROM users WHERE username = ? AND password\_hash = ?";

**try** (Connection conn = *connect*(); PreparedStatement ps = conn.prepareStatement(query)) { ps.setString(1, username);

ps.setString(2, password); ResultSet rs = ps.executeQuery(); **if** (rs.next()) {

**return** rs.getInt("user\_id");

} **else** {

**return** -1;

}

} **catch** (SQLException e) { e.printStackTrace(); **return** -1;

}

}

**public static void** registerUser(String username, String password, String email) {

String insertQuery = "INSERT INTO users (username, password\_hash, email) VALUES (?, ?, ?)";

**try** (Connection conn = *connect*(); PreparedStatement stmt = conn.prepareStatement(insertQuery)) { stmt.setString(1, username);

stmt.setString(2, password); stmt.setString(3, email); stmt.executeUpdate();

System.***out***.println("User registered successfully.");

} **catch** (SQLException e) { e.printStackTrace();

}

}

**public static void** createChallenge(String title, String description, **int** userId) {

String query = "INSERT INTO challenges (title, description, difficulty) VALUES (?, ?, 'Easy')";

**try** (Connection conn = *connect*(); PreparedStatement ps = conn.prepareStatement(query)) { ps.setString(1, title);

ps.setString(2, description); ps.executeUpdate();

System.***out***.println("Challenge created successfully.");

} **catch** (SQLException e) { e.printStackTrace();

}

}

**public static** List<String> getChallenges() { List<String> challenges = **new** ArrayList<>(); String query = "SELECT title FROM challenges";

**try** (Connection conn = *connect*(); PreparedStatement ps = conn.prepareStatement(query)) { ResultSet rs = ps.executeQuery();

**while** (rs.next()) { challenges.add(rs.getString("title"));

}

} **catch** (SQLException e) { e.printStackTrace();

}

**return** challenges;

}

**public static int** getChallengeId(String challengeTitle) {

String query = "SELECT challenge\_id FROM challenges WHERE title = ?";

**try** (Connection conn = *connect*(); PreparedStatement ps = conn.prepareStatement(query)) { ps.setString(1, challengeTitle);

ResultSet rs = ps.executeQuery();

**if** (rs.next()) {

**return** rs.getInt("challenge\_id");

} **else** {

**return** -1;

}

} **catch** (SQLException e) { e.printStackTrace(); **return** -1;

}

}

**public static void** submitSolution(**int** userId, **int** challengeId, String code) {

String insertSubmission = "INSERT INTO submissions (user\_id, challenge\_id, code, status, submission\_time) VALUES (?, ?, ?, 'Passed', NOW())";

**try** (Connection conn = *connect*(); PreparedStatement stmt = conn.prepareStatement(insertSubmission)) {

stmt.setInt(1, userId); stmt.setInt(2, challengeId); stmt.setString(3, code); stmt.executeUpdate();

*updateScore*(userId, 10); // Assume each successful submission adds 10 points System.***out***.println("Solution submitted successfully and score updated.");

} **catch** (SQLException e) { e.printStackTrace();

}

}

**public static void** updateScore(**int** userId, **int** score) {

String query = "UPDATE users SET score = score + ? WHERE user\_id = ?";

**try** (Connection conn = *connect*(); PreparedStatement stmt = conn.prepareStatement(query)) { stmt.setInt(1, score);

stmt.setInt(2, userId); stmt.executeUpdate();

} **catch** (SQLException e) { e.printStackTrace();

}

}

**public static** List<String> getLeaderboard() { List<String> leaderboard = **new** ArrayList<>();

String query = "SELECT username, score FROM users ORDER BY score DESC";

**try** (Connection conn = *connect*(); PreparedStatement ps = conn.prepareStatement(query)) { ResultSet rs = ps.executeQuery();

**while** (rs.next()) {

leaderboard.add(rs.getString("username") + ": " + rs.getInt("score") + " points");

}

} **catch** (SQLException e) { e.printStackTrace();

}

**return** leaderboard;

}

**public static** List<UserScore> getTopScores() { List<UserScore> scores = **new** ArrayList<>();

String query = "SELECT username, score FROM users ORDER BY score DESC LIMIT 10";

**try** (Connection conn = *connect*(); PreparedStatement ps = conn.prepareStatement(query)) { ResultSet rs = ps.executeQuery();

**while** (rs.next()) {

String username = rs.getString("username");

**int** score = rs.getInt("score");

scores.add(**new** UserScore(username, score));

}

} **catch** (SQLException e) { e.printStackTrace();

}

**return** scores;

}

}

**5. RESULTS AND DISCUSSION**

**Results:**

The Coding Platform Management system was developed and successfully tested with basic functionalities such as user registration, login, project management, repository creation, and commit tracking. Below are the key observations from the testing and operation of the system:

1. **User Registration and Authentication:**
   * The system allows users to sign up with a unique username and email. During registration, the password is securely hashed using the SHA-256 algorithm.
   * After registration, users can log in using their username and password, with proper authentication mechanisms in place.
   * The system prevents duplicate usernames and email addresses, ensuring data consistency and user uniqueness.
2. **Project Management:**
   * Users can create multiple projects associated with their account. Each project is uniquely identified and stored in the database.
   * Projects are successfully linked to specific users, and each user can manage their own set of projects.
3. **Repository Management:**
   * Repositories can be created under each project, enabling the user to organize their codebases.
   * The system successfully stores repository names, and repositories are linked to the projects they belong to.
   * Multiple repositories can be created for each project, offering flexibility in managing various components or modules.
4. **Commit Tracking:**
   * Users can record commits within a repository. Each commit includes a commit message and a timestamp, stored in the database.
   * The system successfully tracks commits for each repository and links them to their corresponding repository.
   * The system supports viewing the list of commits made to each repository, ensuring that users can trace the history of their work.
5. **Database Performance:**
   * The use of SQLite ensures that the database is lightweight, fast, and easily manageable for small-scale applications.
   * Database queries for adding and retrieving user data, project details, repositories, and commits executed efficiently, without significant delays.
   * Data integrity is maintained through foreign key relationships between tables (users, projects, repositories, commits).

**Discussion:**

The Coding Platform Management system demonstrates the feasibility of implementing a basic platform for managing code projects, repositories, and version control within a simplified, local environment. However, while the system meets the core objectives of managing users, projects, and commits, several enhancements and challenges must be considered:

1. **Scalability and Performance:**
   * The current implementation uses SQLite, which is ideal for small-scale, single-user applications. However, for large-scale applications with many concurrent users and a large number of projects, repositories, and commits, a more robust database system (e.g., PostgreSQL or MySQL) may be required.
   * As the user base and the amount of data grow, the system's performance could degrade. Thus, optimizing database queries, indexing, and scaling the system architecture would be necessary for handling higher loads.
2. **Security and Privacy:**
   * Although passwords are hashed using SHA-256, it is recommended to use more advanced techniques for password security, such as bcrypt or Argon2, which are more resistant to brute-force attacks.
   * The system currently does not implement features like email verification, password recovery, or role-based access control, which are essential for a production-level system. Implementing these features would improve security and usability.
3. **User Interface:**
   * The current system is a command-line interface (CLI) application. While it is functional, a web-based or graphical user interface (GUI) would provide a more intuitive and user-friendly experience for developers.
   * A web application using frameworks like Django, Flask, or Node.js would make it easier for users to interact with the system from any device with an internet connection.
4. **Version Control Features:**
   * The system offers basic commit tracking, but it lacks more advanced version control features like branching, merging, or conflict resolution. Implementing such features would make the platform more competitive with popular coding platforms like GitHub, GitLab, or Bitbucket.
   * Additionally, integrating with existing version control systems like Git could enhance functionality and allow users to interact with the platform more effectively.

1. **Collaboration and Code Sharing:**
   * The current system does not support real-time collaboration or code sharing between users. Adding features for team collaboration, such as pull requests, code review systems, or chat functionalities, would significantly enhance its value.
   * Allowing multiple users to collaborate on the same project and providing version control features such as merge requests could make the platform more appealing to teams working on larger codebases.
2. **Backup and Data Recovery:**
   * Since the platform is running on a local SQLite database, there is no built-in mechanism for backup or data recovery. Implementing automatic backups and data recovery strategies would ensure the safety of the user's work in case of system failures or data loss.
3. **Code Compilation and Execution:**
   * A feature for compiling and running code directly within the platform would enhance the user experience. Allowing users to write, test, and debug code in different programming languages would make the platform a more complete coding environment.
   * Integrating an IDE-like environment (either web-based or standalone) could make the platform more powerful and suitable for coding and testing code snippets.

**6.CONCLUSION:**

The Coding Platform Management system successfully demonstrates a simple and effective way to manage projects, repositories, and commits. The system fulfills the basic objectives of a coding management platform, providing core functionalities like user authentication, project management, and commit tracking.

However, several areas need to be addressed to improve scalability, security, user experience, and functionality. Moving forward, the system could be enhanced with features such as a web-based interface, advanced version control capabilities, collaboration tools, and integration with existing version control systems like Git. By expanding these features and addressing the current limitations, this platform could evolve into a robust tool for managing coding projects and collaboration in a professional development environment.