Add Dependencies

In your pom.xml, include dependencies for DJL and mBERT:

xml

CopyEdit

```
<dependency>
   <groupId>ai.djl</groupId>
   <artifactId>api</artifactId>
   <version>0.15.0</version>
</dependency>
<dependency>
   <groupId>ai.djl</groupId>
   <artifactId>tensorflow-engine</artifactId>
   <version>0.15.0</version>
</dependency>
<dependency>
   <groupId>ai.djl</groupId>
   <artifactId>pytorch-engine</artifactId>
   <version>0.15.0</version>
</dependency>
<dependency>
   <groupId>ai.djl</groupId>
   <artifactId>bert</artifactId>
   <version>0.15.0</version>
</dependency>
```

2-**mBERT Model Loading**

Create a class to load the mBERT model and generate embeddings.

java

CopyEdit

```
@Service
public class EmbeddingService {

   private final BertModel bertModel;
   private static final int EXPECTED_EMBEDDING_SIZE = 768; // Ensure this matches
your model

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

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

      // Check if embedding size matches expected
      if (embedding.size() != EXPECTED_EMBEDDING_SIZE) {
         throw new IllegalStateException("Embedding size mismatch! Expected: " +
EXPECTED_EMBEDDING_SIZE +
            ", but got: " + embedding.size());
      }

      return embedding;
   }
}}
```

3-2. **BertModel Class - Model Load**

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


import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

import java.util.Map;


import ai.djl.ModelException;

import ai.djl.huggingface.tokenizers.HuggingFaceTokenizer;

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 HuggingFaceTokenizer tokenizer;

  private final Predictor<String, float[]> predictor;


  public BertModel() throws ModelException, IOException {

    // Load pre-trained model from DJL ModelZoo

    model = ModelZoo.loadModel(

        Criteria.builder()

            .setTypes(String.class, float[].class)
```

```
                .optModelUrls("djl://ai.djl.huggingface/sentence-transformers/all-
MiniLM-L6-v2")
                .optArguments(Map.of("tokenizer", "bert-base-uncased"))
                .build());

        tokenizer = HuggingFaceTokenizer.newInstance("bert-base-uncased");
        predictor = model.newPredictor();
    }

    public List<Float> getEmbedding(String text) throws TranslateException {
        // Fix List<String> → String[] conversion
        List<String> tokenList = tokenizer.tokenize(text);
        String[] tokens = tokenList.toArray(new String[0]);

        // Correct token joining
        String processedText = String.join(" ", tokens);

        float[] embedding = predictor.predict(processedText);

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

}}**Internship Controller - Using Real Embeddings**

In the **InternshipController**, when a new internship is created or updated, we'll generate real embeddings using **mBERT** and store them in MongoDB.

```
package com.internlink.internlink.controller;


import com.internlink.internlink.model.Internship;

import com.internlink.internlink.service.EmbeddingService;

import com.internlink.internlink.service.InternshipService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.*;


import java.util.List;


@RestController

@RequestMapping("/api/internships")

public class InternshipController {


    @Autowired

    private InternshipService internshipService;


    @Autowired

    private EmbeddingService embeddingService;
```

```java
  // Create Internship (with real embeddings)
  @PostMapping("/create")
  public ResponseEntity<?> createInternship(@RequestBody Internship internship) {
    try {
      // Generate embedding for the internship description and required skills
      String text = internship.getDescription() + " " + String.join(" ", internship.getRequiredSkills());
      List<Float> embedding = embeddingService.generateEmbedding(text);
      internship.setEmbedding(embedding);

      // Save the internship
      internshipService.createInternship(internship);
      return ResponseEntity.ok("Internship created successfully!");
    } catch (Exception e) {
      return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error creating internship: " + e.getMessage());
    }
  }

  // Update Internship (Recompute embeddings)
  @PutMapping("/update/{internshipId}")
  public ResponseEntity<?> updateInternship(@PathVariable String internshipId, @RequestBody Internship updatedInternship) {
    try {
      // Get existing internship
```

```
            Internship existingInternship =
internshipService.getInternshipById(internshipId);
            if (existingInternship == null) {
                return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Internship not
found");
            }

            // Generate new embedding
            String text = updatedInternship.getDescription() + " " + String.join(" ",
updatedInternship.getRequiredSkills());
            List<Float> newEmbedding = embeddingService.generateEmbedding(text);
            updatedInternship.setEmbedding(newEmbedding);

            // Save updated internship
            internshipService.updateInternship(internshipId, updatedInternship);
            return ResponseEntity.ok("Internship updated successfully!");
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
updating internship: " + e.getMessage());
        }
    }

    // Recommend Internships based on student's profile (vector search)
    @GetMapping("/recommend")
    public ResponseEntity<?> recommendInternships(@RequestParam String
studentId) {
        try {
            // Retrieve student embedding (this part needs to be properly implemented)
            List<Float> studentEmbedding =
internshipService.getStudentEmbedding(studentId);


            // Get recommended internships using vector similarity (MongoTemplate)
            List<Internship> recommendedInternships =
internshipService.getRecommendedInternships(studentEmbedding);
            return ResponseEntity.ok(recommendedInternships);
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error fetching
recommendations: " + e.getMessage());
        }
    }
}
```

Internship Service - Store and Query Vector Search

In your **InternshipService**, when querying for internships using MongoDB Atlas Vector Search, you'll need to use the stored embeddings.

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


import com.internlink.internlink.model.Internship;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.stereotype.Service;


import java.util.List;
```

```
@Service
public class InternshipService {

   @Autowired
   private MongoTemplate mongoTemplate;

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

   // Update Internship
   public void updateInternship(String internshipId, Internship updatedInternship) {
      mongoTemplate.save(updatedInternship);
   }

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

   // Get Student embedding (to be implemented)
public List<Float> getStudentEmbedding(String studentId) {
   Student student = mongoTemplate.findById(studentId, Student.class);
   if (student != null && student.getEmbedding() != null) {
      return student.getEmbedding();
```

```
  }
  throw new IllegalStateException("Student embedding not found!");
}
  // Recommend internships based on vector similarity (using MongoDB Vector Search)
public List<Internship> getRecommendedInternships(List<Float> studentEmbedding) {
  Document vectorSearchQuery = new Document("$vectorSearch", new Document()
      .append("queryVector", studentEmbedding)
      .append("path", "embedding")
      .append("numCandidates", 10)
      .append("limit", 5)
      .append("index", "embedding_index")
  );

  // Use aggregation pipeline with vector search
Aggregation aggregation = Aggregation.newAggregation(
    Aggregation.match(vectorSearchQuery) // ✅ `$vectorSearch` does NOT need another match.
);
  AggregationResults<Internship> results = mongoTemplate.aggregate(aggregation, "internships", Internship.class);
  return results.getMappedResults();
}
```
Student Controller

We'll modify the **StudentController** to use **mBERT embeddings** for the student profiles and store those embeddings in MongoDB.

java

CopyEdit

```
        package com.internlink.internlink.controller;

import com.internlink.internlink.model.Student;
import com.internlink.internlink.service.EmbeddingService;
import com.internlink.internlink.service.StudentService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/students")
public class StudentController {

  @Autowired
  private StudentService studentService;

  @Autowired
  private EmbeddingService embeddingService;

  @Autowired
  private PasswordEncoder passwordEncoder;
  // Register a student
  @PostMapping("/register")
  public ResponseEntity<String> registerStudent(@RequestBody Student student) {
    try {
      // Hash the password before saving
      student.setPassword(passwordEncoder.encode(student.getPassword()));
      student.setUserRole("STUDENT");

      // Generate embedding for the student profile (skills, majors, location, etc.)
      String text = student.getSkills() + " " + student.getMajor() + " " +
student.getLocation();
      List<Float> embedding = embeddingService.generateEmbedding(text);
      student.setEmbedding(embedding);

      // Save the student in the database
      studentService.registerStudent(student);

      return ResponseEntity.ok("Student registered successfully!");
    } catch (Exception e) {
      return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
registering student: " + e.getMessage());
    }
  }

  // Update student profile (recompute embedding)
  @PutMapping("/{studentId}")
  public ResponseEntity<?> updateStudent(@PathVariable String studentId,
@RequestBody Student updatedStudent) {
    try {
      // Get existing student from database
      Student existingStudent = studentService.getStudentById(studentId);
      if (existingStudent == null) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body("Student not
found");
      }

      // Generate new embedding for updated profile
      String text = updatedStudent.getSkills() + " " + updatedStudent.getMajor() + " "
+ updatedStudent.getLocation();
      List<Float> newEmbedding = embeddingService.generateEmbedding(text);
      updatedStudent.setEmbedding(newEmbedding);

      // Update student data in database
      studentService.updateStudent(studentId, updatedStudent);

      return ResponseEntity.ok("Student profile updated successfully!");
    } catch (Exception e) {
      return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error
updating student: " + e.getMessage());
    }
  }
```

```
    // Get student by ID
    @GetMapping("/{studentId}")
    public ResponseEntity<?> getStudentById(@PathVariable String studentId) {
        try {
            Student student = studentService.getStudentById(studentId);
            return student != null ? ResponseEntity.ok(student) :
ResponseEntity.status(HttpStatus.NOT_FOUND).body("Student not found");
        } catch (Exception e) {
            return
ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body("Error fetching
student: " + e.getMessage());
        }
    }
}
```

Student Service

In the **StudentService**, we will implement methods using **MongoTemplate** to handle student operations such as registering, updating, and retrieving students. We will also ensure the embeddings are handled properly.

java

CopyEdit

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


import com.internlink.internlink.model.Student;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.mongodb.core.MongoTemplate;
import org.springframework.stereotype.Service;
import java.util.List;

import java.util.Optional;


@Service

public class StudentService {


  @Autowired

  private MongoTemplate mongoTemplate;


  // Register a new student

  public void registerStudent(Student student) {

    mongoTemplate.save(student);

  }


  // Get a student by ID

  public Student getStudentById(String studentId) {

    return mongoTemplate.findById(studentId, Student.class);

  }


  // Update student profile

  public void updateStudent(String studentId, Student updatedStudent) {

    // Find the existing student by ID and update

    Student existingStudent = mongoTemplate.findById(studentId, Student.class);

    if (existingStudent != null) {

      updatedStudent.setId(existingStudent.getId()); // Ensure the same student ID
```

```
        mongoTemplate.save(updatedStudent); // Save the updated student
      }
    }

    // Optionally, you could implement a method to retrieve the embedding for a
student (e.g., for recommendation purposes)
    public List<Float> getStudentEmbedding(String studentId) {
      Student student = mongoTemplate.findById(studentId, Student.class);
      if (student != null) {
        return student.getEmbedding();
      }
      return null; // Return null or empty list if student not found
    }
}
```