23.10.2024

Assignment Task: **Machine Learning Model Management Application**

Objective

Develop a machine learning model management application that allows users to create, view, and manage machine learning models through a web interface. The application should include both frontend and backend components, demonstrating skills in full-stack development using Java and either React or Angular.

Requirements Overview

The application should consist of three main UI components:

1. **Model Listing Page**: Displays all existing machine learning models form the backend
2. **Model Creating Page**: Allows users to create a new model, specifying its name and selecting layers (For simplicity: provide a multi select options of those values: Text Classifier, Visual Classifier, Optical Recognizer).
3. **Model Detail Page**: Provides detailed information about a specific model, including options for importing training data, starting training, and viewing training results.

Task Breakdown

Frontend (React or Angular)

1. **Model Listing Page**
    - Display a table or list showing:
        - **Model Name**
        - **Status** (e.g., Trained, Not Trained, Training)
    - When a model is clicked, navigate to the **Model Detail Page** to display more information about the selected model.
2. **Model Creating Page**
    - Include a form with:
        - **Model Name**: A text input field for the name of the model.
        - **Layers**: A multi-selection dropdown for the available model layers:
            - Text Classifier
            - Visual Classifier
            - Optical Recognizer
    - Upon submission, the model should be created and stored in the backend.
3. **Model Detail Page**
    - Display detailed information about the selected model.
    - Include:

        - An option to **Import Training Data** (a file upload field).
        - A button to **Start Training** the model.
        - If the model is already trained, display the training results:
            - **Training Duration**
            - **Accuracy Percentage**

Backend (Java - Spring Boot)

1. **API Endpoints**
    - Implement a RESTful API using Spring Boot to support frontend interactions. The backend should support the following endpoints:
        - **POST /models**: Create a new model.
        - **GET /models**: Retrieve a list of all models.
        - **GET /models/{id}**: Retrieve detailed information for a specific model by ID.
        - **POST /models/{id}/train**: Start the training process for a model.
        - **POST /models/{id}/data**: Upload training data for a model. For simplicity the data can be any text in string format
        - **GET /models/{id}/status**: Retrieve the training status of a specific model.
2. **Model Class (Example)**
    - Define a Model class with the following properties:
        - UUID id: Unique identifier for the model.
        - String name: Name of the model.
        - List<String> layers: The selected layers (e.g., Text Classifier, Visual Classifier).
        - String status: Status of the model (Not Trained, Being Trained, Trained).
        - Map<String, Object> trainingResults: Stores training details such as duration, and accuracy.
3. **Database Integration**
    - Use **H2 Database** (or any embedded database) for storing model data.
    - The schema should include fields for model information, status, and training results.
4. **Training Logic (Asynchronous)**
    - Simulate the training process using a random duration between 30 seconds and 180 seconds to imitate a realistic, time-consuming task.
    - If a training job is already running, the API should respond with a message: "Training server is busy, try again later."

   - Update the model's status and store training results (duration, accuracy) upon completion.

5. **Service and Repository Implementation**
   - Use Spring's **Service** and **Repository** patterns to handle business logic and data persistence.
   - **Repository Layer**: Use Spring Data JPA to interact with the H2 Database.
   - **Service Layer**: Implement logic for model creation, status updates, training, and managing asynchronous tasks.

Non-Functional Requirements

- **Error Handling**: Ensure proper validation and error handling for each API endpoint.
- **Logging**: Implement basic logging for API requests and training processes.
- **Testing**: Write unit tests using JUnit for service methods
- **Documentation**: Use Swagger (OpenAPI) for API documentation.

Guidelines for Completion

- **Deployment**: Provide instructions for running the application locally, including setup steps for both frontend and backend components. You can use Docker Compose if preferred.
- **Code Quality**: Ensure clean, readable code with meaningful commit messages. Maintain a structured and organized project layout.
- **GitHub Repository**: Upload the solution to your public GitHub repository and share the link.

Bonus Tasks

- Implement **JWT-based Authentication** for API access.

Evaluation Criteria

- Adherence to RESTful best practices
- Consistent code quality across both frontend and backend.
- Clean, functional, and basic UI that adheres to the requirements.
- Proper use of Java Spring Boot features and a clear understanding of backend architecture.
- Usage of Git and GitHub for version control, including clear documentation.

This assignment is designed to showcase your full-stack development skills in a realistic setting similar to production environments. Good luck, and we look forward to reviewing your solution!