+

**Employee Management Project using Spring Boot**

**Done By: M.Kowsthabi**

**Swarnandhra College Of Engineering And Technology**

**Requirements:**

Java-17

Spring Tool Suite

Post Man

Browser

My SQL Work Bench

**Spring Stater project**

During the creation of the Spring Starter project

Name: CRUD

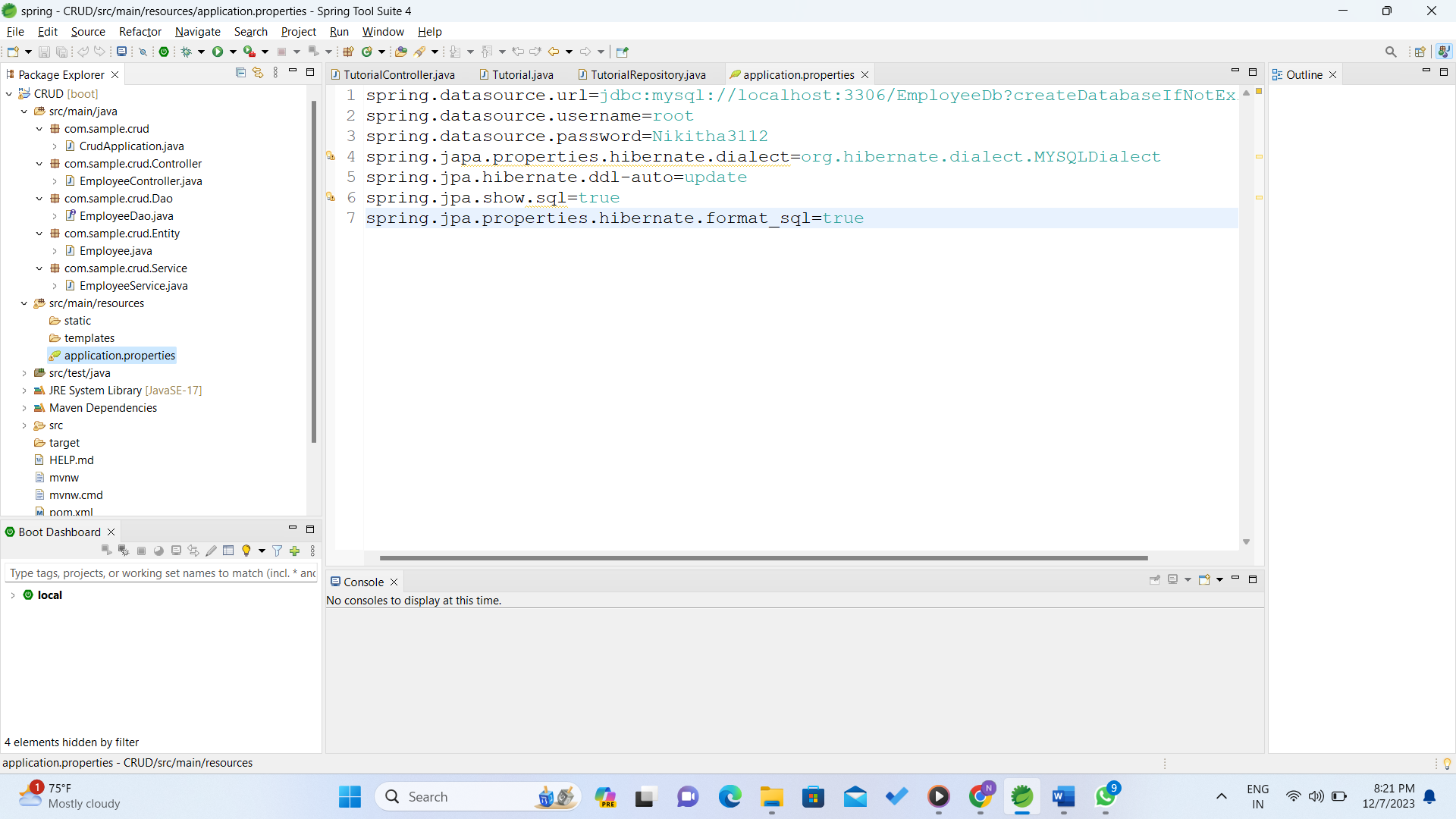
Group: com.sample

Artifact: crud

Package: com.sample.crud

Dependencies: Spring Data JPA , Spring Web , MySQL Driver.

**Folders**



In this I created four packages i.e

com.sample.crud.Controller > EmployeeController.java(class)

com.sample.crud.Dao> EmployeeDao.java(Interface)

com.sample.crud.Entity> Employee.java(class)

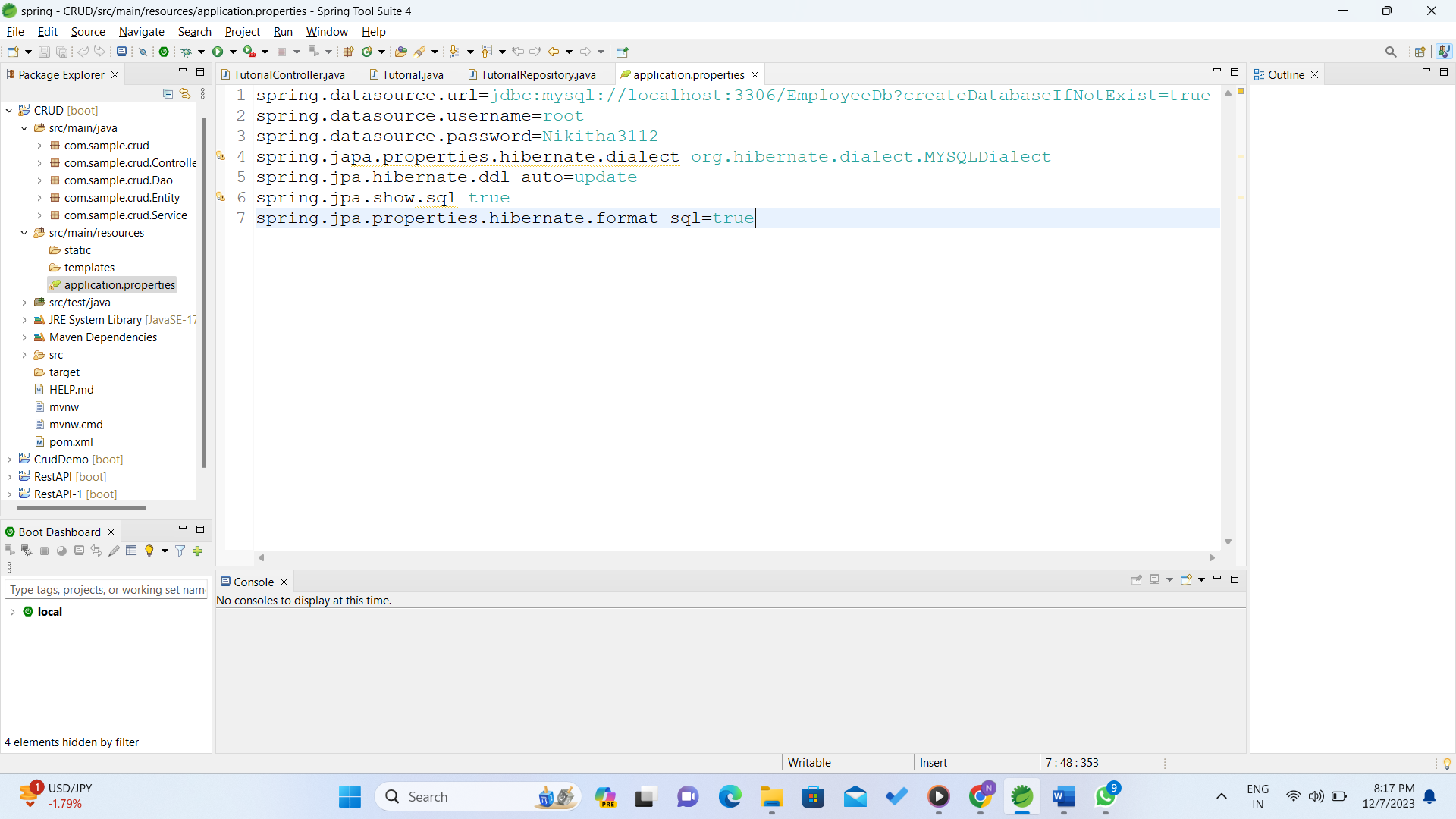
com.sample.crud.Service>EmployeeService.java(class)

**Step:1**

In **src/main/resources** folder there is an **application.properties**

file

I started coding with that file



**spring.datasource.url:** Specifies the URL for the MySQL database. Here, it's connecting to a database named "EmployeeDb" on localhost, port 3306.

**spring.datasource.username:** Specifies the username used to access the database ("root" in this case).

**spring.datasource.password:** Specifies the password for the given username ("Nikitha3112" in this case).

**spring.jpa.properties.hibernate.dialect:** Defines the Hibernate dialect for MySQL.

**spring.jpa.hibernate.ddl-auto:** Sets the behavior for Hibernate to automatically update the database schema. Here, it's set to "update," meaning it'll modify the schema based on the entity mappings.

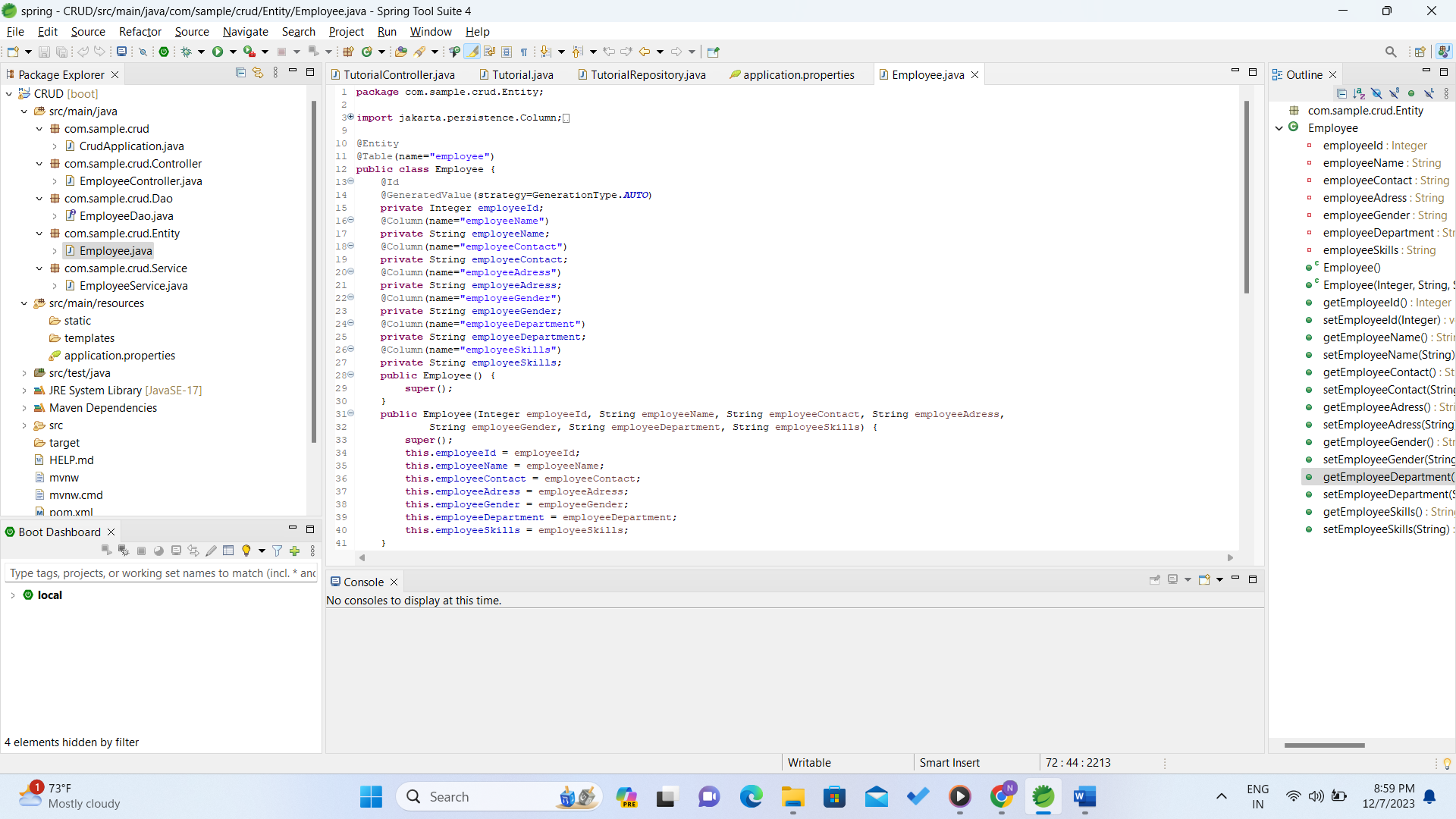
**spring.jpa.show.sql:** Enables the logging of SQL statements executed by Hibernate.

**spring.jpa.properties.hibernate.format\_sql:** Formats the SQL queries in a human-readable format when logging is enabled.

It will create the "EmployeeDb" database if it doesn't already exist and update its schema based on the entities defined in the application

**Step-2:**

com.sample.crud.Entity> Employee.java(class)



**@Entity:** This annotation marks the class as a JPA entity, indicating that instances of this class will be persisted to a database table.

**@Table(name="employee"):** Specifies the name of the table in the database where instances of this entity will be stored. In this case, it's mapped to a table named "employee".

**@Id:** Indicates the primary key of the entity.

**@GeneratedValue(strategy=GenerationType.AUTO):** Specifies that the employeeId field will be automatically generated by the underlying database. The exact strategy (AUTO) may vary based on the database provider.

**@Column:** This annotation is used to specify the mapping between the entity attribute and the corresponding column in the database table.

Below are the attributes of the Employee class:

**employeeId:** An integer field annotated with @Id and @GeneratedValue to serve as the primary key for the Employee entity.

employeeName, employeeContact, employeeAddress, employeeGender, employeeDepartment, employeeSkills: These are fields representing various attributes of an employee. They are mapped to columns in the database table named employee with respective names provided using @Column.

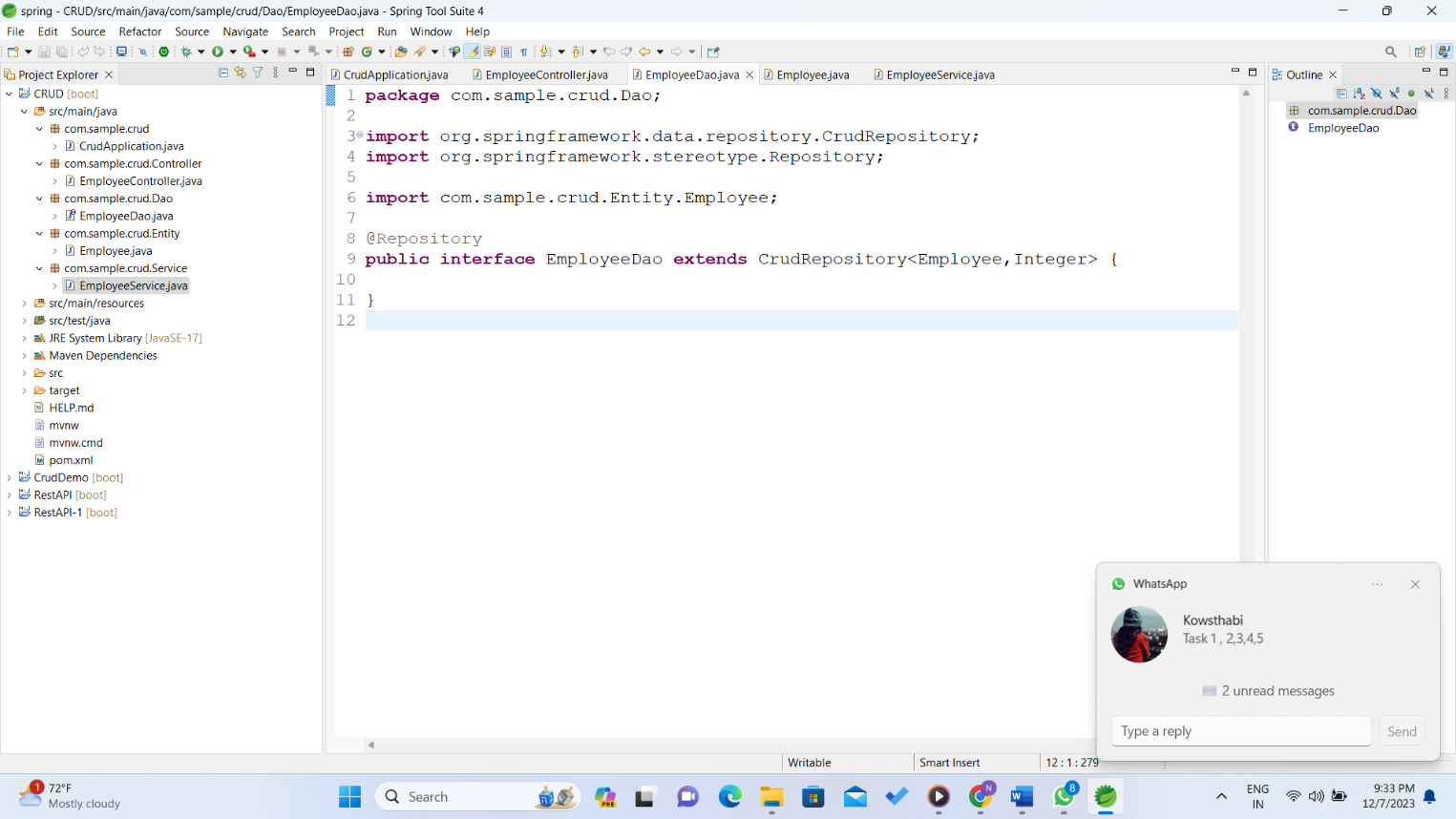
**Two constructors:** One is a default constructor, and the other is a parameterized constructor allowing the initialization of all fields of the Employee class.

**Getter and setter methods for each field:** These methods are used to retrieve and modify the values of the fields encapsulated within the Employee class.

this code defines an Employee entity class that can be used to interact with a database table named "employee," storing information related to employees, such as their name, contact information, address, gender, department, and skills.

**Step-3**

com.sample.crud.Dao> EmployeeDao.java(Interface)



**@Repository:** This annotation marks the interface as a Spring-managed repository component, allowing it to be automatically detected during component scanning.

**CrudRepository<Employee, Integer>:** This interface extends Spring Data's CrudRepository, which provides basic CRUD operations (Create, Read, Update, Delete) for the Employee entity.

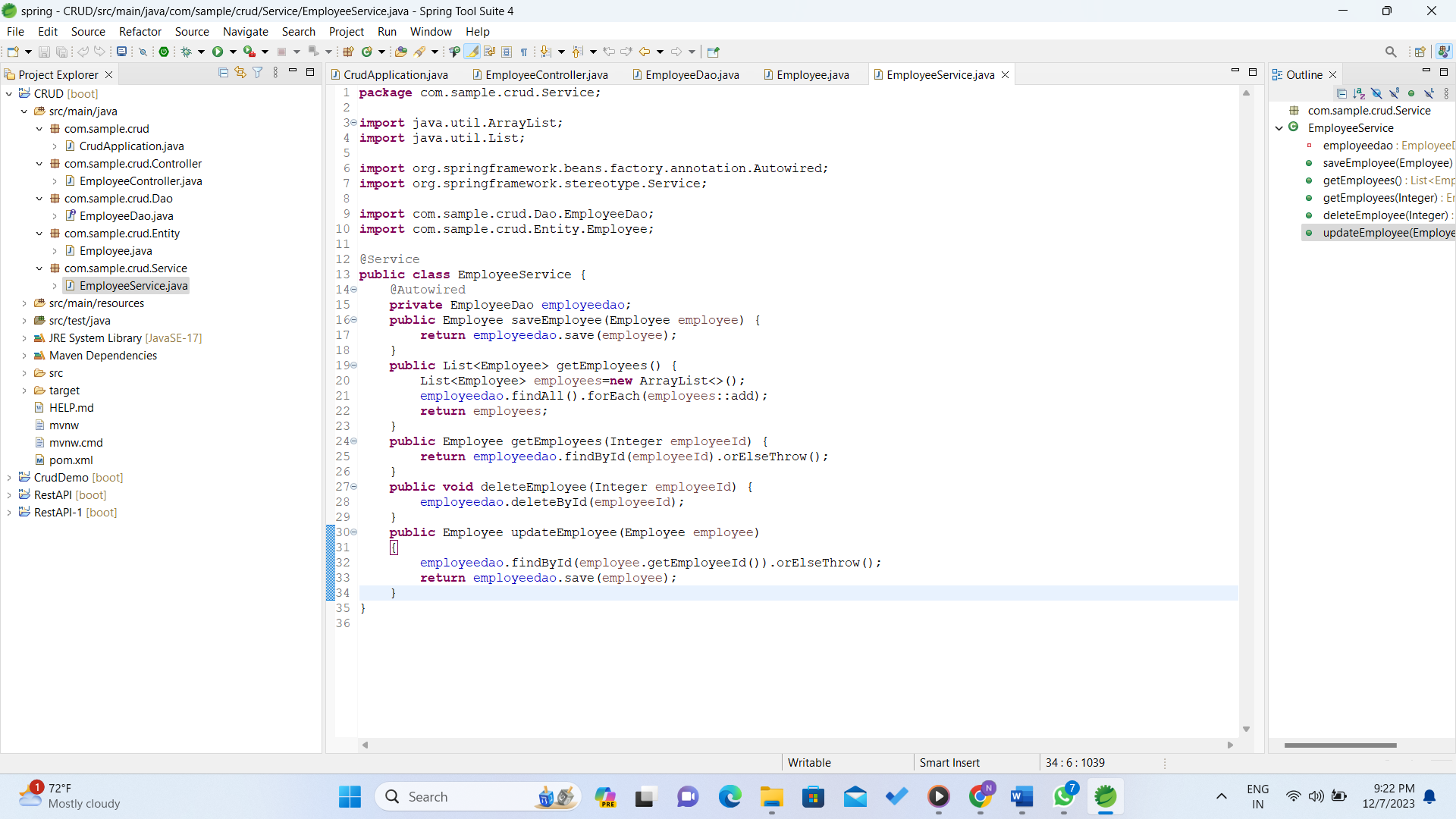
**Employee:** Specifies the entity type that this repository manages (in this case, Employee).

**Integer:** Represents the type of the entity's primary key (employeeId), indicating that it's an integer.

By extending CrudRepository, EmployeeDao inherits several methods such as save, findAll, findById, and deleteById, among others, which can be used to perform database operations on the Employee entity without explicitly implementing these methods. Spring Data automatically generates the necessary queries based on method names or using query derivation from method signatures. This interface acts as an intermediary between the service layer and the database, allowing easy access to the database operations related to the Employee entity.

**Step-4**

com.sample.crud.Service>EmployeeService.java(class)



This code represents a service class EmployeeService responsible for managing Employee entities.

@Service: This annotation marks the class as a Spring-managed service component, allowing it to be automatically detected by component scanning.

**@Autowired private EmployeeDao employeedao:** This annotation injects an instance of EmployeeDao into the service. The EmployeeDao presumably handles the CRUD (Create, Read, Update, Delete) operations for the Employee entity.

**public Employee saveEmployee(Employee employee):** This method saves an employee entity by utilizing the EmployeeDao's save() method, returning the saved entity.

**public List<Employee> getEmployees():** Retrieves a list of all employees by fetching all records from the database using employeedao.findAll() and returning them as a list.

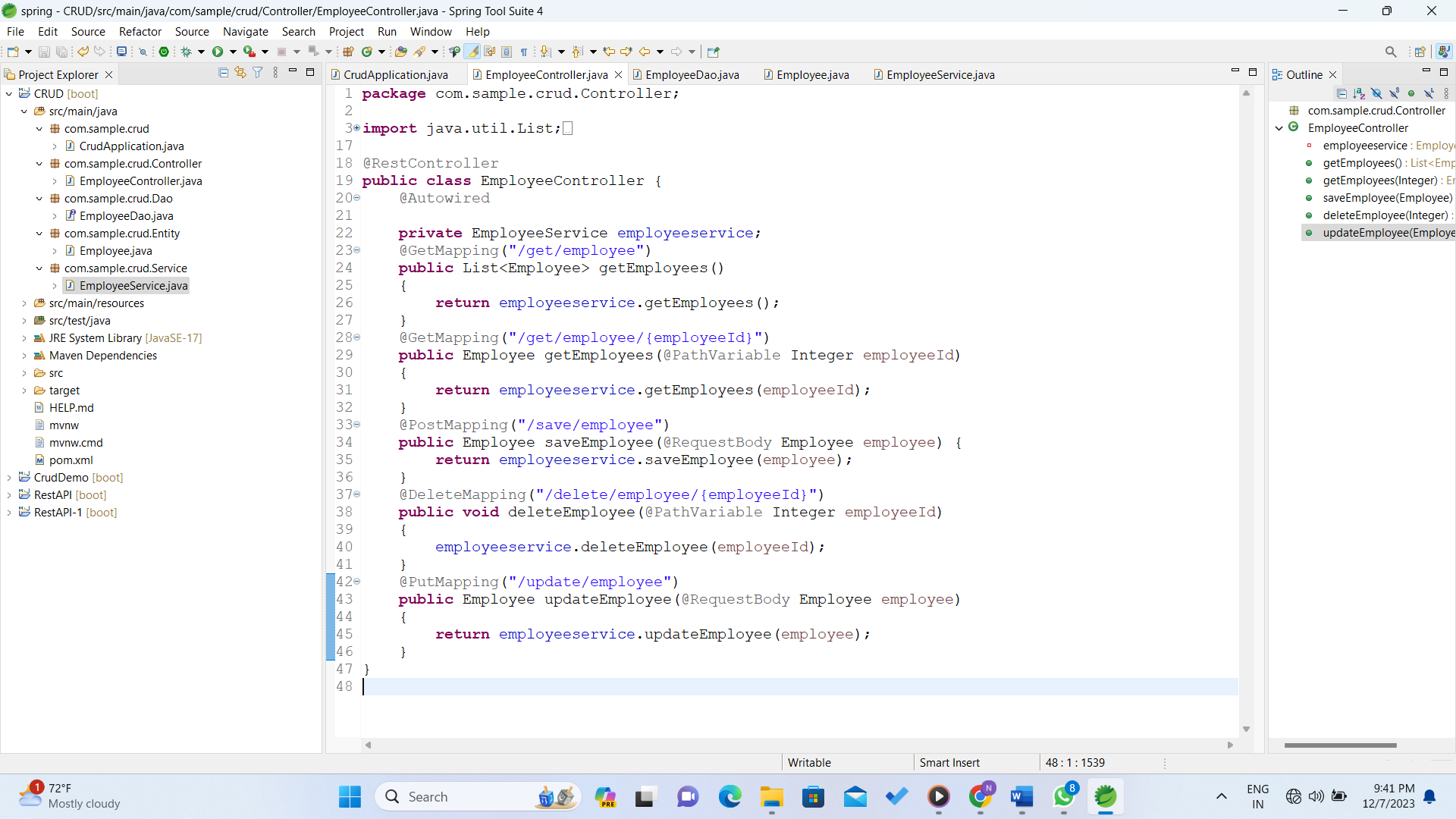
**public Employee getEmployees(Integer employeeId):** Retrieves a specific employee by their employeeId using employeedao.findById(employeeId). If the employee exists, it returns the employee; otherwise, it throws an exception.

**public void deleteEmployee(Integer employeeId):** Deletes an employee from the database by their employeeId using employeedao.deleteById(employeeId).

**public Employee updateEmployee(Employee employee):** Updates an employee's information. It first attempts to find the employee by their employeeId. If found, it updates the employee information using employeedao.save(employee).

**Step-5**

com.sample.crud.Controller > EmployeeController.java(class)



**Annotations:**

**@RestController:** This annotation is used to mark the class as a Spring MVC controller that handles web requests.

**@GetMapping, @PostMapping, @DeleteMapping, @PutMapping:** These annotations are used to map HTTP requests to specific methods in the controller.

**Dependencies:**

**@Autowired:** This annotation is used for automatic dependency injection. In this case, it injects an instance of EmployeeService into the controller.

**Instance Variable:**

**private EmployeeService employeeservice:** This private variable represents an instance of the EmployeeService. It is annotated with @Autowired, so Spring will inject an instance of EmployeeService automatically.

**GET Request to Retrieve All Employees:**

This method handles a GET request to "/get/employee" and returns a list of all employees by calling the getEmployees() method of the injected EmployeeService.

**GET Request to Retrieve a Specific Employee by ID:**

This method handles a GET request to "/get/employee/{employeeId}" and returns a specific employee identified by the provided employeeId.

**POST Request to Save a New Employee:**

This method handles a POST request to "/save/employee" and saves a new employee by calling the saveEmployee() method of the injected EmployeeService. The employee details are expected to be in the request body.

**DELETE Request to Delete an Employee by ID:**

This method handles a DELETE request to "/delete/employee/{employeeId}" and deletes the employee with the specified employeeId.

**PUT Request to Update an Existing Employee:**

This method handles a PUT request to "/update/employee" and updates an existing employee by calling the updateEmployee() method of the injected EmployeeService. The updated employee details are expected to be in the request body.