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 comprehensive system for managing 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 robust platform for orchestrating FL experiments, enabling users to monitor their progress, analyze results, and facilitate communication between the Java Web Application and FL Director.

Project Goals

Manage Message Passing

- Implement a communication system between the Java Web Application and FL Director.
- 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 robust protocol will be defined. This protocol will facilitate the exchange of messages and data in a structured format, ensuring consistency and compatibility across different components of the system.

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 leveraged 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.

System Architecture

The system architecture will consist of several key components, including:

Result Collector

Responsible for collecting experiment statistics from participating devices and forwarding them to the Web Console for display and analysis.

Web Console

A user interface component that allows users to interact with the system, initiate new experiments, and monitor their progress in real-time.

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.

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.

Use Case Modeling

Use Case Diagram

Scenarios

Analysis Class Diagram

The analysis class diagram of the project is presented in this subsection.

Sequence Diagrams

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

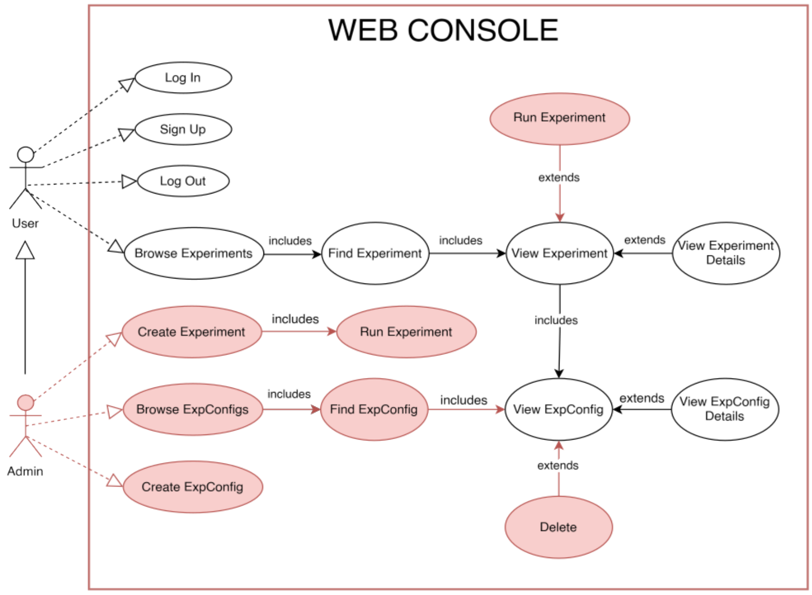

Figure 2.1: Use Case Diagram

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 name 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	–

Design

Software Architecture

The software architecture of the project is described in this section.

Database Design

In this section, the database design of the project is presented.

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",
    "progressList": [
        {
        "creationDate": "2024-03-14T06:00:00Z",
        "parameters": {
            "param1": "value1",
            "param2": "value2"
        },
        "status": "In progress"
        },
        {
        "creationDate": "2024-03-14T09:00:00Z",
        "parameters": {
            "param1": "value1",
            "param2": "value2",
            "param3": "value3"
        },
        "status": "Completed"
        }
    ]
}
```

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"
}
```

Erlang Message Handler

The Erlang message handler design is described in this subsection.

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"
}
```

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.

Functional Testing

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

Test Cases

The test cases used for the project are presented in this section.

Conclusion

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