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 Goals . . . 2
- 1.3 Communication Protocol . . . 2
- 1.4 Data Variety . . . 2
- 1.5 Concurrent Experiment Execution . . . 2
- 1.6 Real-Time Analytics . . . 3
- 1.7 Project Key Points . . . 3
- 1.8 Architecture and Frameworks . . . 3

2 Analysis **4**
- 2.1 Requirements . . . 4
- 2.2 Use Case Diagram . . . 6
- 2.3 Analysis Class Diagram . . . 8
- 2.4 Sequence Diagrams . . . 9

3 Design **10**
- 3.1 Introduction . . . 10
- 3.2 Software Architecture . . . 10
- 3.3 Database Design . . . 11
- 3.4 Message Handler . . . 13

4 Implementation **14**
- 4.1 Development Environment . . . 14
- 4.2 Main Modules . . . 14
- 4.3 Configuration . . . 14
- 4.4 Data Access . . . 14
- 4.5 Data Transfer . . . 14
- 4.6 Service . . . 14
- 4.7 User Interface . . . 14
- 4.8 Adopted Patterns and Techniques . . . 14

5 Testing **15**
- 5.1 Structural Testing . . . 15
- 5.2 Functional Testing . . . 18

6 Conclusion **19**

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.

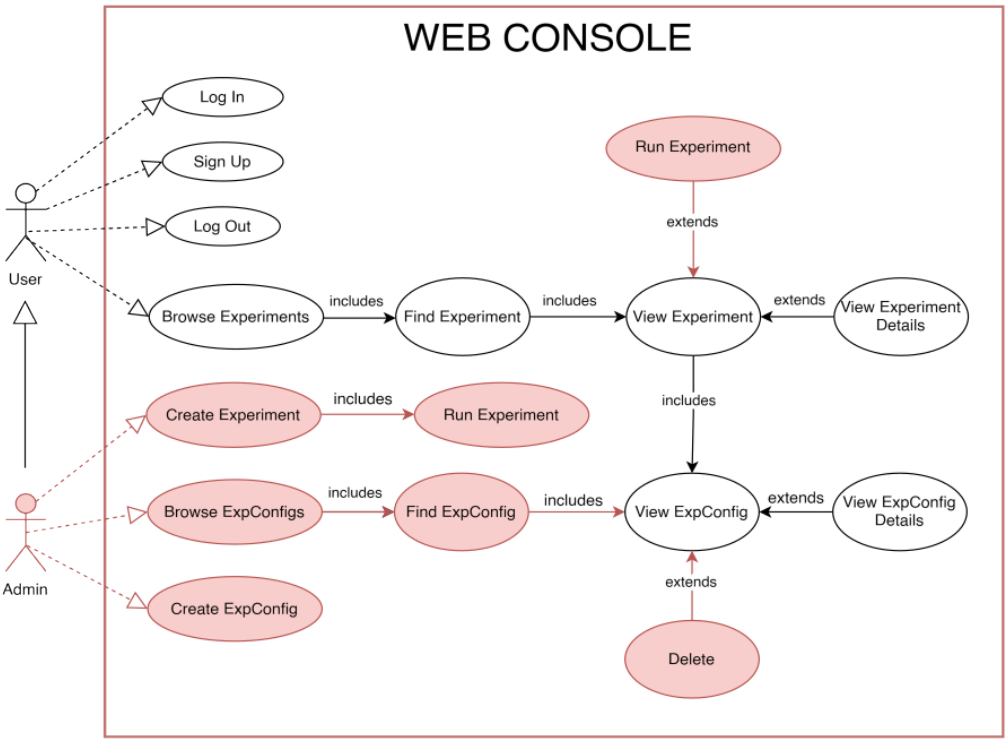

Figure 2.1: Use Case Diagram

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

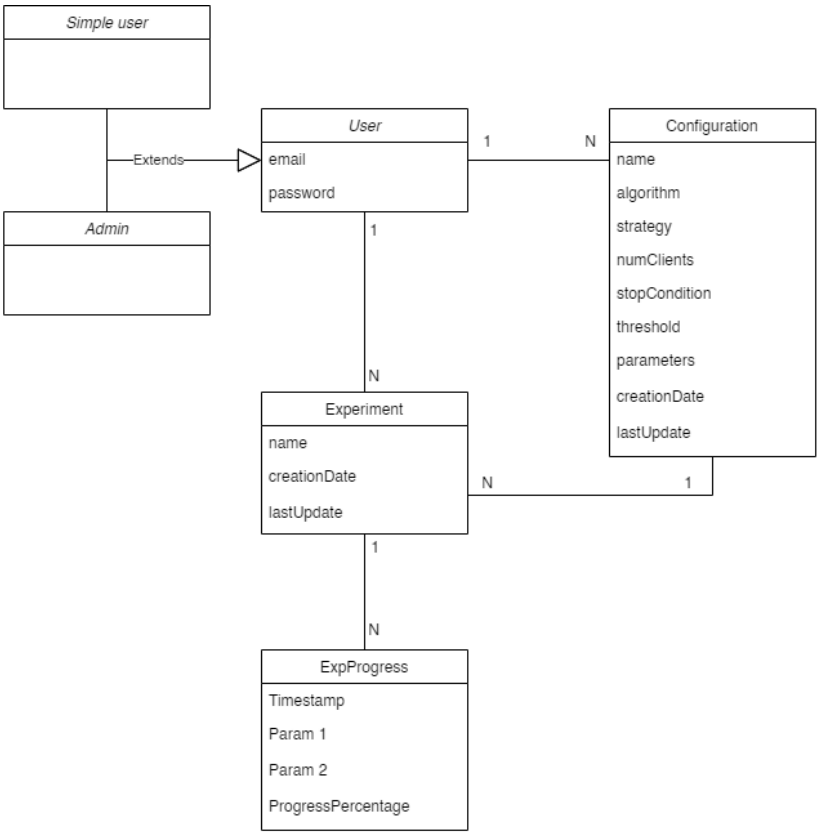

Figure 2.2: Class Diagram

Sequence Diagrams

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

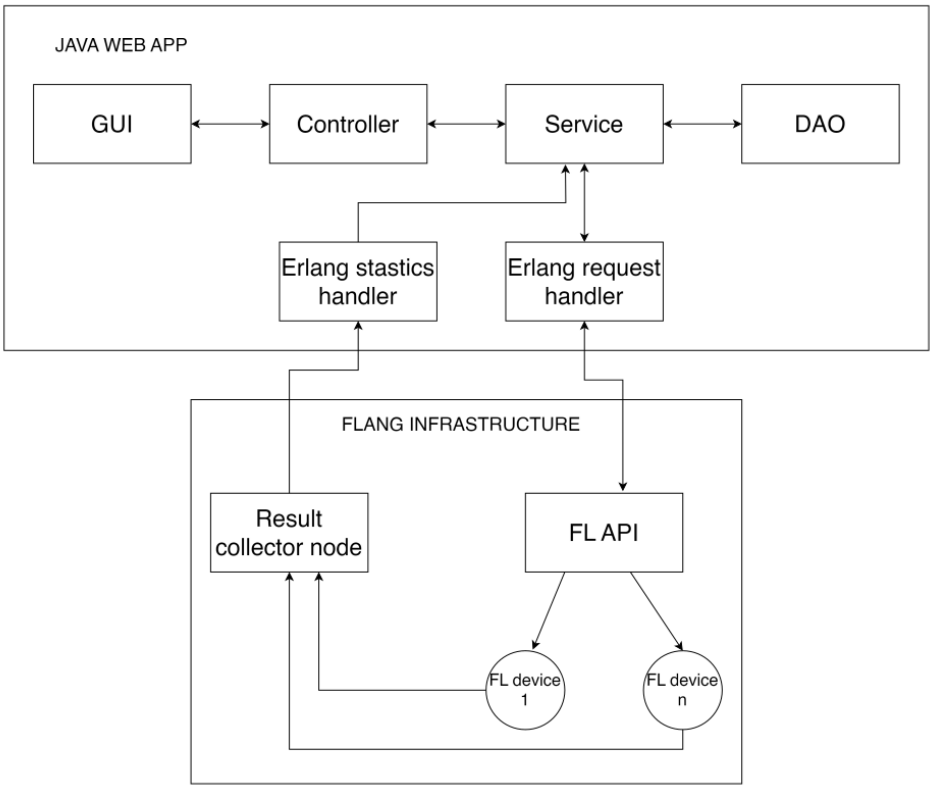

Figure 3.1: System Architecture

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

The development environment used for the project is described in this section.

Main Modules

The main modules of the project are described in this section.

Configuration

The configuration of the project is described in this section.

Data Access

The data access layer of the project is described in this section.

Data Transfer

The data transfer mechanisms used in the project are described in this section.

Service

The services provided by the project are described in this section.

User Interface

The user interface of the project is described in this section.

Adopted Patterns and Techniques

The patterns and techniques adopted in the project are described in this section.

Testing

Structural Testing

The structural testing performed on the project is described in this section.

JUnit Testing

The JUnit testing performed on the project is described in this section.

UserDAO

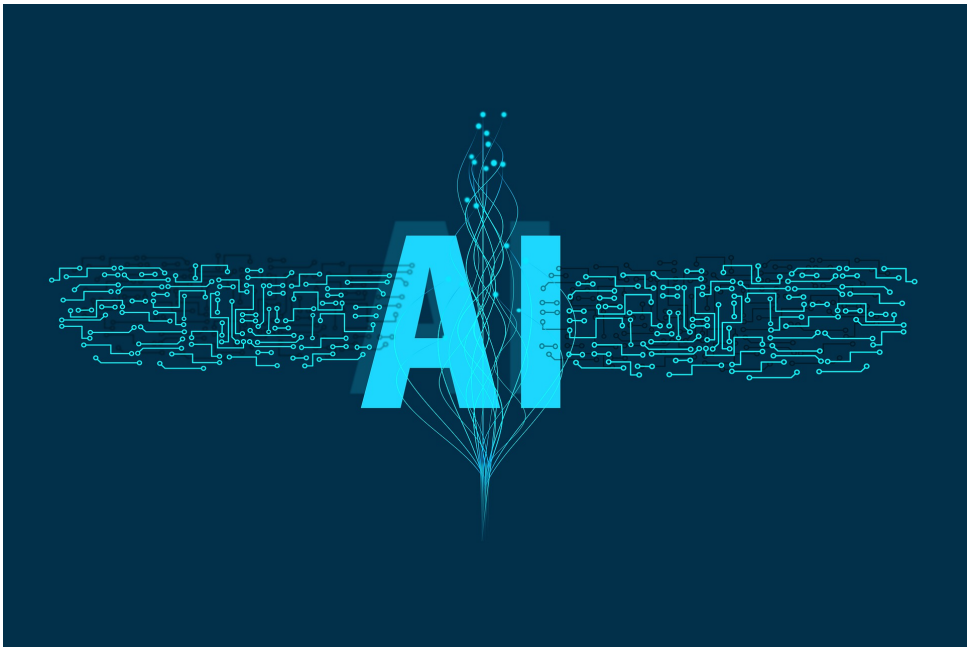

Figure 5.1: User DAO tests

ConfigurationDAO

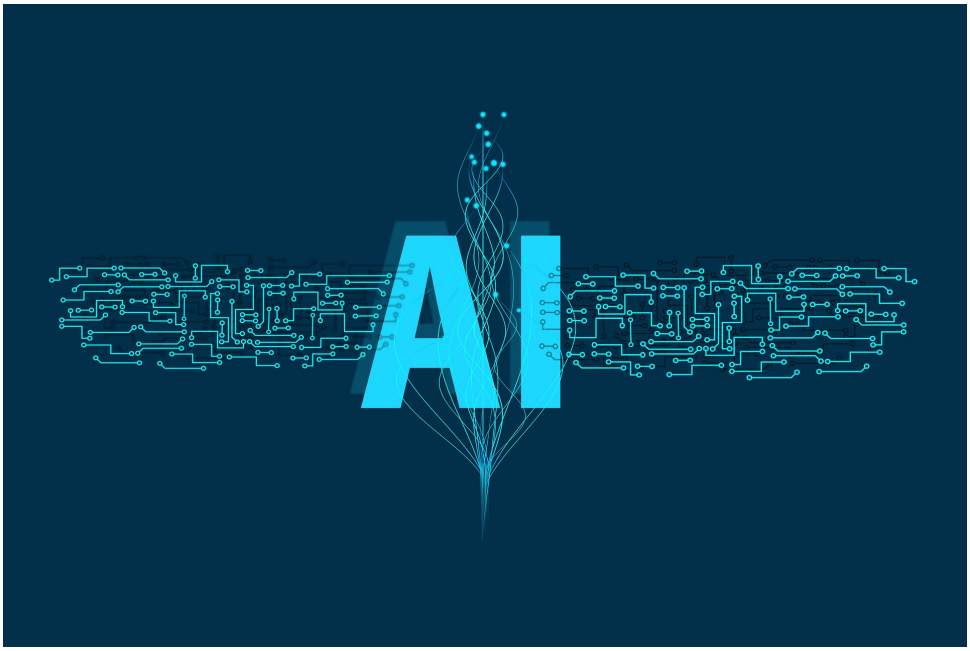

Figure 5.2: Configuration DAO tests

ExperimentDAO

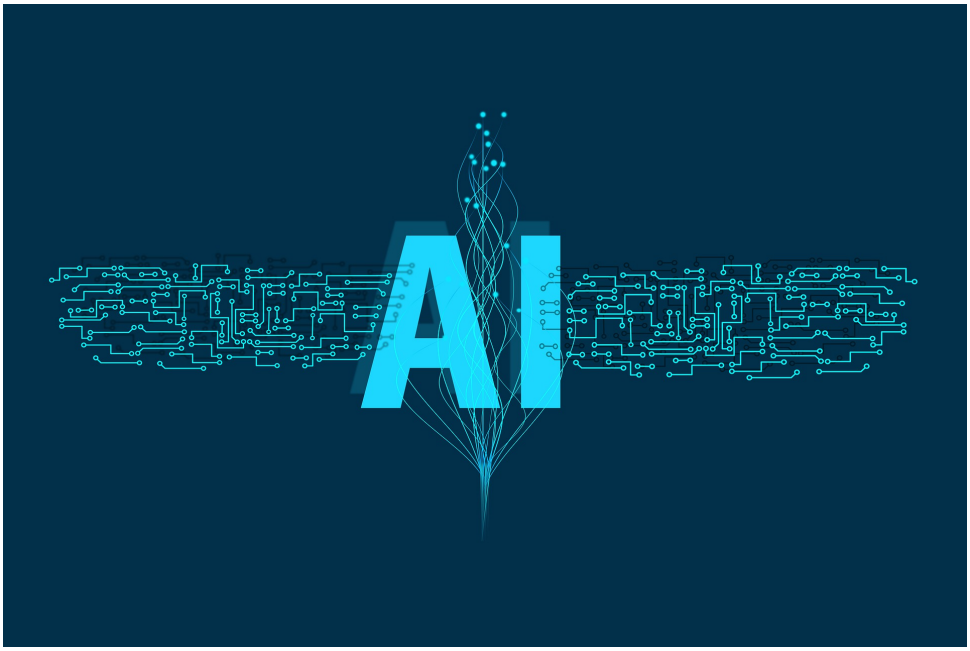

Figure 5.3: Experiment DAO tests

Functional Testing

The functional testing performed on the project is described in this section.

Test Cases

Table 5.1: Test case

Id	**Description**	**Input**	**E. Output**	**Output**	**Outcome**
U_T_01	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum
U_T_02	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum
U_T_03	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum
U_T_04	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum
U_T_05	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum	Lorem Ipsum

Conclusion

In this chapter, we summarize the key points of the document and discuss possible future directions for the project.