```
package com.internlink.internlink.model;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import ai.djl.ModelException;
import ai.djl.inference.Predictor;
import ai.djl.repository.zoo.Criteria;
import ai.djl.repository.zoo.ModelZoo;
import ai.djl.repository.zoo.ZooModel;
import ai.djl.translate.TranslateException;

public class BertModel {
   private final ZooModel<String, float[]> model;
   private final Predictor<String, float[]> predictor;

   public BertModel() throws ModelException, IOException {
      model = ModelZoo.loadModel(
            Criteria.builder()
                  .setTypes(String.class, float[].class)
                  .optModelUrls("djl://ai.djl.huggingface.pytorch/sentence-
transformers/all-MiniLM-L6-v2")
                  .optTranslator(new BertTranslator())
                  .build());
```

```
        predictor = model.newPredictor();
    }

    public List<Float> getEmbedding(String text) throws TranslateException {
        float[] embedding = predictor.predict(text);

        List<Float> embeddingList = new ArrayList<>();
        for (float value : embedding) {
            embeddingList.add(value);
        }
        return embeddingList;
    }
}
package com.internlink.internlink.model;

import ai.djl.huggingface.tokenizers.Encoding;
import ai.djl.huggingface.tokenizers.HuggingFaceTokenizer;
import ai.djl.ndarray.NDArray;
import ai.djl.ndarray.NDList;
import ai.djl.translate.Batchifier;
import ai.djl.translate.Translator;
import ai.djl.translate.TranslatorContext;

public class BertTranslator implements Translator<String, float[]> {
    private HuggingFaceTokenizer tokenizer;
```

```
    public BertTranslator() {
        tokenizer = HuggingFaceTokenizer.newInstance("bert-base-uncased");
    }

    @Override
    public NDList processInput(TranslatorContext ctx, String input) {
        Encoding encoding = tokenizer.encode(input);
        NDArray inputIds = ctx.getNDManager().create(encoding.getIds()).reshape(1, -
1);
        NDArray attentionMask =
ctx.getNDManager().create(encoding.getAttentionMask()).reshape(1, -1);

        return new NDList(inputIds, attentionMask);
    }

    @Override
    public float[] processOutput(TranslatorContext ctx, NDList list) {
        System.out.println("Entering processOutput method");
        System.out.println("NDList size: " + list.size());

        NDArray output = list.get(0); // First element contains the embeddings
        System.out.println("First output shape: " + output.getShape());

        // Ensure the output is (1, 384) as expected
        float[] embeddings = output.squeeze().toFloatArray(); // Flatten the array
properly
        System.out.println("Final embedding size: " + embeddings.length);
```

```
        if (embeddings.length != 384) {
            throw new IllegalStateException(
                    "Unexpected model output shape! Expected 384, but got: " +
embeddings.length);
        }

        return embeddings;
    }

    @Override
    public Batchifier getBatchifier() {
        return null;
    }
}
```

```
package com.internlink.internlink.service;

import java.util.List;

import org.springframework.stereotype.Service;

import com.internlink.internlink.model.BertModel;

import ai.djl.translate.TranslateException;
```

```
@Service
public class EmbeddingService {

    private final BertModel bertModel;
    private static final int EXPECTED_EMBEDDING_SIZE = 384;

    public EmbeddingService(BertModel bertModel) {
        this.bertModel = bertModel;
    }

    public List<Float> generateEmbedding(String text) throws TranslateException {
        List<Float> embedding = bertModel.getEmbedding(text);

        if (embedding.size() != EXPECTED_EMBEDDING_SIZE) {
            throw new IllegalStateException("Embedding size mismatch! Expected: " +
EXPECTED_EMBEDDING_SIZE +
                    ", but got: " + embedding.size());
        }
        return embedding;
    }
}
package com.internlink.internlink.service;

import java.util.ArrayList;
import java.util.List;
```

```
import org.bson.Document;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.stereotype.Service;

import com.internlink.internlink.model.Internship;
import com.internlink.internlink.model.Student;

@Service
public class InternshipService {

  @Autowired
  private MongoTemplate mongoTemplate;

  public void createInternship(Internship internship) {
    mongoTemplate.save(internship);
  }

  public Internship getInternshipById(String internshipId) {
    return mongoTemplate.findById(internshipId, Internship.class);
  }

  public List<Float> getStudentEmbedding(String studentId) {
    // Log the student ID being queried
    System.out.println("Fetching embedding for student ID: " + studentId);
```

```
    // Fetch the student from the database
    Student student = mongoTemplate.findById(studentId, Student.class);
    if (student == null) {
        // Log if the student is not found
        System.err.println("Student not found with ID: " + studentId);
        throw new IllegalStateException("Student embedding not found!");
    }

    // Check if the embedding exists
    List<Float> embedding = student.getEmbedding();
    if (embedding == null || embedding.isEmpty()) {
        // Log if the embedding is missing or empty
        System.err.println("Embedding not found or empty for student ID: " +
studentId);
        throw new IllegalStateException("Student embedding not found!");
    }

    // Log successful retrieval of the embedding
    System.out.println("Embedding successfully fetched for student ID: " +
studentId);
    return embedding;
  }

  public List<Internship> getRecommendedInternships(List<Float>
studentEmbedding) {
```

```
System.out.println("Generating recommendations using student embedding: " + studentEmbedding);

if (studentEmbedding == null || studentEmbedding.isEmpty()) {
    System.err.println("Invalid student embedding: " + studentEmbedding);
    throw new IllegalArgumentException("Invalid student embedding!");
}

// ✅ Corrected: Add "limit" inside $vectorSearch
Document vectorSearchQuery = new Document("$vectorSearch",
        new Document("index", "internship_index2")
                .append("queryVector", studentEmbedding)
                .append("path", "embedding")
                .append("numCandidates", 10) // Number of candidates considered before ranking
                .append("k", 5) // Top K results to return
                .append("limit", 5) // ✅ FIX: "limit" inside $vectorSearch
);

// Add a $project stage to exclude the "embedding" field
Document projectStage = new Document("$project",
        new Document("embedding", 0) // Exclude the "embedding" field
);

System.out.println("Vector search query: " + vectorSearchQuery.toJson());
```

```
        // Run aggregation with the $project stage
        List<Document> results = mongoTemplate.getCollection("internships")
                .aggregate(List.of(vectorSearchQuery, projectStage)) // Add $project stage
                .into(new ArrayList<>());

        // Convert BSON documents to Internship objects
        List<Internship> internships = results.stream()
                .map(doc -> mongoTemplate.getConverter().read(Internship.class, doc))
                .toList();

        System.out.println("Number of internships found: " + internships.size());
        return internships;
    }
}
package com.internlink.internlink.service;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.data.mongodb.core.query.Criteria;
import org.springframework.data.mongodb.core.query.Query;
import org.springframework.data.mongodb.core.query.Update;
import org.springframework.stereotype.Service;
```

```
import com.internlink.internlink.model.Student;

@Service
public class StudentService {

  @Autowired
  private MongoTemplate mongoTemplate;

  // public List<Student> getAllStudents() {
  // return mongoTemplate.findAll(Student.class);
  // }

  public Student getStudentById(String studentId) {
    Query query = new Query(Criteria.where("studentId").is(studentId));
    return mongoTemplate.findOne(query, Student.class);
  }

  public Student register(Student student) {
    return mongoTemplate.save(student);
  }

  public Student updateStudent(String studentId, Student updatedStudent) {
    Student student = getStudentById(studentId);
    if (student == null)
      return null;
```

```
        student.setName(updatedStudent.getName());
        student.setEmail(updatedStudent.getEmail());
        student.setMajor(updatedStudent.getMajor());
        student.setUniversity(updatedStudent.getUniversity());
        return mongoTemplate.save(student);
    }

    public void deleteStudent(String studentId) {
        Student student = getStudentById(studentId);
        if (student != null) {
            mongoTemplate.remove(student);
        }
    }

    public String getStudentNameById(String studentId) {
        Student student = getStudentById(studentId);
        if (student == null) {
            throw new RuntimeException("Student not found with ID: " + studentId);
        }
        return student.getName();
    }

    public List<Student> getStudentsByFacultySupervisor(String facultySupervisorId)
{
        return mongoTemplate.find(new
Query(Criteria.where("facultySupervisorId").is(facultySupervisorId)),
                Student.class);
```

```
    }

    public List<Student> getStudentsByCompanySupervisor(String
companySupervisorId) {
        return mongoTemplate.find(new
Query(Criteria.where("companySupervisorId").is(companySupervisorId)),
                Student.class);
    }

    public Student assignFacultySupervisor(String studentId, String
facultySupervisorId) {
        Query query = new Query(Criteria.where("studentId").is(studentId));
        Student student = mongoTemplate.findOne(query, Student.class);

        if (student == null) {
            return null; // Student not found
        }

        Update update = new Update().set("facultySupervisorId", facultySupervisorId);
        mongoTemplate.updateFirst(query, update, Student.class);

        return mongoTemplate.findOne(query, Student.class);
    }

    public boolean assignCompanySupervisorToStudents(String supervisorId,
List<String> studentIds) {
        if (studentIds == null || studentIds.isEmpty()) {
```

```
        return false; // No students provided
    }

    Query query = new Query(Criteria.where("studentId").in(studentIds)); // Find students by IDs
    Update update = new Update().set("companySupervisorId", supervisorId); // Set company supervisor

    var result = mongoTemplate.updateMulti(query, update, Student.class); // Apply update

    return result.getModifiedCount() > 0; // Return true if any students were updated
  }

  public String getStudentIdByMongoId(String mongoId) {
    // Query MongoDB for the student using Mongo-generated _id
    Query query = new Query(Criteria.where("_id").is(mongoId));
    Student student = mongoTemplate.findOne(query, Student.class);

    // If student is found, return the custom studentId
    if (student != null) {
        return student.getStudentId(); // Assuming custom studentId field exists
    }

    return null; // Return null if student is not found
  }
```

```
    public boolean existsById(String studentId) {
        Query query = new Query();
        query.addCriteria(Criteria.where("_id").is(studentId));
        return mongoTemplate.exists(query, Student.class);
    }

    public List<Float> getStudentEmbedding(String studentId) {
        Student student = mongoTemplate.findById(studentId, Student.class);
        return (student != null && student.getEmbedding() != null) ?
student.getEmbedding() : new ArrayList<>();
    }

}
```