Yarmouk University

College of Information Technology

and

Computer Science

Department of Information Systems

Data Science & Artificial Intelligence program

ShareMeCode

Students' names

Mahmoud Alaaraj
Nabeel Ismaeel
Wadea Khader

Supervisor

Dr. Suboh Alkhushayni

FIRST TERM 2024

Acknowledgement

We would like to express our deepest gratitude to Dr. Suboh Alkhushayni, our supervisor, for his continuous support, guidance, and insightful feedback throughout the development of the ShareMeCode platform. His expertise in software engineering, Artificial Intelligence, and dedication to student success greatly contributed to the quality and direction of our work.

We are also thankful to the Department of Information Systems at Yarmouk University and the Data Science & Artificial Intelligence program for providing the academic foundation and resources necessary to undertake such a project.

Our sincere appreciation goes to our families and peers for their encouragement and support during this journey. Special thanks to all those who provided technical feedback, helped test our application, or offered motivation during critical development stages.

This project would not have been possible without the collaboration, hard work, and mutual support among the team members: Mahmoud Alaaraj, Nabeel Ismaeel, and Wadea Khader.

Table of contents

List of Tables

List of Figures

1. Introduction

ShareMeCode is a powerful web-based development platform designed to streamline real-time collaboration among developers across the internet. It eliminates the traditional barriers of setting up a development environment by removing the need for IDE installations, programming language configurations, or local setup. With a browser and an internet connection, contributors can instantly dive into live coding sessions, manage project structures, and build applications together, regardless of their physical location.

The platform empowers users with full project management capabilities, allowing them to create and organize new projects, directories, and files directly from the interface. It also supports code execution for multiple programming languages, enabling users to provide inputs, view outputs, and capture error messages—all within the same window. A key innovation is its distributed code execution service, which leverages message queues and containerization for scalable and isolated processing. Furthermore, ShareMeCode introduces an advanced AI-powered code generation feature, allowing users to dynamically generate code snippets based on natural language prompts, significantly enhancing productivity. This feature now includes a sophisticated "LLM-as-a-Judge" evaluation mechanism to select the most optimal generated code. Live editing is a core feature, where changes made by one contributor are instantly visible to all collaborators through WebSocket-based real-time synchronization, ensuring seamless teamwork and eliminating version drift.

To enhance collaborative workflows, the editor integrates a sophisticated version control system. Users can commit changes, revert to previous versions, and perform advanced project operations like cloning and forking. The application also features OAuth 2.0 authentication, allowing users to log in securely with Google or GitHub accounts. Role-Based Access Control (RBAC) is implemented to define specific permissions for different user roles, such as admin, editor, or viewer.

The server-side, built with Java and Spring Boot, provides RESTful APIs for all operations and manages persistent storage of files and user data using a customized file system and a MySQL database.

A modern ReactJS interface offers an intuitive and responsive experience on the client side. For code execution, Docker containers are utilized to ensure safe, isolated, and language-agnostic environments, allowing the platform to support multiple languages while maintaining strong security boundaries and system stability.

2. Problem Statement

Modern software development faces significant challenges in fostering effective collaboration, especially within geographically dispersed teams. The need for individual developers to set up and maintain complex local development environments, including IDEs, compilers, and dependencies, often leads to inconsistencies, conflicts, and delays. Real-time code sharing and synchronized editing remain difficult to achieve with traditional tools, hindering productivity and efficient problem-solving. Furthermore, managing project versions and ensuring secure access control across multiple contributors adds another layer of complexity. There is a clear demand for a streamlined, web-based solution that minimizes setup overhead, facilitates seamless real-time collaboration, and provides robust project management and security features, augmented by intelligent code generation capabilities that leverage advanced evaluation techniques to accelerate development.

3. Project Objectives

The primary objectives for the development of the ShareMeCode platform are to:

- Provide a web-based, real-time collaborative development environment for developers across the internet.
- Eliminate the necessity for local IDE installations, programming language configurations, or complex local setups.

- Enable users to instantly engage in live coding sessions with only a browser and an internet connection.
- Offer comprehensive project management capabilities, including the creation and organization of projects, directories, and files directly from the user interface.
- Support code execution for multiple programming languages, allowing users to provide inputs, view outputs, and capture error messages within the platform with a scalable system.
- Implement real-time synchronization of code changes made by collaborators using WebSockets.
- Integrate a sophisticated version control system to allow users to commit changes, revert to previous versions, and perform operations like cloning and forking.
- Establish secure user authentication through OAuth 2.0, supporting login with Google or GitHub accounts.
- Implement Role-Based Access Control (RBAC) to define and manage specific permissions for different user roles within projects.
- Develop a robust backend using Java Spring Boot to provide RESTful APIs and manage persistent storage using a customized file system and MySQL database.
- Create an intuitive and responsive client-side interface using ReactJS.
- Utilize Docker containers for safe, isolated, and language-agnostic code execution environments.
- Integrate an AI-powered code generation service to assist developers in writing code more efficiently, incorporating an "LLM-as-a-Judge" mechanism for response evaluation and selection.

4. Literature Review

The ShareMeCode project builds upon established principles and technologies in distributed computing, collaborative software engineering, and web application development. The concept of a web-based Integrated Development Environment (IDE) draws parallels with existing cloud IDEs, which aim to abstract away local environment complexities by moving development workflows to the browser. Real-time collaboration, a core feature, is heavily reliant on WebSocket technology, a bidirectional communication protocol over a single TCP connection, commonly used for interactive web applications.

The distributed execution architecture, leveraging message queues like RabbitMQ, aligns with microservices patterns for scalability and resilience in handling computational tasks.

For code execution, the adoption of containerization technologies like Docker aligns with modern DevOps practices, providing isolated, reproducible, and portable environments for diverse programming languages. This approach mitigates "it works on my machine" issues and enhances security by sandboxing execution. Security mechanisms, including OAuth 2.0 for external authentication and Role-Based Access Control (RBAC) for granular authorization, are industry standards for secure web applications, ensuring controlled access to resources.

The integration of AI for code generation represents a cutting-edge advancement, drawing from recent developments in large language models (LLMs) and their application in software development to enhance developer productivity and automate repetitive coding tasks. Notably, the adoption of an "LLM-as-a-Judge" approach for evaluating and selecting optimal AI-generated code snippets reflects advanced techniques in prompt engineering and LLM evaluation, ensuring higher quality outputs by leveraging the reasoning capabilities of multiple models.

This approach mirrors industry trends seen in tools such as GitHub Copilot, while addressing unique challenges in collaborative environments. Finally, the integration of DevOps practices like Continuous Integration/Continuous Deployment (CI/CD) with GitHub Actions and Docker containerization exemplifies modern software delivery pipelines, promoting automation, consistency, and rapid iteration.

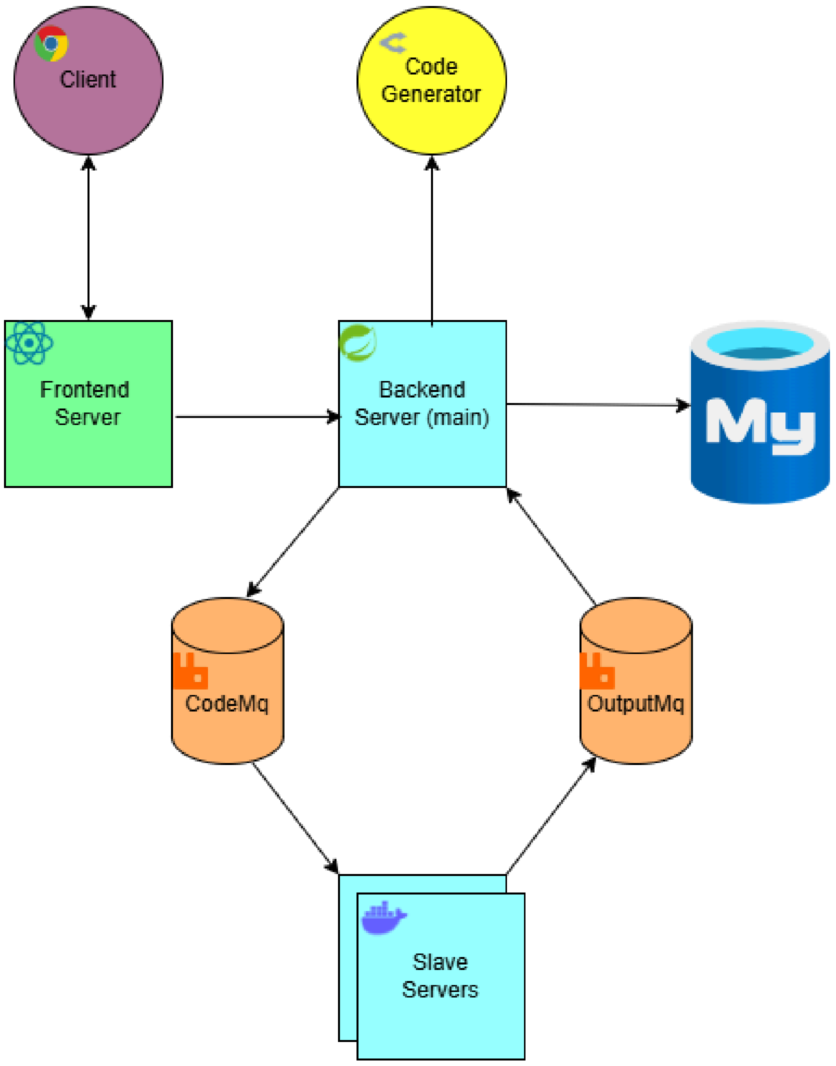
5. Methodology

The development of the ShareMeCode platform adopted a structured and layered approach, integrating modern software engineering principles with robust architectural design and DevOps practices. This methodology ensured a scalable, secure, and highly collaborative system.

System Architecture

ShareMeCode operates as a full-stack web application. The server-side, implemented using Java Spring Boot, provides RESTful APIs for all operations and manages persistent storage of files and user data using a customized file system and a MySQL database. The client-side is a modern ReactJS interface that offers an intuitive and responsive user experience. Real-time collaboration is facilitated through WebSocket-based communication, while code execution is handled by a distributed service leveraging message queues and containerization.

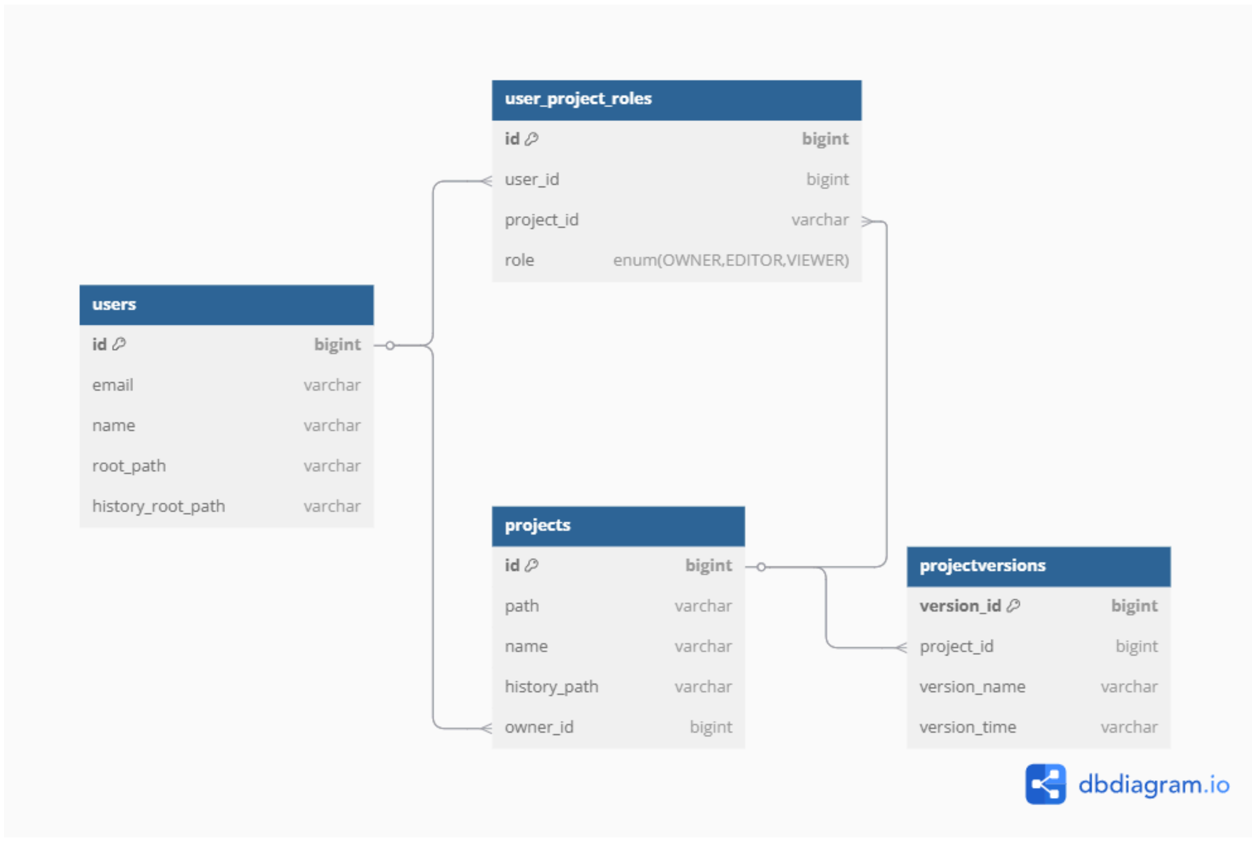
The integration of an AI-powered code Generation API into the platform. This feature allows users to generate code snippets directly within the editor using natural language prompts, streamlining the development workflow.



Backend Development

The backend component is built using the Java Spring Boot framework, responsible for building the server, creating RESTful API endpoints, connecting to the database using Spring JPA, and configuring security settings. It consists of several interconnected packages:

- **Model Package:** This package is responsible for creating and managing database entities. Key entities include: User (manages system users and their information), Project (manages user projects), ProjectVersion (manages project versions for version control), and UserProjectRole (manages project contributors and their roles). The UML diagram below illustrates the relationships between these entities.



- **Storage Package:** This package manages the file system and MySQL database storage. It includes a StorageService for business logic and lower-level operations on the file system and database, such as managing user projects, creating, renaming, moving, and copying directories and files, and retrieving their contents. A REST controller in this package makes these operations accessible from the frontend.

- **Execution Package:** Designed to manage code execution, this package handles "Run" requests from the user interface and returns results (outputs and/or errors). It utilizes ProcessBuilder with Docker to isolate processing from the main server component.

- **Socket Package:** This package configures and handles WebSockets and WebSocket clients. It sets up endpoints, message formats, and controllers for real-time communication between project collaborators within the same session. Multiple sessions can be served simultaneously without overlapping.

- **Security Package:** This package contains all security configurations, services, and controllers for the project. It implements a custom filter chain and a role-based access controller.

- **Version Control Package:** This package manages version control operations such as making commits, clones, forks, and reverting to previous versions. It handles how historical versions are stored, retrieved, and represented.

Frontend Development

The frontend was developed using Vite to build a ReactJS project, leveraging various libraries to simplify implementation and ensure clean code. The user interface is structured around three main web pages:

- **Root Page:** This is the entry page of the website, primarily used for logging in with Google or GitHub accounts via OAuth 2.0.

- **Dashboard Page:** This page displays all projects accessible to the user, whether owned by them or shared as an editor or viewer. Users can easily create new projects or open existing ones from this page.

- **Editor Page:** On this page, users can view and manage their project contents. It provides functionalities for editing code, creating directories and files, running code, adding new contributors, and managing version control (commits, reverts, clones, forks). Real-time updates on the code are a core feature, ensuring simultaneous collaboration among connected contributors.

Code Execution Environment

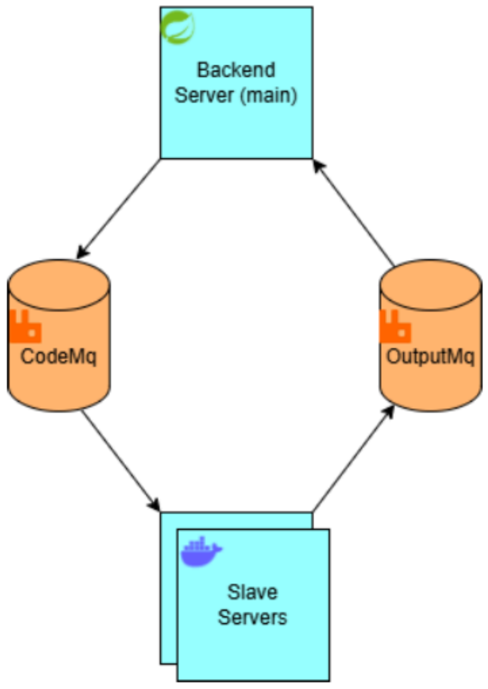
ShareMeCode supports code execution for various programming languages within isolated environments using Docker containers.

- **Supported Languages and Docker Images:** Three languages are currently supported, each with its corresponding Docker image:

Language	Compiler/Interpreter	Docker Image
Java	JDK 17	openjdk:17
C++	GCC 23	gcc:latest
Python	Python 3.9	python:3.9

- **Used Mechanism and The Command Line:** When a user initiates a "Run" request to the /executeFile endpoint, the system detects the project, file, and language. The appropriate Docker image is selected, project files are copied to the /app directory within the image, and the code is executed using a pre-specified command line. Outputs and errors are captured using ProcessBuilder and returned as the response. The general command line structure is docker run -rmi -v <pathToCode>:/app/ <imageName> bash -c <runCommand>. This command runs Docker, starts a container (run), removes it after exit (--rm), keeps STDIN open (-i), mounts host directories into the container (-v), specifies the Docker image, and executes a bash shell command within the container.

- **Distributed Code Execution:** The execution code service facilitates the distributed processing of user-submitted code through a series of coordinated steps involving message queues and containerized execution. The process is illustrated in the Figure, which depicts the interactions between the key components of the system. The workflow begins with the user submitting code via an HTTP request to the Main Server. The Main Server then packages the code and associated data into a message and publishes it to a RabbitMQ queue named codeQueue, then the thread that holds the request will be locked. Multiple Execution Servers are actively listening to the codeQueue, and one of them consumes the message. The selected Execution Server spawns a Docker Container to execute the code in an isolated environment.



Upon completion, the Docker Container returns the output to the Execution Server, which then publishes the result to another RabbitMQ queue, outputQueue. The Main Server, listening to outputQueue, retrieves the output, and then the thread that is locked will be released to return the output to the user in the same HTTP request, completing the cycle. This architecture leverages RabbitMQ for asynchronous message passing, ensuring scalability through multiple Execution Servers, and uses Docker Containers for secure and isolated code execution. The visual Figure provides a clear overview of the distributed system, highlighting the flow of data and the roles of each component.

AI-Powered Code Generation

ShareMeCode integrates an advanced AI code generation capability to assist developers, significantly enhancing productivity and code quality through an innovative evaluation mechanism. This feature allows users to embed natural language prompts directly within their code using a specific syntax (e.g., $generate a Python function to reverse a string$).

When a user submits code containing such prompts, the CodeGenerationController exposes a /codeGenerator/generate API endpoint that receives the user's code and the target language. The CodeGenerationService then identifies these embedded prompts. Instead of querying a single Large Language Model (LLM), the service now makes multiple parallel API calls to diverse LLM providers or different models. Each of these APIs generates a response (code snippet) for the given prompt.

Following the generation phase, ShareMeCode employs an **"LLM-as-a-Judge" evaluation mechanism** to select the optimal code snippet. This process involves:

1. **Peer Evaluation:** Each of the generating LLMs (or a designated set of evaluation LLMs) is prompted to act as an evaluator. They receive the original prompt and *all* the generated responses from the other APIs.

2. **Scoring:** Each evaluating LLM assesses the quality of *each* generated response, assigning a score from 0.0 to 100.0 based on a predefined set of criteria:
    A. **Correctness (40 percent)** — Does the code meet the functional requirements of the prompt?
    B. **Clarity (15 percent)** — Is the code easy to read and understand?
    C. **Idiomatic Use (15 percent)** — Does the code follow best practices and idiomatic patterns for the language?
    D. **Error Handling (10 percent)** — Does the code properly anticipate and handle errors?
    E. **Performance (10 percent)** — Is the code reasonably efficient?
    F. **Maintainability (10 percent)** — Is the code modular, well-structured, and easy to modify?
3. **Averaging:** The scores provided by all evaluating LLMs for a specific generated response are averaged.
4. **Best Response Selection:** The code snippet with the highest average score across all evaluators is selected as the best response.

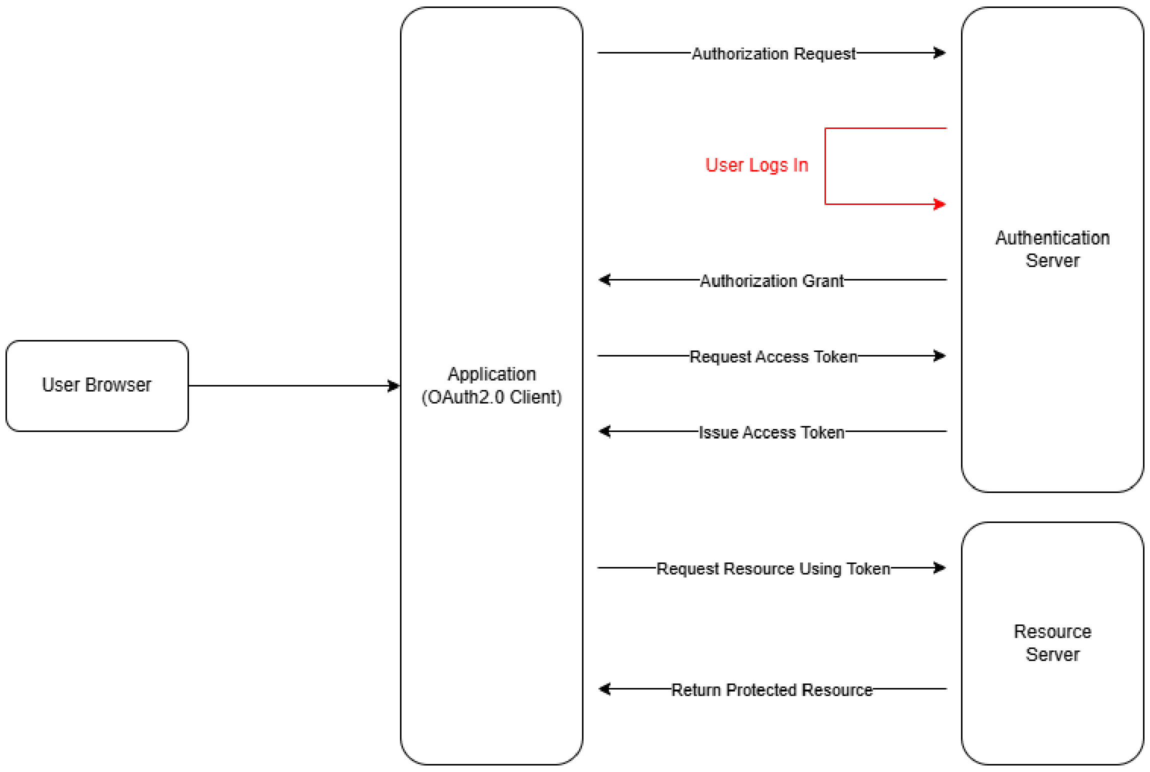
This multi-model generation and "LLM-as-a-Judge" evaluation approach ensures higher quality, more robust, and more relevant code suggestions, significantly accelerating development by automating the creation of complex logic based on natural language descriptions with an intelligent selection process

Security and Authentication

The project implements robust security and authentication measures:

- **Security Filter Chain and OAuth2 login:** The security filter chain is configured to require authentication for all paths except those necessary for login/registration and their associated sources (e.g., /, /login, /logout, /index.html, /assets/**, /static/**). All other requests require authentication. OAuth 2.0 login is configured to use a loginSuccessHandler upon successful authentication for custom actions.

Exception handling includes an authenticationEntryPoint that is invoked if an unauthenticated user attempts to access a protected resource, commonly returning a 401 Unauthorized status. Form-based login is explicitly disabled, indicating reliance on OAuth 2.0. Logout functionality redirects the user to the root path upon success. CSRF protection is disabled.



- **Role-Based Access Control (RBAC):** A role-based access control system manages different types of contributors to a project: owner/admin, editor, or viewer. A dedicated service and controller ensure that operations are performed by authorized users correctly. The project owner has the authority to grant, remove, or change roles within their project. This system relies on a database that stores contributors and their roles for each project.

DevOps Practices

Several DevOps practices have been integrated to ensure reliability, consistency, and ease of deployment:

1. **Containerization with Docker:** The entire application stack is containerized using Docker. A Dockerfile builds the Spring Boot backend and React frontend into a single image. A docker-compose.yml file orchestrates application services, including the MySQL database, ensuring consistent multi-container deployments.

2. **Environment Configuration:** Externalized configuration via application.properties enables easy management of environment-specific variables. Sensitive credentials, such as database passwords and OAuth client secrets, are managed through environment variables or secret management in production.

3. **Dependency Management:** Maven is configured to resolve all dependencies during the image build process, preventing runtime issues and speeding up deployments.

4. **Frontend Integration in CI:** The React frontend is built using npm within the Docker context and automatically copied into the backend's static resources folder. This ensures a seamless full-stack build process that avoids version mismatches.

5. **Database Initialization & Persistence:** MySQL data is persisted using Docker volumes, ensuring data remains intact across container restarts. The database is automatically created and updated based on the JPA ddl-auto=update strategy.

6. **Logging and Debugging:** Spring Security logs are enabled at TRACE level to facilitate debugging of OAuth login flows. Logs from all containers are accessible via Docker, aiding observability during development and testing.

7. **GitHub Action:** A GitHub Action workflow streamlines the end-to-end CI/CD process by automatically building, testing, and deploying the application upon code changes. The pipeline checks out the repository code, sets up Docker Builds, and authenticates with Docker Hub to build and tag a new Docker image. Tests are executed to ensure code quality, and on successful validation, the image is pushed to Docker Hub.

Principles and Techniques

The project adhered to several software engineering principles and utilized common techniques:

a. **Clean Code:** Principles of clean code were followed:
   i. **Class Naming:** Class names are nouns, specific, and use PascalCase (e.g., WebSocketConfig, MessageController, FileNode).
   ii. **Method Naming:** Names are descriptive, concise, use camelCase, and are specific (e.g., createDirectory, getFileContent, hasRoleOnProject).
   iii. **Small Functions:** Functions are designed to perform one specific task well (e.g., createDirectory creates directories, rename renames, getFileContent/setFileContent read/write file content).
   iv. **No Duplication (DRY):** While some validations are repeated, the code structure ensures that logic is not copied across unrelated methods, and behavior is encapsulated in distinct units for easier extraction of repeated logic.

b. **Effective Java:** Examples of following Effective Java principles include:
   i. **Use exceptions only for exceptional conditions (Item 69):** Methods correctly validate preconditions (e.g., Files.isDirectory) and throw IllegalArgumentException or NoSuchFileException for misuse, ensuring exceptions are thrown only when truly exceptional cases occur.

ii. **Validate method arguments (Item 49):** Methods like createDirectory and createFile check that rootPath is a directory before proceeding, preventing unexpected bugs.

iii. **Empty collections or null-safe objects rather than null (Item 54):** The getDirStructure method returns a well-structured FileNode object rather than null, adhering to a safety mindset.

iv. **Prefer dependency injection to hardwiring resources (Item 5):** The class uses Spring's @Value("${file.storage.root}") to inject the root path, improving flexibility and testability.

c. **SOLID Principles:**

i. **Single Responsibility Principle (SRP):** Classes have one clear reason to change. For example, ProjectRoleService is solely responsible for handling logic around user roles on projects.

ii. **Open/Closed Principle (OCP):** Software entities are open for extension but closed for modification. ExecutorInterface defines a contract for code execution without caring about language details. Language-specific classes (CPPExecutor, JavaExecutor, PythonExecutor) implement this interface, extending behavior without modifying existing implementations.

iii. **Liskov Substitution Principle (LSP):** Objects of a superclass or interface can be replaced with objects of its subclasses or implementations without altering program correctness. Anywhere ExecutorInterface is used, CPPExecutor, JavaExecutor, or PythonExecutor can be plugged in, and the program will function properly. Subtypes do not violate expectations set by the interface.

d. **Design Patterns:**

i. **Factory Design Pattern:** The ExecutorFactory class creates executors by taking the desired language and returning an implementation of ExecutorInterface (e.g., JavaExecutor, PythonExecutor).

ii. **Strategy Design Pattern:** The ExecutorFactory also enables selecting an algorithm's behavior at runtime, with three predefined algorithms (JavaExecutor, PythonExecutor, CPPExecutor) that can be chosen based on the language of the sent code.

6. Results and Discussion

The ShareMeCode project successfully developed a robust, web-based collaborative development platform. The core functionality of real-time collaborative editing is effectively achieved through WebSocket-based synchronization, allowing multiple users to work on the same file with instant updates. The implementation of a Docker-based execution environment ensures secure, isolated, and language-agnostic code execution for Java, C++, and Python, providing a consistent and reliable runtime for user code.

Project management capabilities, including file and directory creation, manipulation, and a version control system, are fully integrated, empowering users with comprehensive control over their projects. The security framework, incorporating OAuth 2.0 authentication with Google and GitHub and a robust Role-Based Access Control (RBAC) system, effectively manages user access and permissions, ensuring data integrity and user privacy. A significant achievement is the integration of the AI-powered code generation feature, which dramatically improves developer productivity by allowing natural language prompts to generate executable code. This feature is further enhanced by an "LLM-as-a-Judge" evaluation mechanism that selects the most optimal code from multiple AI responses, ensuring higher quality and more reliable suggestions.

The backend, built with Java Spring Boot, provides a stable and scalable foundation with well-defined RESTful APIs and persistent data storage. The ReactJS frontend offers an intuitive and responsive user interface across different pages, enhancing user experience.

Adherence to software engineering best practices like Clean Code, SOLID principles, and the use of design patterns (Factory, Strategy) has resulted in a maintainable, extensible, and high-quality codebase. Furthermore, the integration of DevOps practices, including Docker containerization and a comprehensive

GitHub Actions CI/CD pipeline, significantly improves deployment consistency, speed, and overall application reliability. The automated pipeline from code changes to deployment with minimal downtime represents a significant achievement in modern software delivery.

7. Limitations

While ShareMeCode offers extensive functionality, certain limitations exist:

a. **Network Dependency:** As a purely web-based platform, continuous internet connectivity is required for all collaborative and development activities. There are no inherent offline capabilities.
b. **Comprehensive IDE Features:** While providing core editing and execution, the platform may not replicate the full suite of advanced features found in sophisticated desktop IDEs (e.g., advanced debugging tools, integrated development tools, extensive plugin ecosystems).
c. **Advanced Version Control UI:** The current version control system supports core operations like commits and reverts, but a more sophisticated user interface for complex Git operations (e.g., branching visualization, complex merges) could be developed.
d. **WebSocket Scalability:** While WebSockets provide real-time updates, scaling the real-time collaboration feature to support an extremely large number of simultaneous collaborators on a single project might require further optimization strategies.
e. **AI Model Limitations:** The effectiveness of the AI code generation feature is dependent on the capabilities and current training data of the underlying large language model (agentica-org/deepcoder-14b-preview:free). Complex or highly specialized prompts might yield less accurate or complete code.

8. Conclusion

The ShareMeCode project has successfully delivered a robust and efficient web-based collaborative development platform. It effectively addresses the challenges associated

with traditional development environments by providing a seamless, real-time coding experience, eliminating local setup complexities, and offering comprehensive project management and version control features.
The integration of secure authentication and authorization through OAuth 2.0 and RBAC ensures a controlled and safe environment for users. The strategic application of Docker for isolated code execution, coupled with a distributed execution service, enhances scalability and reliability. Furthermore, the innovative integration of an AI-powered code generation feature, now augmented with an "LLM-as-a-Judge" evaluation system, significantly boosts developer productivity by ensuring higher quality code suggestions. The implementation of a fully automated CI/CD pipeline via GitHub Actions underscores a commitment to modern, reliable, and scalable software development practices. The project demonstrates a strong foundation built upon sound software engineering principles, making it a valuable tool for collaborative programming.

9. Future Work

To further enhance the capabilities and user experience of ShareMeCode, the following areas are recommended for future development:

a. **Expand Language and Framework Support:** Integrate support for additional programming languages and popular frameworks to cater to a broader developer base.
b. **Advanced Debugging Tools:** Develop and integrate more sophisticated debugging functionalities directly within the editor interface.
c. **Enhanced Real-time Collaboration Features:** Introduce features such as integrated voice/video calls, screen sharing, or dedicated collaborative whiteboard functionalities.
d. **Code Linting and Quality Analysis:** Incorporate real-time code linting, static analysis, and code quality metrics to provide instant feedback to developers.
e. **Plug-in Architecture:** Implement a flexible plug-in architecture to allow for third-party extensions and customization by the community.

f. **Optimized Resource Management:** Investigate and implement advanced container orchestration techniques to optimize resource usage, especially under high concurrent user loads.
g. **Offline Mode:** Explore the feasibility of a limited offline mode, allowing users to work on projects without continuous internet connectivity and synchronize changes once reconnected.
h. **Advanced Version Control UI:** Develop a more visual and interactive interface for complex version control operations, such as branching, merging, and conflict resolution.
i. **Templating System:** Allow users to create and share project templates to accelerate new project setup.
j. **Refinement of AI Code Generation:** Continuously improve the AI code generation feature by exploring different LLM models, fine-tuning prompts, and allowing for more contextual understanding to generate highly accurate and relevant code snippets. Further research into advanced "LLM-as-a-Judge" methodologies, including dynamic weighting of evaluators or hybrid evaluation approaches, could also be explored to optimize response selection.

10. References

a. ***Docker Documentation.*** *(n.d.). Docker Overview. Retrieved from https://docs.docker.com*
b. ***Spring Boot.*** *(n.d.). Building REST Services with Spring. Retrieved from https://spring.io/guides/gs/rest-service*
c. ***ReactJS.*** *(n.d.). React Documentation. Retrieved from https://react.dev*
d. ***IETF.*** *(2011). RFC 6455: The WebSocket Protocol. Retrieved from https://tools.ietf.org/html/rfc6455*
e. ***RabbitMQ.*** *(n.d.). Messaging with RabbitMQ. Retrieved from https://www.rabbitmq.com*
f. ***Hardt, D.*** *(2012). The OAuth 2.0 Authorization Framework. RFC 6749. Retrieved from https://tools.ietf.org/html/rfc6749*
g. ***GitHub.*** *(n.d.). GitHub Copilot. Retrieved from https://copilot.github.com*

h. ***Agentica.*** *(n.d.). *DeepCoder-14b Model*. Retrieved from https://openrouter.ai/models/agentica-org/deepcoder-14b-preview*

11. Appendix

…