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

1 Introduction and Project Overview **2**
1.1 Context and Project Objective 2
1.2 Project Key Points 2

2 Analysis **3**
2.1 Requirements 3
2.2 Use Case Diagram 4
2.3 Analysis Class Diagram 6
2.4 Sequence Diagrams 7

3 Design **9**
3.1 Introduction 9
3.2 Software Architecture 9
3.3 Database Design 10
3.4 Message Handler 12

4 Implementation **15**
4.1 Development Environment 15
4.2 Main Modules 15
4.3 Configuration 16
4.4 Data Access 16
4.5 Data Transfer 16
4.6 Service 16
4.7 User Interface 16
4.8 Adopted Patterns and Techniques 17
4.9 System Configuration 18

5 Testing **20**
5.1 Structural Testing 20
5.2 Functional Testing 23

6 Conclusion **25**

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. The system will use the Federated Learning Director (FL Director) to coordinate the execution of experiments among the devices. The FL Director is an Erlang node that manages the communication between the devices and the Web Console. The system will be designed to support concurrent execution of multiple FL experiments, real-time analytics, and flexible storage of experiment statistics. The project will adopt the Model-View-Controller (MVC) pattern to structure the Web Console, promoting separation of concerns and maintainability. The system will utilize DocumentDB for flexible storage of experiment statistics and its horizontal scalability. Concurrent execution of experiments will be implemented using Java threads and ExecutorService to optimize resource utilization. WebSocket communication will be established for real-time data exchange, enabling seamless interaction between the frontend and backend. The project will define message formats and outline the structure of Erlang nodes for efficient communication. The system will provide a user-friendly Web Console to initiate and manage FL experiments, as well as centralized access to experiment statistics for easy monitoring and analysis.

Project Key Points

- Adopt the MVC (Model-View-Controller) pattern to structure the Web Console, promoting separation of concerns and maintainability.
- Utilize DocumentDB for flexible storage of experiment statistics and its horizontal scalability.
- 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.
- System with a graphical interface and web console for executing FL experiments.
- Coordination of experiments through FL Director, an Erlang node.
- Real-time analytics and flexible storage of experiment statistics.
- Centralized access to experiment statistics for monitoring and analysis.

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 be able to view other administrators' experiments.
- 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 search for experiments based on configuration and experiment names.
- Users must be able to monitor real-time progress of experiments.

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.
3. **Security:** User authentication and authorization mechanisms must be implemented.
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.

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.

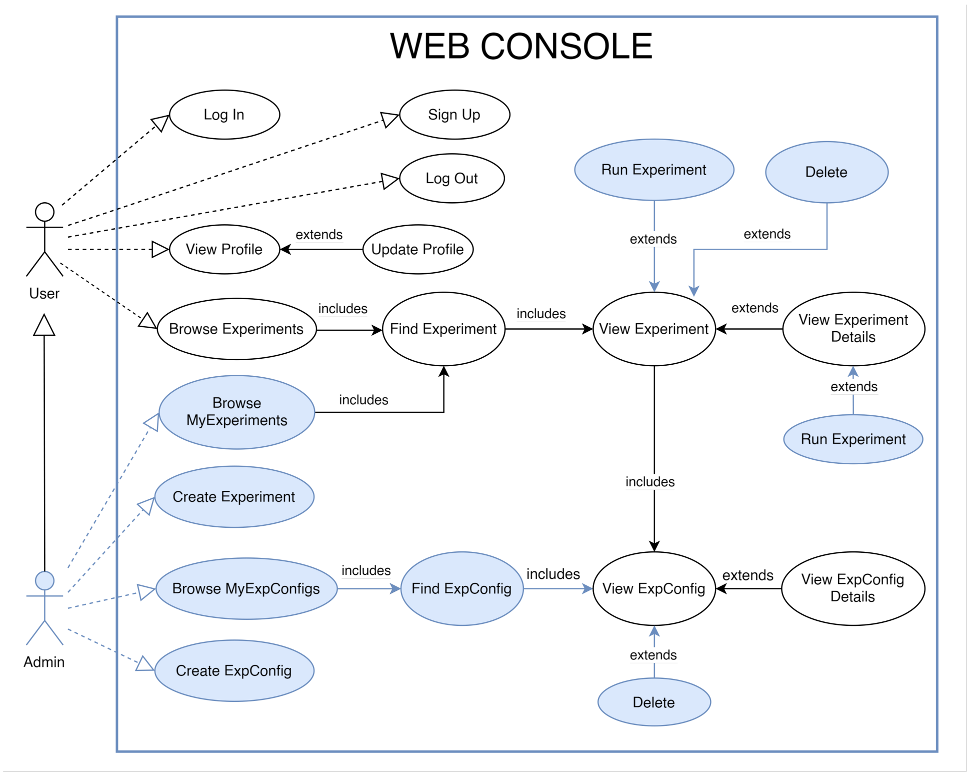


Figure 2.1: Use Case Diagram

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. If the experiment has not started yet 2.1 When the start button is clicked 2.2 The system will send a request to start the experiment 2.3 If the experiment is successfully started, the system will display a success message
Post-Conditions	The experiment statistics are shown on the experiment details page and saved in the database.
Correlated Use cases	
Alternative event steps	–

Table 2.4: Use Case: View Experiment Progress

Use Case	**View Experiment Progress**
Primary Actor	User, Admin
Secondary Actor	–
Description	Allows the admin to view a specific experiment progress
Pre-Conditions	Actor must be logged in and have admin privileges
Main event steps	1. The actor selects the experiment and reaches the details page. 2. If the experiment is started 2.1 Retrieve the stored progresses from the database 2.2 Display the progress on the experiment details page 3. If the experiment is running 3.1 Connect to the websocket 3.2 Subscribe to the experiment progress topic 3.3 While the experiment is running, display the progress in real-time each time a new message is received 3.4 Unsubscribe from the topic when the experiment is finished
Post-Conditions	The experiment progress is displayed on the experiment details page.
Correlated Use cases	
Alternative event steps	–

Analysis Class Diagram

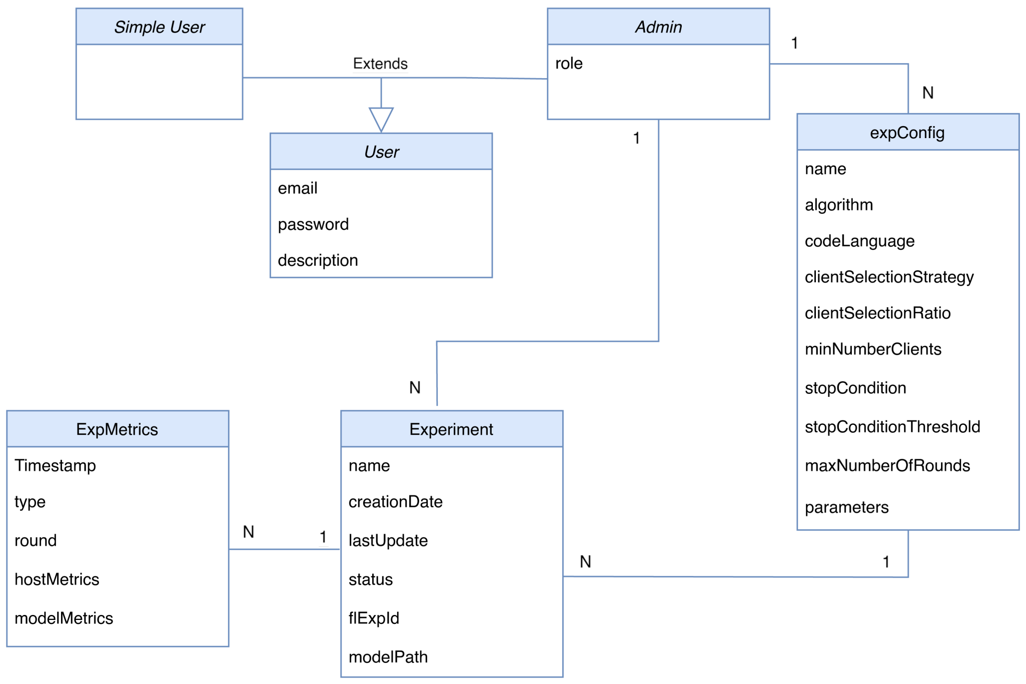


Figure 2.2: Class Diagram

Sequence Diagrams

The following sequence diagrams illustrate the interactions between the actors and the system for the key use cases described in the previous section: the request to start an experiment and the request to view the progress of an experiment.

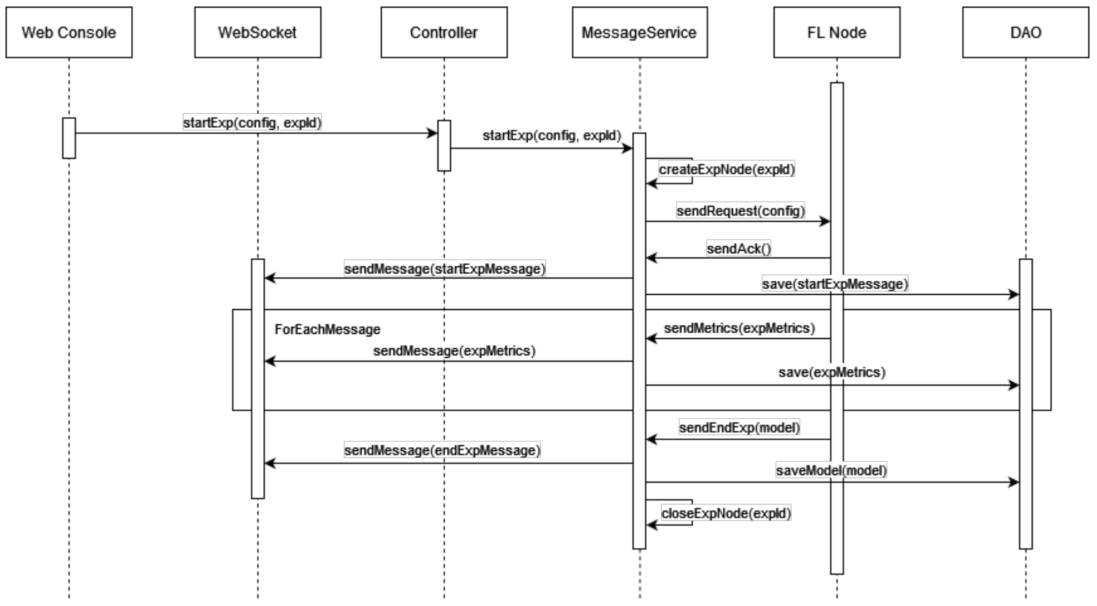


Figure 2.3: Sequence Diagram - Start Experiment

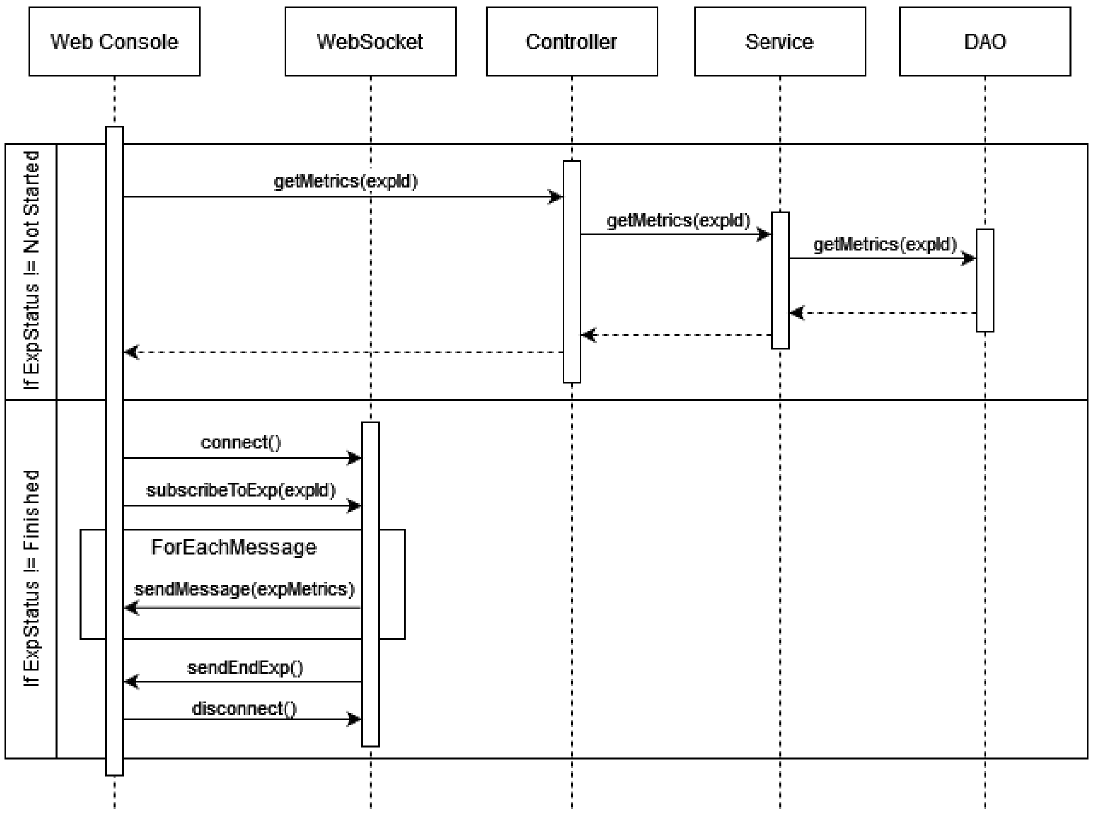


Figure 2.4: Sequence Diagram - View Progress

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

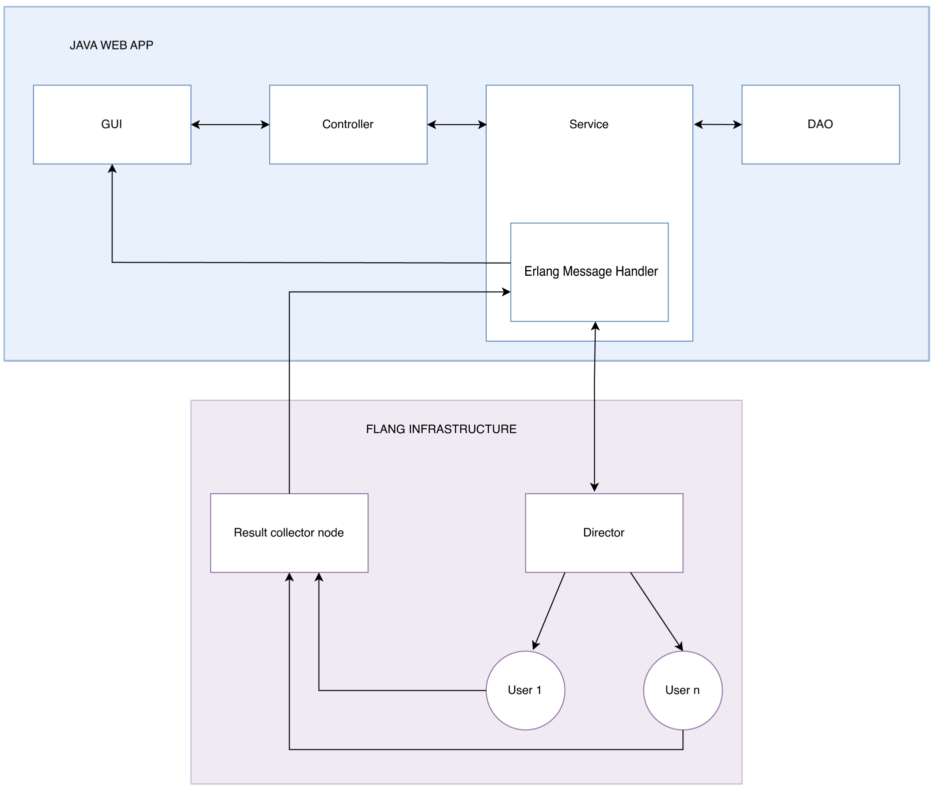


Figure 3.1: System Architecture

Database Design

MongoDB

Collections

ExpConfig document example:

```
{
    "_id": {
      "$oid": "6613f8b7aed2e52b006dea10"
    },
    "name": "TestConfig",
    "algorithm": "fcmeans",
    "codeLanguage": "python",
    "clientSelectionStrategy": "probability",
    "clientSelectionRatio": 1,
    "minNumberClients": 2,
    "stopCondition": "max_number_rounds",
    "stopConditionThreshold": 5,
    "maxNumberOfRounds": 10,
    "parameters": {
      "targetFeature": "16",
      "lambdaFactor": "2",
      "numFeatures": "16",
      "seed": "10",
      "numClusters": "10"
    },
    "creationDate": {
      "$date": "2024-04-08T14:01:27.232Z"
    },
  }
```

Experiment document example:

```
{
    "_id": {
      "$oid": "661c3d780bb4be3bd9b891b9"
    },
    "name": "ExpTest",
    "expConfig": {
      "_id": {
        "$oid": "6613f8b7aed2e52b006dea10"
      },
      "name": "TestConfig",
      "algorithm": "fcmeans"
    },
    "creationDate": {
      "$date": "2024-04-14T20:32:56.022Z"
    },
    "status": "FINISHED",
    "flExpId": "\"d9d1bc7c-d733-4219-b4fb-16a3849db323\"",
    "modelPath": "\\FL_models\\exp_661c3d780bb4be3bd9b891b9.bin"
  }
```

ExperimentMetrics document example:

```
{
    "_id": {
      "$oid": "66144b5337a2fd7f67582f67"
    },
    "expId": "661c3d780bb4be3bd9b891b9",
    "type": "STRATEGY_SERVER_METRICS",
    "hostMetrics": {
      "cpuUsagePercentage": 5,
      "memoryUsagePercentage": 9.27
    },
    "modelMetrics": {
      "FRO": 845.7339394664009
    },
    "timestamp": {
      "$date": "1970-01-20T19:43:26.034Z"
    },
    "round": 1,
  }
```

User document example:

```
{
    "_id": {
      "$oid": "6611252030f96a50aebda458"
    },
    "email": "admin@example.com",
    "password": "P@ssw0rd",
    "creationDate": {
      "$date": "2024-04-06T10:34:08.669Z"
    },
    "role": "admin",
    "configurations": [
      "6613f8b7aed2e52b006dea10"
    ],
    "experiments": [{
        "_id": {
          "$oid": "661c3e800bb4be3bd9b891da"
        },
        "name": "ExpTest",
        "config": "TestConfig",
        "creationDate": {
          "$date": "2024-04-14T20:32:56.022Z"
        }
      }]
  }
```

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

Almost all the messages sent from the FLang Infrastructure to the Erlang message handler are tuples containing one atom to specify the message type and a string for the message body in Json format. The json string contains the information about the message, such as the timestamp, the round number, the client id, and the metrics. The following are some examples of message bodies in json format:

- Experiment Queued:

```
{
    "type": "experiment_queued",
    "timestamp": "1712206255"
}
```

- Worker Ready:

```
{
    "type": "worker_ready",
    "timestamp": "1712206257",
    "client_id": "1"
}
```

- Strategy Server Ready:

```
{
    "type": "strategy_server_ready",
    "timestamp": "1712206257"
}
```

- All Workers Ready:

```
{
    "type": "all_workers_ready",
    "timestamp": "1712206257"
}
```

- Start Round:

```
{
    "type": "start_round",
    "timestamp": "1712206257",
    "round": "1"
}
```

- Woker Metrics:

```
{
  "type": "woker_metrics",
  "timestamp": "1712206258",
  "round": "1"
  "client_id": "1"
  "hostMetrics": {
    "cpuUsagePercentage":39.4,
    "memoryUsagePercentage": 52.69
  }
  "modelMetrics":{
    "ARI":"0.10764575464353877"
  }
}
```

- End Round:

```
{
  "type": "end_round",
  "timestamp": "1712206260",
  "round": "1"
}
```

- Strategy Server Metrics:

```
{
  "type": "strategy_server_metrics",
  "timestamp": "1712206264",
  "round": "2",
  "hostMetrics": {
    "cpuUsagePercentage":39.4,
    "memoryUsagePercentage": 52.69
  }
   "modelMetrics":{
    "FRO":"0.13237183370436725"
  }
}
```

-

The trained model is sent at the end of the experiment in a special message that consists of a tuple with arity 3, where the first element is the message type's atom "fl_end_str_run", the second element is the identifier used by the FLang Infrastructure to identify the experiment, and the third element is the model in binary format.

The message structure is designed to be flexible and extensible, allowing for easy integration of new message types and additional information as needed. This design choice enables the system to adapt to changing requirements and accommodate future enhancements without significant modifications to the existing codebase.

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:** MongoDB is chosen as a database to ensure storing data for experiment statistics and user information.
- **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, MetricsDao, UserDao the applications guarantee effective operations, management of experiments and tracking of the progress.

Data Transfer

Data Transfer layer contains a ExpConfigSummary, ExperimentSummary and UserSummary classes to ensure the functionality of transferring data structure between different layers and components of the application. With the help of the DTO classes, related information 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
- Message Service is implemented for managing communication with Erlang node for sending experiment configuration and monitoring the progress.
- Metrics Service is created to handle operations of retrieving experiment metrics from the database with the related experiment ID.

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.

System Configuration

MongoDB

To add each replica to the replica set, the initiation of a MongoDB instance is required using the command:

```
mongod --replSet SETNAME --dbpath PATH --port PORT --bind_ip localhost,IP
--oplogSize 200
```

This procedure has been executed across all provided clusters. For instance:

```
mongod --replSet DSMT --dbpath ~/data --port 27017 --bind_ip localhost,10.2.1.109
--oplogSize 200
```

Once all clusters are online, connecting to the primary is necessary, for example, using:

```
mongosh --host 10.2.1.109 --port 27017
```

Subsequently, providing the replica set configuration is required:

```
rsconf = {_id: "dsmt", members: [{_id: 0, host: "10.2.1.103", priority:1},
{_id: 1, host: "10.2.1.106", priority:2},{_id: 3, host: "10.2.1.107", priority:3},
{_id: 4, host: "10.2.1.108", priority:4},{_id: 2, host: "10.2.1.109", priority:5}]};
```

Finally, initializing the replica set using the command:

```
rs.initiate()
```

Upon completion of these operations, the replicas will be operational, and connections can be made.

Tomcat

The initial step involves copying the .war file to the webapps directory of Tomcat. This can be achieved via scp:

```
scp MY_LOCAL_PATH root@REMOTE_IP:/REMOTE_PATH/apache-tomcat-10.1.16/webapps/FILENAME.war
```

To start the Tomcat server, navigating to the installation directory and executing the command is necessary:

```
bin/startup.sh
```

For example:

```
./servers/apache-tomcat-10.1.16/bin/startup.sh
```

Nginx

After downloading Nginx, configuring the configuration file using the nano editor is required:

```
nano /etc/nginx/nginx.conf
```

Within the file, adding the following directives to route incoming traffic on port 80 to Tomcat on port 8080 is necessary:

```
server {
        listen 80;
        server_name 10.2.1.102;

        location / {
            proxy_pass http://localhost:8080;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
        }
    }
```

Subsequently, restarting Nginx using the command:

```
sudo systemctl restart nginx
```

After this operation, the system will be operational.

EPMD Erlang

To start the EPMD service, executing the command is sufficient:

```
epmd
```

Using the FL Web Console Application

Once the above configurations are completed, the FL Web Console application is ready for use.

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

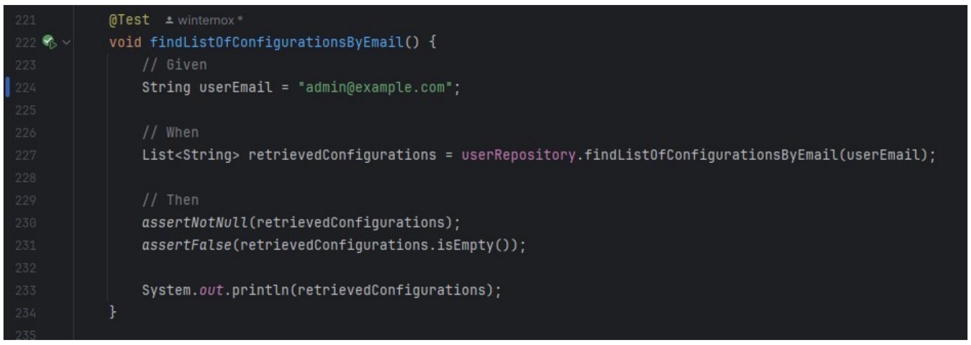
Structural testing, also known as white-box testing is applied to the project to ensure that the implemented code is working as expected and evaluate the internal structure of the system. For primary structure testing JUnit testing is applied as a testing methodology.

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:

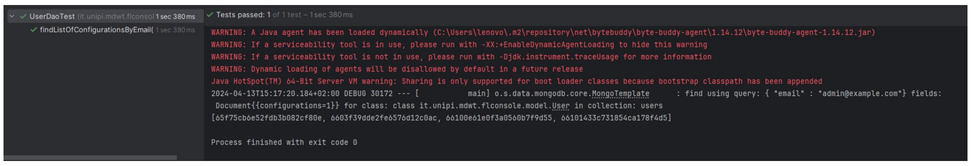
UserDAO

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.



```java
@Test  winternox *
void findListOfConfigurationsByEmail() {
    // Given
    String userEmail = "admin@example.com";

    // When
    List<String> retrievedConfigurations = userRepository.findListOfConfigurationsByEmail(userEmail);

    // Then
    assertNotNull(retrievedConfigurations);
    assertFalse(retrievedConfigurations.isEmpty());

    System.out.println(retrievedConfigurations);
}
```

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



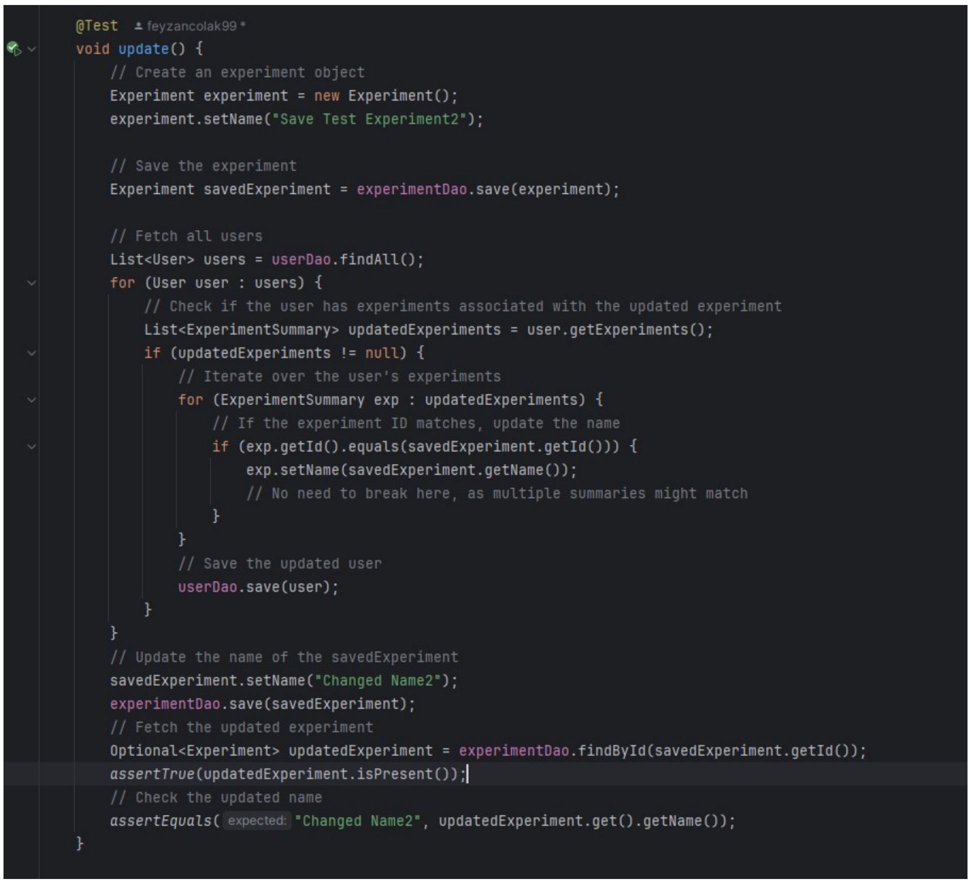
```
UserDaoTest (it.unipi.mdwt.flconsol 1 sec 380 ms
  findListOfConfigurationsByEmail( 1 sec 380 ms

Tests passed: 1 of 1 test – 1 sec 380 ms
WARNING: A Java agent has been loaded dynamically (C:\Users\lenovo\.m2\repository\net\bytebuddy\byte-buddy-agent\1.14.12\byte-buddy-agent-1.14.12.jar)
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
2024-04-13T15:17:20.184+02:00 DEBUG 30172 --- [           main] o.s.data.mongodb.core.MongoTemplate      : find using query: { "email" : "admin@example.com"} fields:
 Document{{configurations=1}} for class: class it.unipi.mdwt.flconsole.model.User in collection: users
[65f75cb6e52fdb3b082cf80e, 6603f39dde2fe6576d12c0ac, 66100e61e0f3a0560b7f9d55, 66101433c731854ca178f4d5]

Process finished with exit code 0
```

Figure 5.2: Testing result for 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.

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.

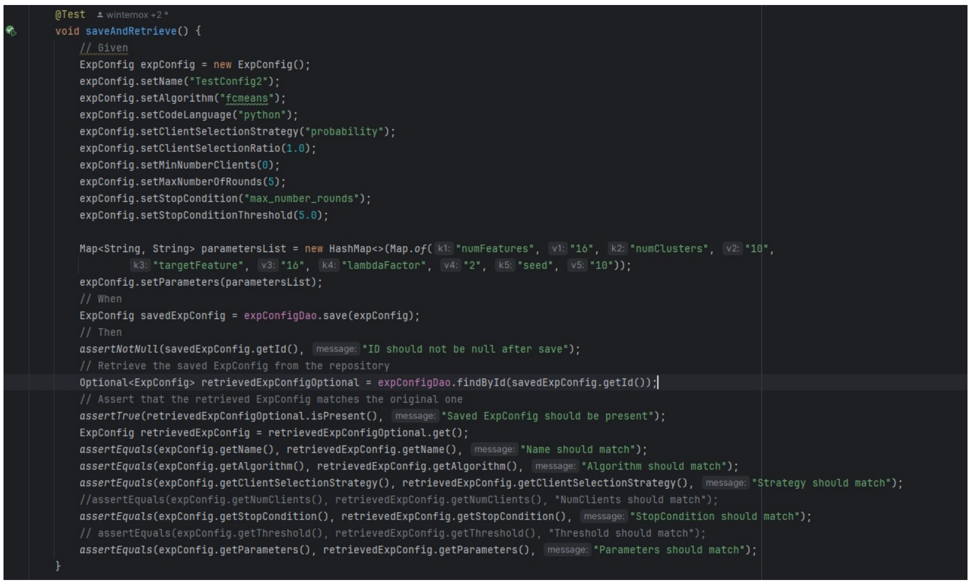


```java
@Test  feyzancolak99 *
void update() {
    // Create an experiment object
    Experiment experiment = new Experiment();
    experiment.setName("Save Test Experiment2");

    // 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

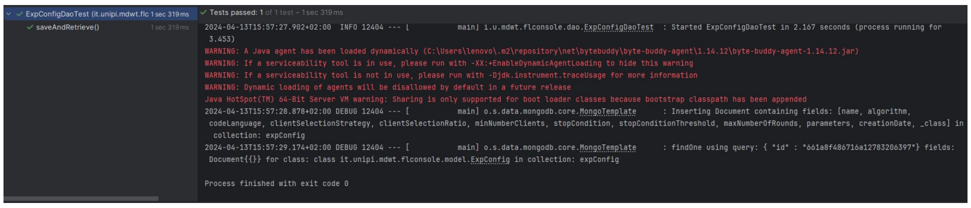
ConfigurationDAO

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.



```java
@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



```
✓ ExpConfigDaoTest (it.unipi.mdwt.flc 1 sec 319 ms
  ✓ saveAndRetrieve()                1 sec 319 ms

✓ Tests passed: 1 of 1 test – 1 sec 319 ms
2024-04-13T15:57:27.902+02:00  INFO 12404 --- [           main] i.u.mdwt.flconsole.dao.ExpConfigDaoTest  : Started ExpConfigDaoTest in 2.167 seconds (process running for
 3.453)
WARNING: A Java agent has been loaded dynamically (C:\Users\lenovo\.m2\repository\net\bytebuddy\byte-buddy-agent\1.14.12\byte-buddy-agent-1.14.12.jar)
WARNING: If a serviceability tool is in use, please run with -XX:+EnableDynamicAgentLoading to hide this warning
WARNING: If a serviceability tool is not in use, please run with -Djdk.instrument.traceUsage for more information
WARNING: Dynamic loading of agents will be disallowed by default in a future release
Java HotSpot(TM) 64-Bit Server VM warning: Sharing is only supported for boot loader classes because bootstrap classpath has been appended
2024-04-13T15:57:28.878+02:00 DEBUG 12404 --- [           main] o.s.data.mongodb.core.MongoTemplate      : Inserting Document containing fields: [name, algorithm,
 codeLanguage, clientSelectionStrategy, clientSelectionRatio, minNumberClients, stopCondition, stopConditionThreshold, maxNumberOfRounds, parameters, creationDate, _class] in
  collection: expConfig
2024-04-13T15:57:29.174+02:00 DEBUG 12404 --- [           main] o.s.data.mongodb.core.MongoTemplate      : findOne using query: { "id" : "661a8f486716a12783206397"} fields:
 Document{{}} for class: class it.unipi.mdwt.flconsole.model.ExpConfig in collection: expConfig

Process finished with exit code 0
```

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
A_T_07	Search Configuration by name	Write name with existing configuration name	Show the list of the configuration with that name	List of configurations with that name	Passed
A_T_08	Search experiment by name	Write name with existing experiment name	Show the list of the experiments with that name	List of experiments with that name	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_03	User Signup	Email: new@example.com Password: P@ssw0rd (valid input)	Sign up successfully	Redirected to user dashboard	Passed
U_T_04	User Signup 2	Email: new@example.com Password: invalid (invalid password)	Error message displayed	Unable to Signup	Passed
U_T_05	Search Configuration by name	Write name with existing configuration name	Show the list of the configuration with that name	List of configurations with that name	Passed
U_T_06	Search experiment by name	Write name with existing experiment name	Show the list of the experiments with that name	List of experiments with that name	Passed

Conclusion

In this project Federated Learning Web Console is introduced which is a decentralized platform for managing FL experiments with using Java and Erlang as primary programming languages, Spring as a Java framework and MongoDB as a database. The primary goal of this project is to implement and provide robust system that can coordinate FL experiment with machine learning approach and being in a collaboration with multiple devices while data locality is retained.

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. MongoDB is chosen as a document DB database to have scalable and adaptive database.

With the help of this architecture FL Web Console project ensures a scalable design and having seamless functionality between different components. The user-centric 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.