University of Pisa

Artificial Intelligence and Data Engineering

Distributed Systems and Middleware Technologies

FLconsole documentation

Authors: Çolak F. Messina F. Nocella F.

Academic Year 2023/2024

Contents

Introduction and Project Overview

Context and Project Objective

The goal of this project is to develop a system to manage Federated Learning (FL) experiments. FL is a decentralized machine learning approach where multiple devices collaborate to train a shared model while keeping their data locally. The project aims to provide a graphic interface with a web console to run FL experiments, enabling users to monitor their progress and analyze results.

Project Goals

Manage Message Passing

- Implement a communication system between the Java Web Application and Erlang FL Director. The FL Director is a software that starts experiments among the subscribed devices.
- Establish a reliable message passing protocol to exchange information seamlessly.

Implement a Web Console

- Develop a user-friendly Web Console to initiate and manage FL experiments.
- Provide centralized access to experiment statistics for easy monitoring and analysis.

Communication Protocol

To ensure effective communication between the Java Web Application and FL Director, a communication protocol will be defined. It will facilitate the exchange of messages and data in a structured format, ensuring consistency and compatibility across different components of the system. The FL director is an Erlang node, so it's necessary send the data in the proper way.

Data Variety

FL experiments generate various types of statistics, ranging from model performance metrics to training data distribution. To accommodate this diversity, a flexible data storage mechanism, such as DocumentDB, will be designed. This will allow for efficient storage and retrieval of experiment data while supporting scalability and adaptability.

Concurrent Experiment Execution

The system will support concurrent execution of multiple FL experiments to maximize resource utilization and efficiency. Java threads and ExecutorService will be used to manage experiment execution concurrently, ensuring optimal performance and resource allocation.

Real-Time Analytics

Real-time analytics capabilities will be implemented using WebSockets to enable seamless communication between the frontend and backend of the Web Console. This will facilitate real-time monitoring of experiment progress and display of relevant statistics as they become available.

Project Key Points

- Utilize DocumentDB for flexible storage of experiment statistics, allowing for efficient data management and retrieval.
- Implement concurrent execution of experiments using Java threads and ExecutorService to optimize resource utilization.
- Establish WebSocket communication for real-time data exchange, enabling seamless interaction between the frontend and backend.
- Define message formats and outline the structure of Erlang nodes for efficient communication, ensuring reliability and scalability.
- Choose a suitable message acknowledgment mechanism to ensure reliable delivery of messages, minimizing the risk of data loss.

Architecture and Frameworks

- Adopt the MVC (Model-View-Controller) pattern to structure the Web Console, promoting separation of concerns and maintainability.
- Utilize Spring as the Java framework for building the Web Console, leveraging its robust features and ecosystem for rapid development.
- Implement WebSockets to enable real-time data communication between the frontend and backend of the Web Console, providing a responsive and interactive user experience.
- Employ MongoDB as the database for storing experiment data, benefiting from its flexibility, scalability, and support for complex data structures.

Analysis

Requirements

In this section, we outline the functional and non-functional requirements necessary for the successful implementation of the project.

Functional Requirements

For Administrators:

- Administrators must be able to log in to the application.
- Administrators must be able to log out of the application securely.
- Administrators must be able to create new configurations for experiments.
- Administrators must be able to create experiments based on the configurations they have defined.
- Administrators must be able to initiate experiments and oversee their execution.
- Administrators must possess the authority to perform CRUD operations on configurations and experiments.

For Users:

- Users must be able to log in to the application.
- Users must be able to log out of the application securely.
- Users must be able to register for a new account within the application.
- Users must be able to monitor real-time progress of experiments.
- Users must be able to search for experiments based on configuration and experiment names.

Non-Functional Requirements

1. **Performance:** The system must handle a large number of concurrent users without significant performance degradation. Response times for critical operations should be kept within acceptable limits.
2. **Reliability:** The system should be highly available with minimal downtime. Data integrity must be maintained at all times.
3. **Security:** User authentication and authorization mechanisms must be implemented. Data transmission must be encrypted.
4. **Scalability:** The system should scale horizontally to accommodate increasing user loads and data volumes.
5. **Usability:** The user interface must be intuitive and error messages must be informative.

6. **Maintainability:** The codebase must be well-organized and documentation must be comprehensive.
7. **Compatibility:** The application must be compatible with a wide range of web browsers and devices. Integration with external systems must be seamless.

Constraints/Other Requirements

- **Regulatory Compliance:** The application must comply with relevant laws and regulations.
- **Hardware Requirements:** Specific hardware specifications may be necessary for hosting.
- **Localization Requirements:** The application may need to support multiple languages.
- **Data Migration:** Data from existing systems may need to be migrated.
- **Interact with FL Director:** Integration with the FL Director, including defining message formats and communication technologies.
- **Data Variability and Schemaless Handling:** The application must be capable of handling variable and schemaless data.
- **Deployment on Three Different Nodes:** The application must be deployed on separate nodes for redundancy and reliability.

Use Case Diagram

Actors

The actors who can interact with the web console system consist of the following:

- **User:** The user is the actor who can browse the system to view running and completed experiments and their results.
- **Admin:** The admin is the actor who can manage the system, including creating and deleting configurations and experiments, and viewing the results of experiments.

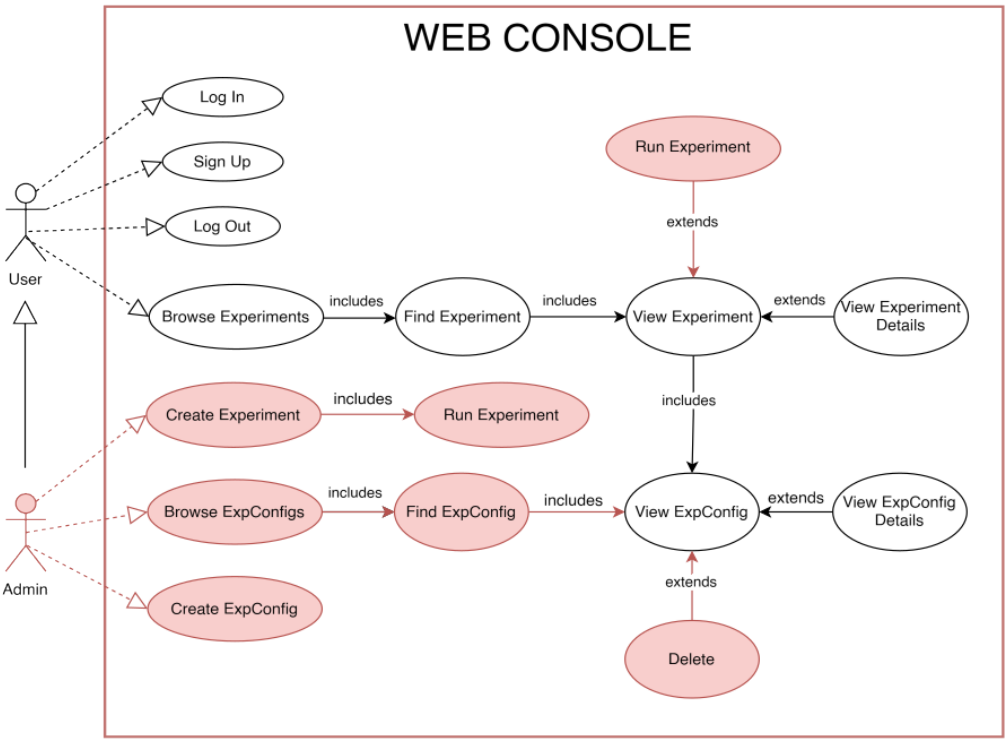

Figure 2.1: Use Case Diagram

Scenarios

In the following tables, we present several key use cases related to the management and execution of experiments within the application. These use cases cover actions performed by different actors, including users and administrators, and describe the steps involved in each scenario, along with pre-conditions and post-conditions.

Table 2.1: Use Case: Find Experiment

Use Case	**Find Experiment**
Primary Actor	User, Admin
Secondary Actor	–
Description	Allows the actor to find a specific experiment
Pre-Conditions	Actor must be logged in
Main event steps	1. The actor navigates to the "Search" feature 2. The actor enters the Experiment and/or the configuration name 3. The system searches for the list of experiments in database for matching results
Post-Conditions	The actor views a list of experiments matching the search criteria if there are any
Correlated Use cases	
Alternative event steps	–

Table 2.2: Use Case: Create Experiment

Use Case	**Create Experiment**
Primary Actor	Admin
Secondary Actor	–
Description	Allows the admin to create a specific experiment
Pre-Conditions	Actor must be logged in and has the admin privileges
Main event steps	1. Admin selects the option to create a new experiment. 2. Admin fills in the name and configurations for the experiment. 3. Admin confirms the creation of the experiment.
Post-Conditions	The experiment is successfully created.
Correlated Use cases	Run Experiment
Alternative event steps	–

Table 2.3: Use Case: Run Experiment

Use Case	**Run Experiment**
Primary Actor	Admin
Secondary Actor	–
Description	Allows the admin to start a specific experiment
Pre-Conditions	Actor must be logged in and have admin privileges
Main event steps	1. Admin selects the experiment and reaches the details page. 2. Admin presses the start button. 3. If the experiment has not started yet 3.1 the system will start the experiment. Otherwise 3.2 the system will show a message with the experiment status.
Post-Conditions	The experiment statistics are shown on the experiment details page and saved in the database.
Correlated Use cases	
Alternative event steps	–

Analysis Class Diagram

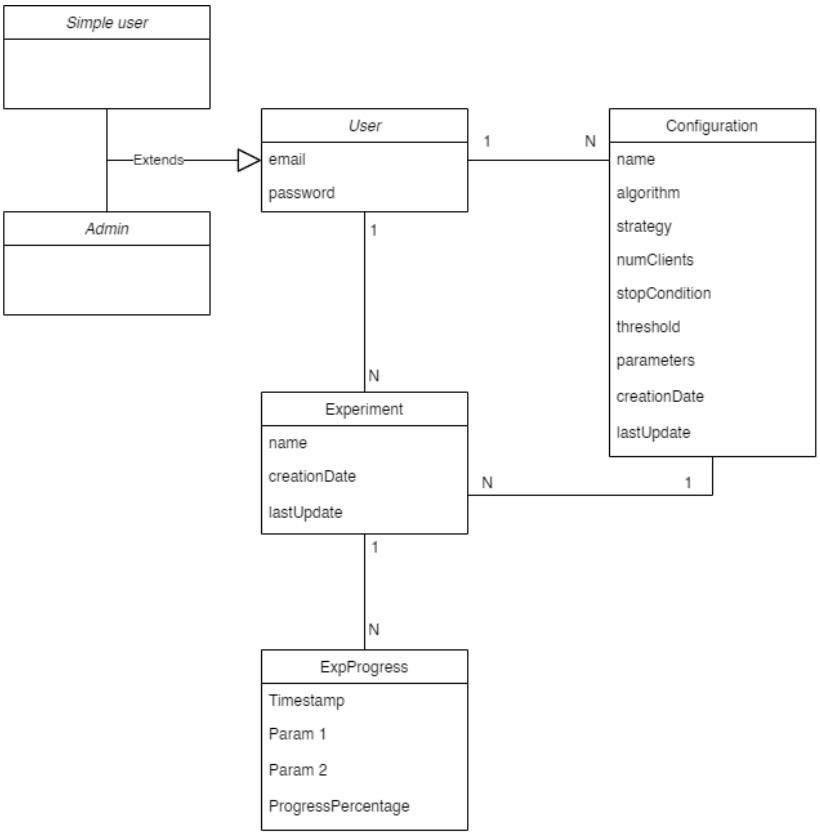

Figure 2.2: Class Diagram

Sequence Diagrams

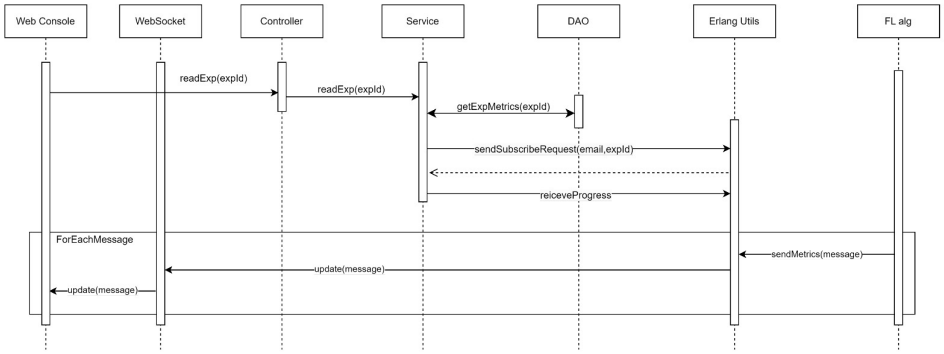

Figure 2.3: Sequence Diagram

The sequence diagrams of the project are presented in this subsection.

Design

Introduction

This chapter aims to provide a detailed overview of the software architecture and database design of the project. It is essential for understanding the organization and structure of the system, as well as the design choices made to ensure the efficiency, scalability, and robustness of the software.

The design of the software architecture focuses on the organization and distribution of software components, defining roles, responsibilities, and interactions among them. Key architectural decisions guiding the project's development will be presented within this context.

Additionally, the database design will be examined, with particular attention to the decision to use a NoSQL database like MongoDB. This decision was motivated by the need to adapt to the specific requirements of the project, including flexible management of unstructured data and horizontal scalability.

Software Architecture

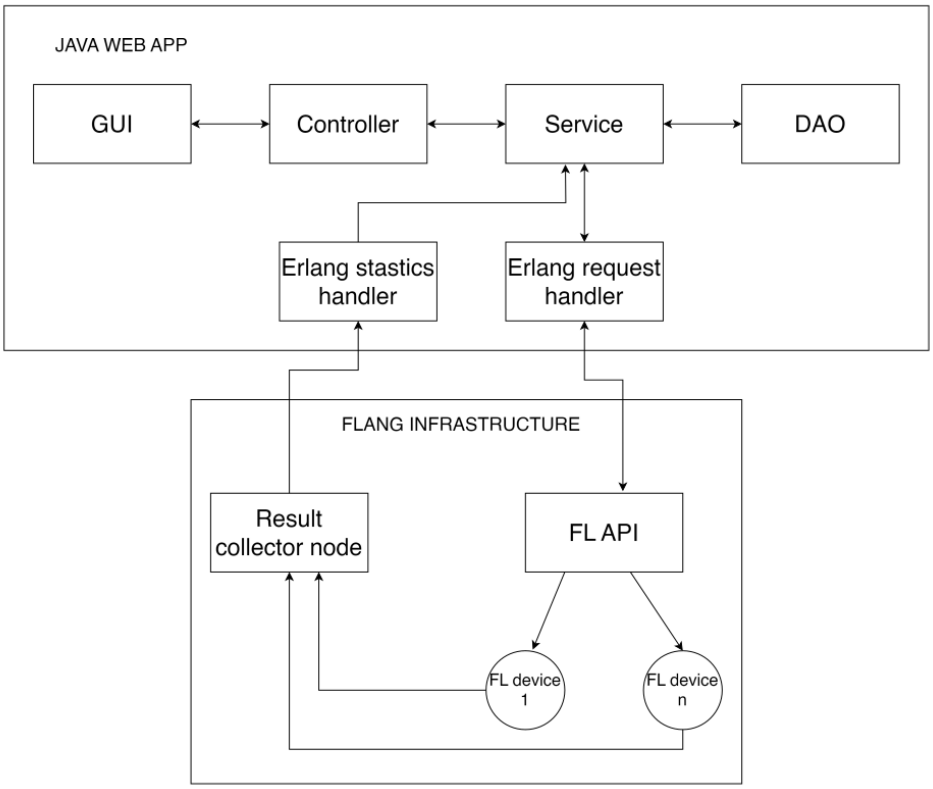

Figure 3.1: System Architecture

Database Design

MongoDB

Collections

ExpConfig document example:

```
{
    "id": "example_id",
    "name": "Example Experiment",
    "algorithm": "example_algorithm",
    "strategy": "example_strategy",
    "numClients": 10,
    "stopCondition": "example_condition",
    "threshold": 0.5,
    "parameters": {
        "param1": "value1",
        "param2": "value2",
        "param3": "value3"
    },
    "creationDate": "2024-03-14T00:00:00Z",
    "lastUpdate": "2024-03-14T12:00:00Z"
}
```

Experiment document example:

```
{
    "id": "example_id",
    "name": "Example Experiment",
    "expConfigSummary": {
        "id": "exp_config_id",
        "name": "Example Configuration",
        "algorithm": "example_algorithm"
    },
    "creationDate": "2024-03-14T00:00:00Z",
    "lastUpdate": "2024-03-14T12:00:00Z",
    "status": "Completed"
}
```

ExperimentMetrics document example:

```
{
    "id": "id",
    "expId": "exp_id",
    "type": "type",
    "hostMetrics": {
        "cpuUsage": 15.5,
        "memoryUsage": 65,
    },
    "modelMetrics": {
        "FRO": 0.154,
        "F1": 0.654,
    },
    "Timestamp": "2024-03-14T00:00:00Z",
    "expStatus": "running"
}
```

User document example:


```
{
"id": "example_user_id",
"email": "user@example.com",
"password": "example_password",
"creationDate": "2024-03-14T00:00:00Z",
"configurations": ["config_id1", "config_id2"],
"experiments": [
    {
    "id": "experiment_summary_id1",
    "name": "Experiment 1",
    "configName": "Configuration 1",
    "creationDate": "2024-03-14T06:00:00Z",
    },
    {
    "id": "experiment_summary_id2",
    "name": "Experiment 2",
    "configName": "Configuration 2",
    "creationDate": "2024-03-14T09:00:00Z",
    "lastUpdate": "2024-03-14T12:00:00Z"
    }
],
"role": "example_role"
}
```

Message Handler

Erlang for Message Passing

The message handler is implemented using the Erlang programming language. Erlang is a functional programming language designed for building scalable and fault-tolerant systems. It is particularly well-suited for building distributed systems, thanks to its lightweight processes and built-in support for message passing. In this project, it's utilized the Jinterface library, which allows to write Java code that can communicate with Erlang processes to send and receive messages, arriving from the FLang Infrastructure and vice versa.

Message Structure

- Error Message:

```
{
    "type": "error",
    "cause": "error_in_collecting_data",
    "timestamp": "2024-03-13T12:34:56"
}
```

- Stop Message:

```
{
    "type": "stop",
    "cause": "experiment_finished",
    "timestamp": "2024-03-13T12:34:56"
}
```

- Data Message:

```
{
    "type": "data",
    "parameters": {
        "param1": "value1",
        "param2": "value2"
    },
    "timestamp": "2024-03-13T12:34:56",
    "status": "running"
}
```

Description of the Erlang Message Handler Module

The Erlang message handler module is a crucial component of the system responsible for managing incoming messages, processing them accordingly, and facilitating communication between different parts of the distributed system. It encapsulates the logic for handling various types of messages, such as error notifications, stop signals, and data updates, ensuring proper routing and processing. Additionally, the module provides interfaces for sending and receiving messages, abstracting the underlying communication mechanisms and enabling seamless integration with other system components. Its robust design and fault-tolerant features contribute to the overall reliability and performance of the distributed system.

Implementation

Development Environment

To be able to have efficient and successful implementation of Federated Learning Web Console Project, having a well-chosen development environment is one of the most important aspects. In this section, it is specified that the necessary tools, frameworks, and configuration requirements of the project.

- **Programming Language:** Java is used for creating a Web Application. Erlang is used for facilitation the development of middleware component and FL director is an Erlang node. So that effective communication between the Web Application and the FL director is provided.
- **Frameworks:** Spring is used for Java framework. It ensures to integrate dependencies for WebSocket communication and MongoDB support. WebSocket is implemented to provide real-time communication between frontend and backend components.
- **Database Management:** Spring is used for Java framework. It ensures to integrate dependencies for WebSocket communication and MongoDB support. WebSocket is implemented to provide real-time communication between frontend and backend components.
- **Version Control:** Git is used for version control. It is used to manage the source code of the project. GitHub is used to provide a collaborative development with its version control system. Efficient code management and collaboration is ensured by using repositories which is provided by the platform itself.
- **Integrated Development Environment:** IntelliJ IDEA is used as an IDE. It is a Java integrated development environment for developing computer software. It is developed by JetBrains. It is used to write, compile, and run the code. It also provides a user-friendly interface for developers.
- **Build Automation:** Maven is used for build automation. It is a build automation tool used primarily for Java projects. It is used to manage the project's build, reporting, and documentation from a central piece of information. Maven is used to control project dependencies and build configurations.
- **Testing:** Junit testing is used for testing Java code.

Main Modules

Implementation of the project is structured by diving the project into modules. Each module ensures specific requirements of the project architecture. The modules are:

- Configuration
- Controller
- DAO (Data Access Object)
- DTO (Data Transfer Object)
- Model

- Service
- Utils

Configuration

Configuration classes of the Federated Learning Web Console project are created to provide responsibilities for configuring different parts of the application such as logging, execution, HTTP request handling, MVC setup and WebSocket communication. Efficient operation, security and scalability of the system can be ensured by those configuration properties.

Data Access

The data access classes are fulfilling the requirements of interacting with the database layers, providing data retrieval, storage, and manipulation. This module includes classes includes CRUD (create, read, update, delete) operations and query executions. With the Data Access classes such as ExpConfigDao, ExperimentDao, ExpProgressDao, UserDao the applications guarantee effective operations, management of experiments and tracking of the progress.

Data Transfer

Data Transfer layer contains a UserDto class to ensure the functionality of transferring data structure between different layers and components of the application. With the help of the UserDto class, user related information such as email, password and description will be able to be transferred between frontend, backend, and service layers. User information is transferred in a more standardized way for achieving better communication.

Service

Service module includes business logic and operations for ensuring the fully functional application. It provides data processing and interaction between different components. Service module includes:

- Cookie Service is for managing cookie operations such as cookie creation, retrieval, and deletion. The purpose of this service is ensuring session management and personalized user experience.
- Experiment Configuration Service is for implementing business logic for experiment configuration includes creation, deletion, retrieval and searching by some parameters.
- Experiment Service is for creating operations that are related with experiment like creation, running, deletion, retrieval and searching.
- User Service is implemented for ensuring business logic for user-based operations. Those operations include authentication of user, sign up, deletion of account, updating user information and retrieval of the user

User Interface

User Interface module is responsible for providing a user-friendly interface for the users. This module makes application functionalities visible for the end-user. It includes the following components:

- Login and Sign Up Page: This page is for user authentication and registration. Users can log in to the system by providing their email and password. If the user does not have an account, they can sign up by providing their email, password, and description.
- User Dashboard: This page is for displaying the experiments to the user. Users can see experiments and their progress on this page.

- Experiment Page: This page is for displaying the details of the experiment. The page shows the details of the experiment and its progress on this page.
- Admin Dashboard: This page is for displaying all experiments. Admins can see all experiments and their progress on this page and also it provides creating new experiments for the admin.
- Profile Page: The profile page allows users to view and manage their account settings and profile information.

Adopted Patterns and Techniques

During the implementation of the Federated Learning Web Console project, various patterns and techniques are adopted to ensure the efficiency, scalability, and maintainability of the application. These are some of the used patterns and techniques:

Model-View-Controller (MVC) Pattern

The Federated Learning Web Console project is implemented by following the Model-View-Controller (MVC) pattern. This pattern is used to separate the application into three main components: Model, View, and Controller. The Model represents the data and business logic of the application, the View represents the presentation layer, and the Controller handles the user input and updates the model and view accordingly. This pattern ensures a clean separation of concerns and makes the application easier to maintain and extend.

WebSocket Communication

WebSocket communication is implemented to provide real-time communication between the frontend and backend. This allows the application to send and receive messages in real-time without the need for polling or long-polling. WebSocket communication is used to update the user interface with the latest data and provide a seamless user experience.

Asynchronous Processing

Asynchronous processing techniques like Java threads and ExecutorService are e used to manage concurrent execution of experiments and other tasks. This allows the application to handle multiple requests and tasks simultaneously and improve performance and scalability.

Message Passing Protocol

To achieve seamless communication between Erlang FL director and Java web application, a customized and well specified message passing protocol is defined. This protocol guarantees the reliable and well-defined exchange of messages and data.

Testing

Testing methodologies are used to ensure about the reliability, correctness, functionality and quality of the Federated Learning Web Console. In this chapter, the testing methodologies used in the project are described. The testing methodologies are divided into two main categories: structural testing and functional testing.

Structural Testing

Structural testing, also known as white-box testing is applied to the project to ensure that the implemented code is working as expected and evaluate the internal structure of the system. For primary structure testing JUnit testing is applied as a testing methodology.

JUnit Testing

The JUnit testing performed on the project of FL Web Console. The JUnit testing is applied to various classes like DAOs and Services to check whether the implemented code is working as expected or not and specified requirement are hold by the methods. Some examples of the JUnit testing that performed on the classes:

UserDAO

The UserDAO class is an important component of the project which is responsible for interacting with the database to handle data related with users. With the help of the JUnit tests different scenarios are tested to ensure that the implemented code is working as expected and the requirements are fulfilled. This scenarios are including creating new user, deleting existing user, finding user by some criterias. These tests show the correctness of the CRUD operations of the UserDAO class. Below it can be seen an example of performing JUnit test for some methods in the UserDAO.

```
221     @Test  winternox *
222     void findListOfConfigurationsByEmail() {
223         // Given
224         String userEmail = "admin@example.com";
225
226         // When
227         List<String> retrievedConfigurations = userRepository.findListOfConfigurationsByEmail(userEmail);
228
229         // Then
230         assertNotNull(retrievedConfigurations);
231         assertFalse(retrievedConfigurations.isEmpty());
232
233         System.out.println(retrievedConfigurations);
234     }
235
```

Figure 5.1: Testing the method of findListOfConfigurationsByEmail() in UserDAO class

With this JUnit test method, the findListOfConfigurationsByEmail() method of the UserDAO class is tested. The test is performed by finding all the related configuration that are belong to the user with that email. The test is successful and the expected result is returned as a list of configurations.

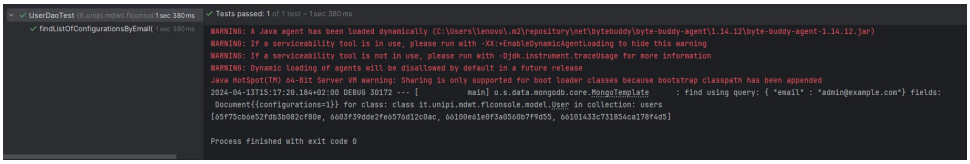

Figure 5.2: Testing result for the method of findListOfConfigurationsByEmail() in UserDAO class

ExperimentDAO

Experiment DAO is another important class of the project which is responsible for interacting with the database to handle data related with experiments. JUnit tests are created to ensure that the experiment related functions are working as expected and the requirements are fulfilled. The tests are including creating new experiment, updating existing experiment, deleting existing experiment, finding experiment by some criterias. Below it can be seen an example test method for update an experiment.

```java
@Test  feyzancolak99 *
void update() {
    // Create an experiment object
    Experiment experiment = new Experiment();
    experiment.setName("Save Test Experiment2");

    // Save the experiment
    Experiment savedExperiment = experimentDao.save(experiment);

    // Fetch all users
    List<User> users = userDao.findAll();
    for (User user : users) {
        // Check if the user has experiments associated with the updated experiment
        List<ExperimentSummary> updatedExperiments = user.getExperiments();
        if (updatedExperiments != null) {
            // Iterate over the user's experiments
            for (ExperimentSummary exp : updatedExperiments) {
                // If the experiment ID matches, update the name
                if (exp.getId().equals(savedExperiment.getId())) {
                    exp.setName(savedExperiment.getName());
                    // No need to break here, as multiple summaries might match
                }
            }
            // Save the updated user
            userDao.save(user);
        }
    }
    // Update the name of the savedExperiment
    savedExperiment.setName("Changed Name2");
    experimentDao.save(savedExperiment);
    // Fetch the updated experiment
    Optional<Experiment> updatedExperiment = experimentDao.findById(savedExperiment.getId());
    assertTrue(updatedExperiment.isPresent());
    // Check the updated name
    assertEquals( expected: "Changed Name2", updatedExperiment.get().getName());
}
```

Figure 5.3: Testing the method of update() in ExperimentDAO class

ConfigurationDAO

Test class for Configuration DAO is another example of JUnit test that is performed on the project. The Configuration DAO class is responsible for interacting with the database to handle storage and retrival of the system experiment configuration. Implementing JUnit for this class guarantees that system operates the data in an intended way for configuration class. Below there is an example test and test result for saveAndRetrieve() method of Configuration DAO class. The test is performed by saving a configuration and then retrieving it from the database. The test is successful and the expected result is returned.

```
@Test  winternox +2 *
void saveAndRetrieve() {
    // Given
    ExpConfig expConfig = new ExpConfig();
    expConfig.setName("TestConfig2");
    expConfig.setAlgorithm("fcmeans");
    expConfig.setCodeLanguage("python");
    expConfig.setClientSelectionStrategy("probability");
    expConfig.setClientSelectionRatio(1.0);
    expConfig.setMinNumberClients(0);
    expConfig.setMaxNumberOfRounds(5);
    expConfig.setStopCondition("max_number_rounds");
    expConfig.setStopConditionThreshold(5.0);

    Map<String, String> parametersList = new HashMap<>(Map.of( k1: "numFeatures",  v1: "16",  k2: "numClusters",  v2: "10",
            k3: "targetFeature",  v3: "16",  k4: "lambdaFactor",  v4: "2",  k5: "seed",  v5: "10"));
    expConfig.setParameters(parametersList);
    // When
    ExpConfig savedExpConfig = expConfigDao.save(expConfig);
    // Then
    assertNotNull(savedExpConfig.getId(),  message: "ID should not be null after save");
    // Retrieve the saved ExpConfig from the repository
    Optional<ExpConfig> retrievedExpConfigOptional = expConfigDao.findById(savedExpConfig.getId());
    // Assert that the retrieved ExpConfig matches the original one
    assertTrue(retrievedExpConfigOptional.isPresent(),  message: "Saved ExpConfig should be present");
    ExpConfig retrievedExpConfig = retrievedExpConfigOptional.get();
    assertEquals(expConfig.getName(), retrievedExpConfig.getName(),  message: "Name should match");
    assertEquals(expConfig.getAlgorithm(), retrievedExpConfig.getAlgorithm(),  message: "Algorithm should match");
    assertEquals(expConfig.getClientSelectionStrategy(), retrievedExpConfig.getClientSelectionStrategy(),  message: "Strategy should match");
    //assertEquals(expConfig.getNumClients(), retrievedExpConfig.getNumClients(), "NumClients should match");
    assertEquals(expConfig.getStopCondition(), retrievedExpConfig.getStopCondition(),  message: "StopCondition should match");
    // assertEquals(expConfig.getThreshold(), retrievedExpConfig.getThreshold(), "Threshold should match");
    assertEquals(expConfig.getParameters(), retrievedExpConfig.getParameters(),  message: "Parameters should match");
}
```

Figure 5.4: Testing the method of saveAndRetrieve() in Configuration DAO class

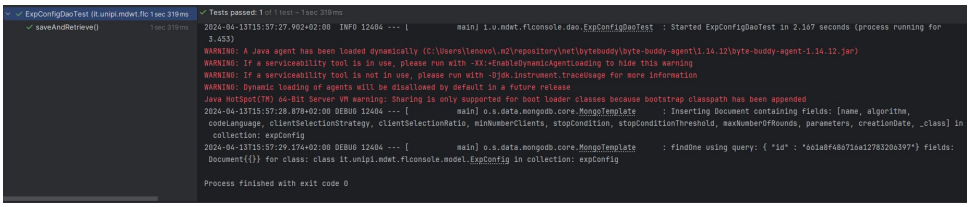

Figure 5.5: Testing result for the method of saveAndRetrieve() in Configuration DAO class

Functional Testing

Functional testing, also known as black-box testing is applied to the project to evaluate the system behaviour that needs to fulfill functional requirements. Functional testing helps to ensure that user expectations are provided in a right way. The functional testing is performed by creating test cases for the system.

Test cases are identified according to the functional requirements. It shows how the system should behave in different scenarios that are both normal and anormal. Those test cases are including user authentication, creating new configuration, creating new experiment, deleting experiment, finding experiment, finding configuration, deleting configuration, etc. Below there is a table that shows some examples of the test cases that are created for the project. As a result of the test cases, it can be said that the Federated Learning Web Console provides all the necessary functionalities and meets user expectations.

Test Cases

Table 5.1: Admin Test case

Id	Description	Input	Expected Output	Output	Outcome
A_T_01	Admin Login	Email: admin@example.com Password: Adm1nP@ss (valid credentials)	Login Successfully	Redirected to admin dashboard	Passed
A_T_02	Admin Login 2	Email: invalid@example.com Password: invalid (invalid credentials)	Error message displayed	Unable to login	Passed
A_T_03	Creating New Configuration	Adding all necessary values to the new FL configuration form	Configuration Created successfully	Configuration Created successfully	Passed
A_T_04	Creating New Configuration 2	Entering all values except stop condition	Error of missing value message displayed	Configuration is not created	Passed
A_T_05	Creating New Experiment	Name is written and FL configuration is selected	Experiment Created successfully	Experiment Created successfully	Passed
A_T_06	Starting an Experiment	Press Start Experiment button	Experiment starts	Experiment starts	Passed

Table 5.2: User Test case

Id	Description	Input	Expected Output	Output	Outcome
U_T_01	User Login	Email: firstTest@example.com Password: P@ssw0rd (valid credentials)	Login Successfully	Redirected to user dashboard	Passed
U_T_02	User Login 2	Email: wrong@example.com Password: invalid (invalid credentials)	Error message displayed	Unable to login	Passed
U_T_02	User Signup	Email: new@example.com Password: P@ssw0rd (valid input)	Sign up successfully	Sign up su and Redirected to user dashboard	Passed
U_T_02	User Signup 2	Email: new@example.com Password: invalid (invalid password)	Error message displayed	Unable to Signup	Passed

Conclusion

This chapter summarizes and highlights the key points of the document like architecture, implementation details, user interface components and discusses possible future directions for the project of Federated Learning Web Console.

Key Points

In this project Federated Learning Web Console is introduced which is a centralized platform for managing FL experiments with using Java and Erlang as primary programming languages, Spring as a Java framework and MongoDB as a database. As a main project architectural structure, the project follows the MVC pattern. Various and functional models are implemented in the project such as Configuration, Data Access Object, Services and User Interface with their specific roles and functionalities for serving the application. With the help of this architecture FL Web Console project ensures a scalable design and having seamless functionality between different components. The user interface includes essential pages to provide fully functional experience for the user. Those pages are login/signup, user/admin dashboard, profile page and experiment details.