



MIDDLE EAST TECHNICAL UNIVERSITY NORTHERN CYPRUS CAMPUS

Computer Engineering Program

CNG 300

SUMMER PRACTICE REPORT

Name of Student : Fatma Erem Aksoy

ID Number : 2315075

Name of Company : VBT Information Technology Inc. (VBT Yazılım A.Ş.)

Project Title : Testing and Analyzing Software by Using Different Tools

Date of Submission : 09 April 2021

ABSTRACT

This report has all the necessary information about the problems I faced and solutions during my summer practice at VBT Yazılım A.Ş for 21 days. My practice's main objective is learning the necessary basics for testing and analyzing software, getting familiar with different software tools, learning a new programming language, working with databases, and getting work experience. During my internship, I worked on tasks that make a connection between the backend and frontend. I have gained knowledge about object-oriented languages in general and worked with Java for the back-end part of my tasks. I learned how to get data from the database and make the required changes on the data sets. I learned how to get the request from the frontend by a user and write functions for the required changes on the back-end side. I also learned how to analyze the code and write a cleaner and more efficient version of it. Some of my tasks also focused on connecting to a database by writing codes on Java, how to grasp the problem in the code by debugging and how to create solutions for them. I have gained satisfying working experience and improved my skills, including problem-solving, analytical thinking, communication, and software analyzing/testing. As I have been given daily tasks about different topics, I believe this summer practice has gained me loads of new comprehensions from different aspects. I can list my gains as learning a new language, Java, using a development tool, Eclipse, getting familiar with the databases and learning SQL, and using other platforms/frameworks such as Spring Boot and Postman.

TABLE OF CONTENTS

ABSTRACT	2
LIST OF FIGURES AND TABLES	5
1. DESCRIPTION OF THE COMPANY	6
1.1 Name, History, Location and Services	6
1.2 Organizational Structure.....	6
1.3 Technologies Used by The Company	7
1.4 Personal Observations	7
2. INTRODUCTION	8
3. PROBLEM STATEMENT	8
3.1 Object Oriented Programming (OOP) - Java.....	9
4. ECLIPSE IDE	9
5. POSTGRESQL.....	10
5.1 pgAdmin	10
6. POSTMAN.....	11
7. SPRING	12
7.1 Spring Boot.....	12
7.2 Spring Initializr	13
7.3 Spring Data JPA.....	14
8. HIBERNATE – ORM (Object Oriented Mapping).....	14
9. SOLUTION	15
9.1 Entering Data to Database	15
9.2 Building a Maven Project in Spring Initializr.....	18
9.3 Creating a Maven Project in Eclipse.....	19
9.4 Creating Packages, Classes and Objects	20
9.5 Spring Boot Annotations.....	25
9.5.1 Spring Framework Stereotype Annotations.....	26
9.5.2 Spring REST Annotations	31
9.6 Connecting to Database via Eclipse	32
9.7 Sending Request from Postman.....	33
9.8 Service	40
9.8.1 Service Implementation	40
9.8.2 Mapper.....	41

9.8.3 Design Patterns: The Build Pattern	42
10. CONCLUSION.....	45
REFERENCES	46

LIST OF FIGURES AND TABLES

Figure 1: pgAdmin	10
Figure 2: Postman	11
Figure 3: The working way of Spring Boot	12
Figure 4: Spring Initializr	13
Figure 5: Displaying the data in <i>advisor</i> table by using SELECT keyword.....	16
Figure 6: Displaying the data in <i>course_advisor</i> table by using SELECT keyword	17
Figure 7: Displaying the data in <i>courses</i> table by using SELECT keyword.....	17
Figure 8: Displaying the data in <i>student</i> table by using SELECT keyword	18
Figure 9: Project demo and its pom.xml file with JPA dependency added.....	19
Figure 10: DemoApplication / Main of the project.....	22
Figure 11: <i>Advisor</i> class and necessary annotations	23
Figure 12: <i>Course_Advisor</i> class and necessary annotations	23
Figure 13: <i>Courses</i> class and necessary annotations	24
Figure 14: <i>Student</i> class and necessary annotations	24
Figure 15: StudentController class.....	27
Figure 16: AdvisorController class	28
Figure 17: StudentService interface.....	28
Figure 18: AdvisorService interface	29
Figure 19: StudentRepository interface.....	29
Figure 20: AdvisorRepository interface	30
Figure 21: To connect to the database: application.properties	32
Figure 22: Connection to the database	33
Figure 23: POST request to save a new student and updated table in the database	34
Figure 24: GET request to print the student list.....	35
Figure 25: GET request to find student(s)	36
Figure 26: POST request to update student's information.....	37
Figure 27: DELETE request to delete a specific student.....	38
Figure 28: GET request for an advisor.....	39
Figure 29: AdvisorServiceImplementation and StudentServiceImplementation classes	41
Figure 30: The Mapper Class	41
Figure 31: StudentDTO and Builder Classes for a Student Entity	43
Figure 32: AdvisorDTO and Builder Classes for an Advisor Entity	44

DESCRIPTION OF THE COMPANY

In this section, detailed information about the company will be given. The information is divided into three parts as shown below.

1.1 Name, History, Location and Services

VBT Yazılım A.Ş.(current name of the company) was founded on June 24, 1993 under the name of "Vizyon Bilgi İşlem ve Danışmanlık Limited Şirketi" in Turkey. It has local offices in 3 major cities, which are Istanbul, Ankara, and Adana. It has over 50 corporate customers, and 5 of these companies are on the list of Turkey's largest ten companies. VBT ranked 97th in the Top 500 in the 2018 Ranking, so it is among the top 100 IT companies in Turkey. IBM, BMC, BROADCOM, MICRO FOCUS, REDHAT, ABB, NEWGEN, DIGPRO, JENNIFER Business are the companies that have partnerships with VBT Yazılım A.Ş. I have done my internship in Ankara Office which is located at Mahall İş Merkezi, B Blok No:154, Çankaya.

One of the company's leading roles is building applications to transmit invoices electronically, creating virtual books that can save accounting and warehouse expenses, and getting electronic tickets. They also offer consultancy services and unique training opportunities, including mainframe, software development, mobile, database, security, network, web design, and system infrastructure management. VBT also provided services to institutions such as İşbank, Avea, IBM, OYAK Renault, Pepsi, THY, and SGK on outsourcing and still provide services.

1.2 Organizational Structure

The company's vision is to closely follow all the technological innovations by blending them with the company's software to provide a superior service approach to the customer portfolio in Turkey and make R&D activities with Turkish Engineers and compete in the global IT market. Its mission is to increase its customers' competitiveness by using the most up-to-date Information Technology opportunities and creating the highest customer satisfaction with long-term and trust-based relationships.

The company has 220 employees as of the end of 2019. There are nine company shareholders, and the Chairman of the Board is Mr. Birol Başaran. The company is managed by the General Manager, Assistant General Managers, and Directors, who serve as the Executive Board. The company has one Chairman of the

board, one general manager, one management consultant, one head of software architect, four managers, four directors, and five deputy general managers in the management team.

1.3 Technologies Used by The Company

Mainframe Software: ZOS, CICS, DB2, and IMS.

IBM: DB2, Optim, Guardium.

Remedy, Control-M, and TrueSight.

Jennifer: Jennifer APM (Application Performance Monitoring). Programming

Languages: C++, .Net, Java, Web programming, Mainframe Application

Development: COBOL, PL/1, Assembler, Natural.

Mainframe Platform: z/OS, CICS, DB2, IMS, MQ, VTAM, TCP/IP.

AS/400 Platform.

ITMS (IT Service Management System)

Open Systems: Unix, Linux.

Database: DB2, ORACLE, SQL SERVER, ADABAS.

1.4 Personal Observations

During my summer practice, I observed that the office that I worked at is well-organized. The workers were always synchronized and able to solve problems quickly and were willing to help others solve their problems more efficiently. I have a couple of reasons to choose this company for my summer practice. I heard about this company from one of my father's friends. I also saw that this company is available in the ODTU(Ankara Campus) Teknokent, so I thought it must be a good company to improve myself. After researching the company and seeing that they work on my interest areas which are databases and software testing, I thought doing my practice there would be so helpful for me. For more detail about the company, you can check their website [1].

2. INTRODUCTION

I have done my practice at VBT Yazılım A.Ş, Ankara, Turkey. During my internship, I have observed that the working environment there was so relaxing and encouraging. Everyone was trying to aid each other even though when they work on many and different projects. Everyone was full of knowledge which was something I admire so much. The office that I worked at was working on many different projects at that time, and my supervisor was the head of all the projects. His name is Alper Çepni, and he helped to guide me throughout my practice.

I was given daily tasks by my supervisor on different matters and expected to finish them within the day. I have worked on various subjects that I did not know before. I have learned some basics about Java, which is the language that I used for my tasks. I learned about databases, especially SQL, using software tools and frameworks such as Eclipse, Spring Boot, and Postman. My tasks focus on understanding how backend and frontend communicate and what happens at the backend during this process by analyzing the software. To be more specific, I have performed tasks to understand how the back-end code manages to get the request from the frontend, how it connects to the database, and how the code works to perform the demands and make necessary changes. I believe I put all my effort into doing my best for all the tasks I have had because my motivation during the entire practice was to obtain as much knowledge as possible. I know this practice is so beneficial for my future goals as I am very interested in database concepts.

The rest of the report will have pieces of information about my gains during my internship period. I will explain the tasks I have given. I will provide some figures to clarify my work for all the tasks. I will mention my struggles and the solutions afterward. The report will also have a quick summary of everything in the Conclusion section.

3. PROBLEM STATEMENT

During my practice, I have worked on a single problem that has multiple subproblems. I have started with learning the basics of Java and SQL because they are cornerstones of my work. I worked on Java for the back-end part and SQL for the database part. I used Eclipse IDE as a software tool to code on Java, and I found it very handy and beginner-friendly. Then I learned about some other tools like Spring Boot and Postman and used them in my work as well. They both are essential for my work as they helped me solve

connecting to a database, getting the request from the frontend properly, and testing the software. After having the knowledge of how to use all these tools and test the software, I was able to get and process the data. Also, as a final step of my work, I have tried to make my code easier to read and control.

To finalize, I can say that I have had a single problem, but I have worked on several tasks to solve this one problem.

3.1 Object Oriented Programming (OOP) – Java

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). A feature of objects is that an object's own procedures can access and often modify the data fields of itself (objects have a notion of `this` or `self`). In OOP, computer programs are designed by making them out of objects that interact with one another. OOP languages are diverse, but the most popular ones are class-based, meaning that objects are instances of classes, which also determine their types.

Many of the most widely used programming languages (such as C++, Java, Python, etc.) are multi-paradigm and they support object-oriented programming to a greater or lesser degree. Significant object-oriented languages include: Java, C++, C#, Python, R, PHP, etc. [2]. The reason for choosing Java for my work is that I wanted to learn a new language that is widely used and to widen my knowledge of object-oriented languages that I have from C++. Detailed information and options to install Java can be found on the website [3].

4. ECLIPSE IDE

Eclipse IDE is well-known for Java Integrated Development Environment (IDE), and it allows the combination of numerous languages, which is very handy for creating projects that need different languages. I have used Eclipse IDE for Java Developers 2020-12 version, the latest stable version for coding purposes. You can check their website for more detail and installation options [4].

5. POSTGRESQL

PostgreSQL is the most advanced open-source relational database management system (RDBMS) that can smoothly handle a range of workloads, from single machines to data warehouses and Web services with many concurrent users. It is also the default database for Windows, which makes it easy for me to use. I have worked with PostgreSQL 13.2, which was released in February 2021. The reason I have chosen PostgreSQL is that I learned it for my interests last year, and I wanted to improve my knowledge. I also thought that it would be a great idea to use it with Java to work more with object-based programs. It is also effortless to create tables and insert and edit data in those tables. You can find more information on the website [5].

5.1 pgAdmin

PgAdmin is the most famous and feature-rich open-source platform used for the administration and development of PostgreSQL. It supports multiple users over the web and most PostgreSQL server-side encodings, and allows to create, view, and edit all common PostgreSQL objects. I have used pgAdmin 4 v5.0, which is the latest release. The reason why I have picked it is that it operates fast and is user-friendly. More information can be checked on the website [6].

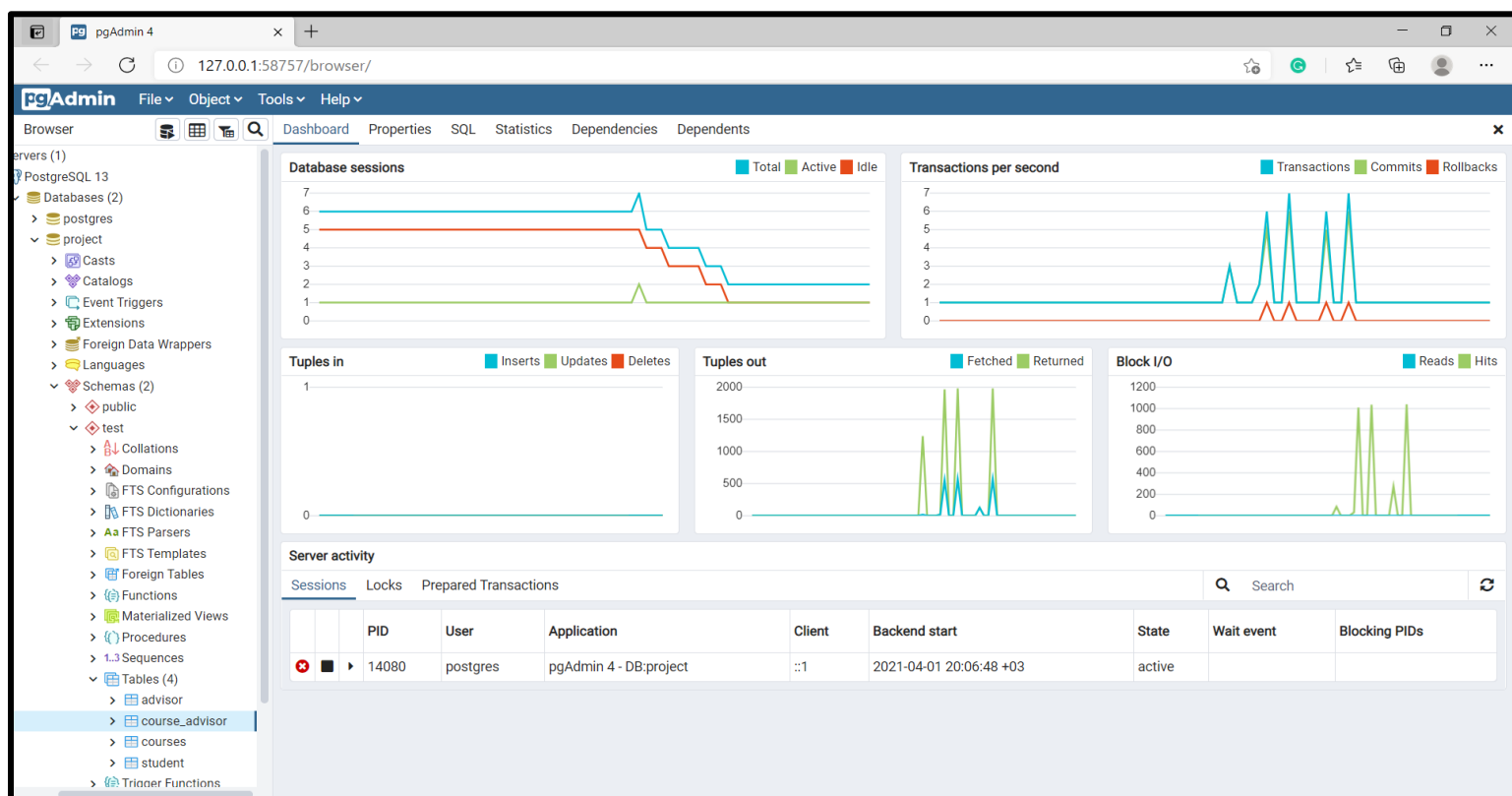


Figure 1: pgAdmin

6. POSTMAN

Postman is a collaboration platform for many purposes such as API development, software testing, and product management. I have used Postman to send different data sorts to see what values it returns for testing purposes. The reason why I have preferred this platform is that it is free and easy to get started and has broad support for all APIs, which I have used for REST.

REST (REpresentational State Transfer) is an architectural style to provide principles between computer systems on the web and make communication with each other easy for them. In the REST architecture, users send requests to recall or alter resources, and servers send responses to these requests [7].

JSON (JavaScript Object Notation) is a data-interchange format that does reading and writing easy for humans and parsing and generating easy for machines. [8]. I have picked JSON format (shown in Figure 2 below) because JSON is a well-known ideal data-interchange language.

For more information and installation options of Postman, you can check their website [9].

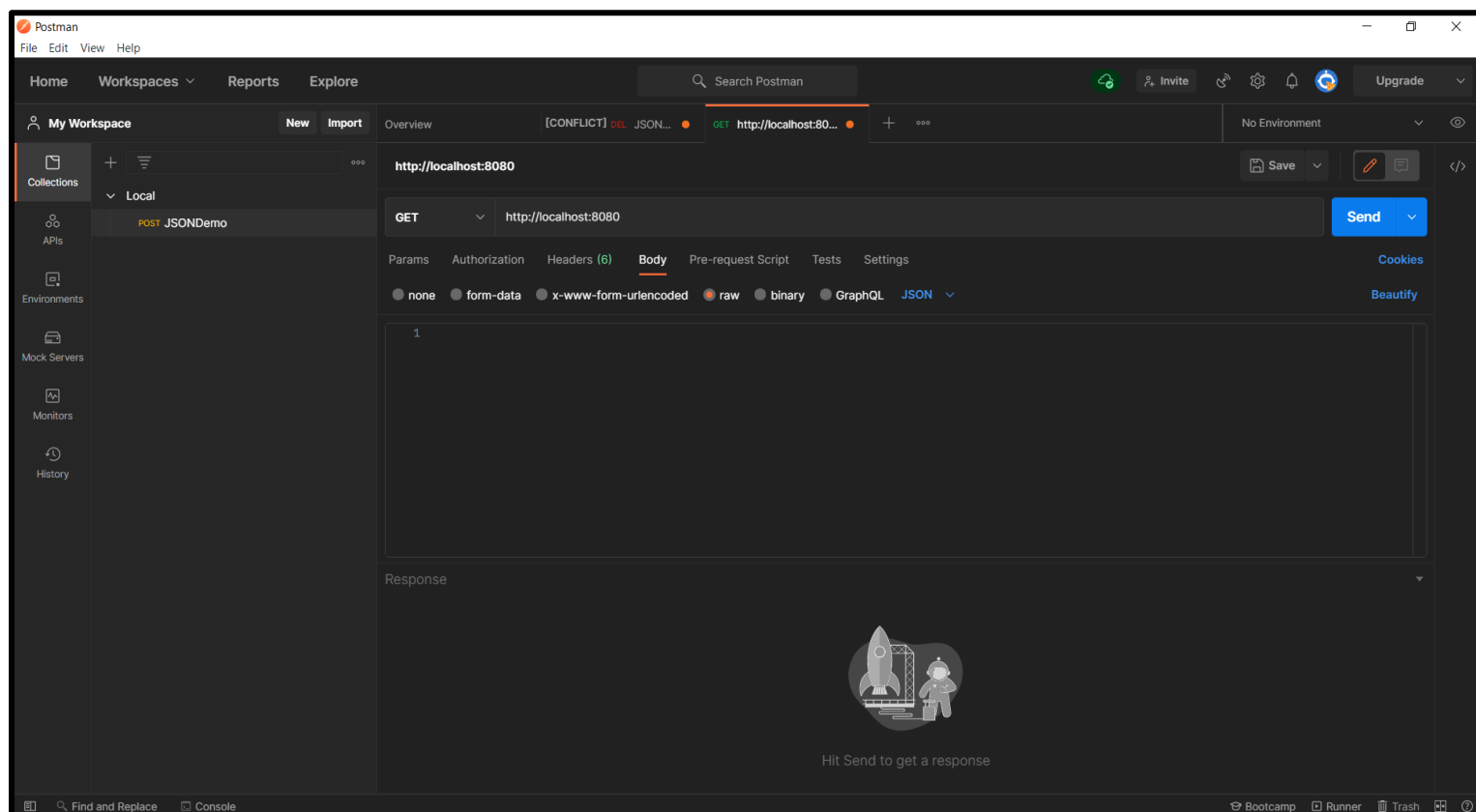


Figure 2: Postman

7. SPRING

Spring is an application framework for the Java platform. It helps to create a contemporary web programming model that can modernize the development of REST APIs by eliminating much of the web-related standard code. I have used the Spring framework for my work because it makes programming Java much faster and safer. It is easy to use and open-source. You can check the website [10] for more information about Spring and its tools.

7.1 Spring Boot

Spring Boot is a tool that hastens application development. To create Spring Boot-based projects, Spring Initializr, which will be discussed in the next session, needs to be used. To be more specific, what Spring Boot does is that it converts JSON text to a proper format so that backend and frontend can communicate/understand each other. In my work, Spring Boot converts everything to make the communication between Eclipse IDE and Postman possible so that they can understand each other and process the data accordingly.

You can check the figure 3 below to see what Spring Boot does:

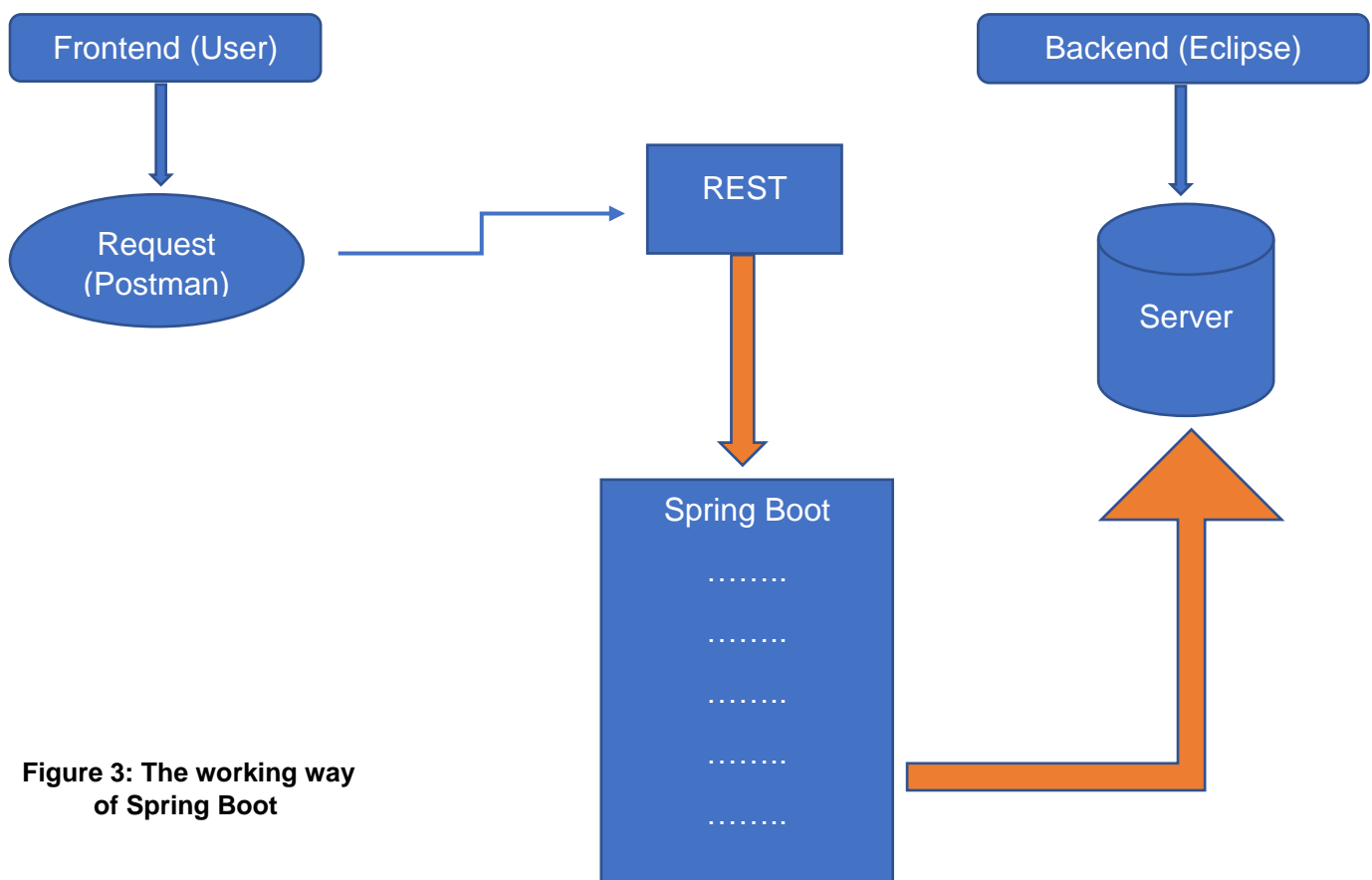


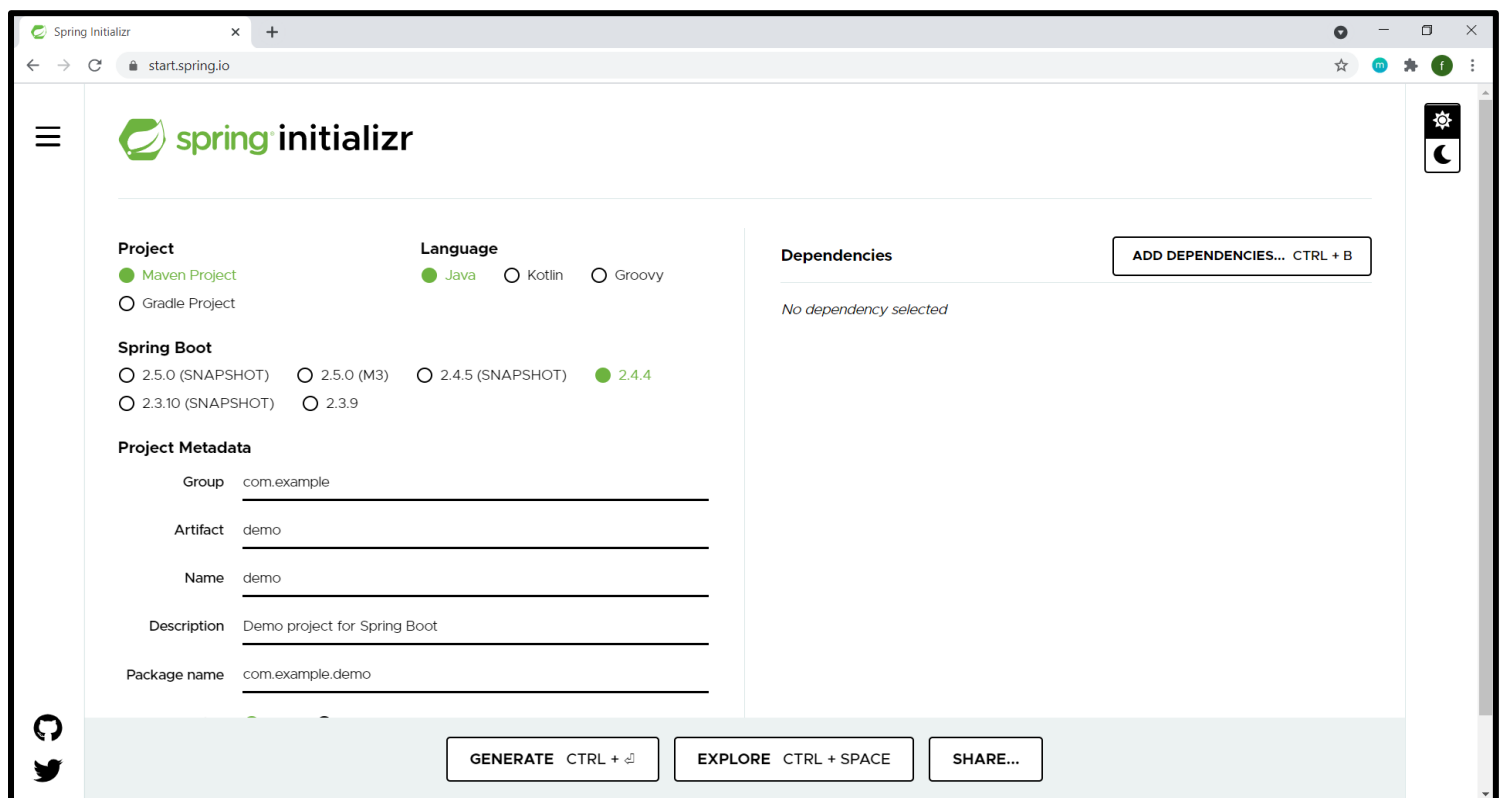
Figure 3: The working way of Spring Boot

The description of Figure 3:

The working way of Spring Boot is shown in Figure 3. We can say that it makes the connection between the request by the user and the server. We have a user send a request at the frontend and Eclipse (Java code) at the backend. When the user sends the request via Postman, the request will be forwarded to REST with the selected format, JSON (please check Figure 2 to recall the request's options). After REST takes the request, it will deliver the JSON text to Spring Boot. Then Spring Boot will convert the JSON format to such a format that the code in Eclipse can understand it and connect to the server to fulfill the requests.

7.2 Spring Initializr

Spring Initializr is a web application that can provoke a Spring Boot project structure. It helps us skip some parts, such as setting up the directory structure for artifacts, producing a build file, and dealing with dependencies for our projects. Even though it does not provide any application code, it gives the core project structure and constructs specifications to build the code. While creating the project, it wants us to specify the dependencies for the application. In other words, it asks for the sort of functionality that we want for our application development.



The screenshot shows the Spring Initializr web application in a browser window. The URL is start.spring.io. The interface is divided into several sections:

- Project:** Radio buttons for ☒ Maven Project and ☐ Gradle Project.
- Language:** Radio buttons for ☒ Java, ☐ Kotlin, and ☐ Groovy.
- Spring Boot:** Radio buttons for versions: ☐ 2.5.0 (SNAPSHOT), ☐ 2.5.0 (M3), ☐ 2.4.5 (SNAPSHOT), ☒ 2.4.4, ☐ 2.3.10 (SNAPSHOT), and ☐ 2.3.9.
- Project Metadata:** Text input fields for:
 - Group: com.example
 - Artifact: demo
 - Name: demo
 - Description: Demo project for Spring Boot
 - Package name: com.example.demo
- Dependencies:** A section with the text "No dependency selected" and a button "ADD DEPENDENCIES... CTRL + B".

At the bottom, there are three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and "SHARE...".

Figure 4: Spring Initializr

7.3 Spring Data JPA

Spring Data JPA makes the implementation of JPA-based repositories easy. It helps to create Spring-powered applications that use data access layers and reduces the effort needed to implement the layers. Spring Data JPA is beneficial when we want to store and retrieve data in a relational database, which is one of the essential parts of my practice. I have used JPA to make a connection between Eclipse and PostgreSQL. When we write the repository interfaces, Spring will automatically provide the implementation, which is very handy and the reason why I have chosen to use Spring Data JPA for my work. Another reason for choosing JPA is that it has support for XML-based entity mapping.

To summarize, JPA creates a map from the datastore to the application's data model objects. While using a relational database, much of the actual connection between the application code and the database will be handled by JDBC, the Java Database Connectivity API.

8. HIBERNATE – ORM (Object Relational Mapping)

Hibernate ORM is an object-relational mapping tool for Java. It maps an object-oriented domain model to a relational database by providing a framework. It replaces direct and stable database accesses with high-level object management functions to tackle object-relational impedance mismatch* problems. The main purpose of using Hibernate was to use its primary feature, which is mapping Java classes to database tables, and converting Java data types to SQL data types. Mapping notifies Hibernate about what Java class object to store in which database table. It uses the XML file or Java annotations to map Java classes to database tables. I have used Java annotations to maintain the database schema as this way is faster and easier to apply. For more information and applications, you can check their website [12].

**Object-relational impedance mismatch: Objects in an object-oriented application follow OOP principles, while objects in the back-end follow database normalization principles, resulting in different representation requirements. This problem is called "object-relational impedance mismatch". Mapping is a way of resolving the object-relational impedance mismatch problem. [11]*

9. SOLUTION

I have used many tools and platforms for my work. Each of them is discussed in detail throughout this section. Almost all the implementations and test cases are shown with figures.

9.1 Entering Data to Database

As I worked with a database, I needed to have some data, but instead of using sample data and tables from the internet, I have created some tables by myself and inserted some random values into them. In this way, I also wanted to practice my SQL skills and improve them. As a first step, I have created a new database called project and a schema called test. Then I have created four different tables for different purposes. The reason why I have created four tables is that I wanted to see the connection between tables to comprehend the differences between primary key and foreign key. To understand related terms better, I have made a connection between all the tables so that I have had a chance to observe how to reach the element of one table by using the other one or vice versa.

Even though I have created four tables, I focused on updating the data for two specific tables to follow all the changes better. The tables are named advisor, course_advisor, courses, and student. (check Figure 1)

advisor table: It keeps the information of four different advisors. It has three columns, which are id, advisor_name, and room. The data types used are integer for id and room, and twenty-five characters long character varying. I have set id as a primary key. (Figure 5)

course_advisor table: It keeps the information about the courses available and the advisors for these courses. The main reason why I have created this table is to use foreign keys and see how to connect two tables. It has three columns, which are ID, Advisor_ID, and Course_ID. I have set all attributes as an integer, and while an ID is a primary key, Advisor_ID and Course_ID are foreign keys. (Figure 6)

courses table: It keeps the information of courses available. It has three columns, which are Course_ID, Name, and Start_Date. I have set Course_ID as an integer, Name as twenty-five characters long character varying type, and Start_Date as date data type. Course_ID is a primary key of this table, and it has no foreign key. (Figure 7)

student table: It keeps all the information about students. It has seven columns: id, student_no, name, advisor_id, age, address, and email. I have used integer data type for id, student_no, advisor_id, and age, and twenty-five characters long character

varying type for the other columns. The primary key is id, and the foreign key is advisor_id for this table. (Figure 8)

I have mainly focused on student and advisor tables, but you can see all the tables below. For the student table (Figure 8), while displaying the data, I have used the ORDER BY keyword, specified the column as id, and added ASC keyword to see the data in ascending order.

The screenshot shows the pgAdmin 4 web interface. On the left, the 'Browser' pane displays a tree view of database objects, with 'Tables (4)' expanded to show the 'advisor' table. The 'Columns (3)' for 'advisor' are listed: 'id' (integer), 'advisor_name' (character varying (25)), and 'room' (integer). The 'Query Editor' pane shows a SQL query: `SELECT id, advisor_name, room FROM test.advisor;`. The 'Data Output' pane displays the results of the query in a table format.

	id [PK] integer	advisor_name character varying (25)	room integer
1	1	Ahmet Kocak	75
2	2	Gizem Yilmaz	86
3	3	Mehmet Bahar	90
4	4	Ayşe Akman	120

Figure 5: Displaying the data in *advisor* table by using SELECT keyword

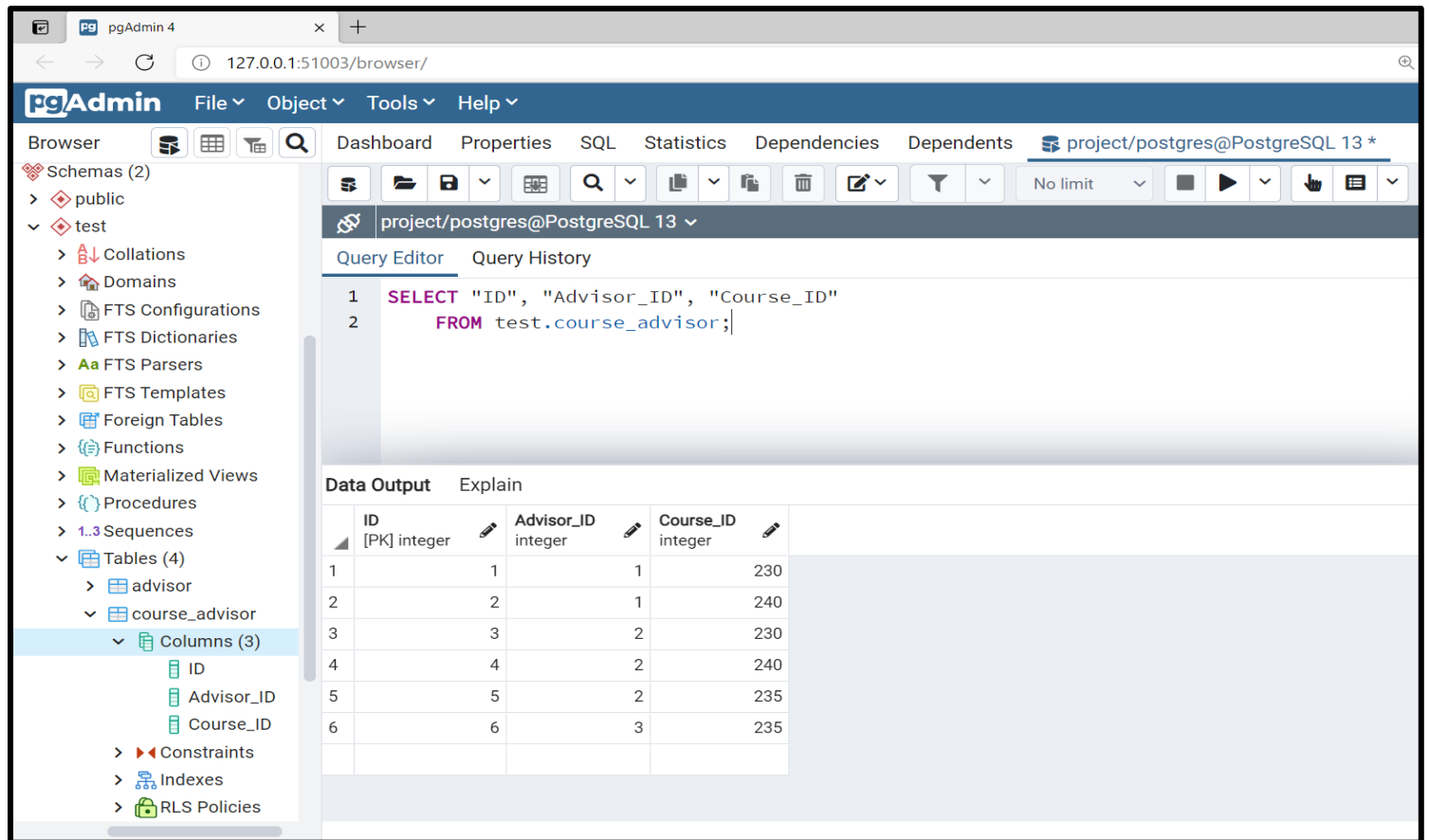


Figure 6: Displaying the data in *course_advisor* table by using SELECT keyword

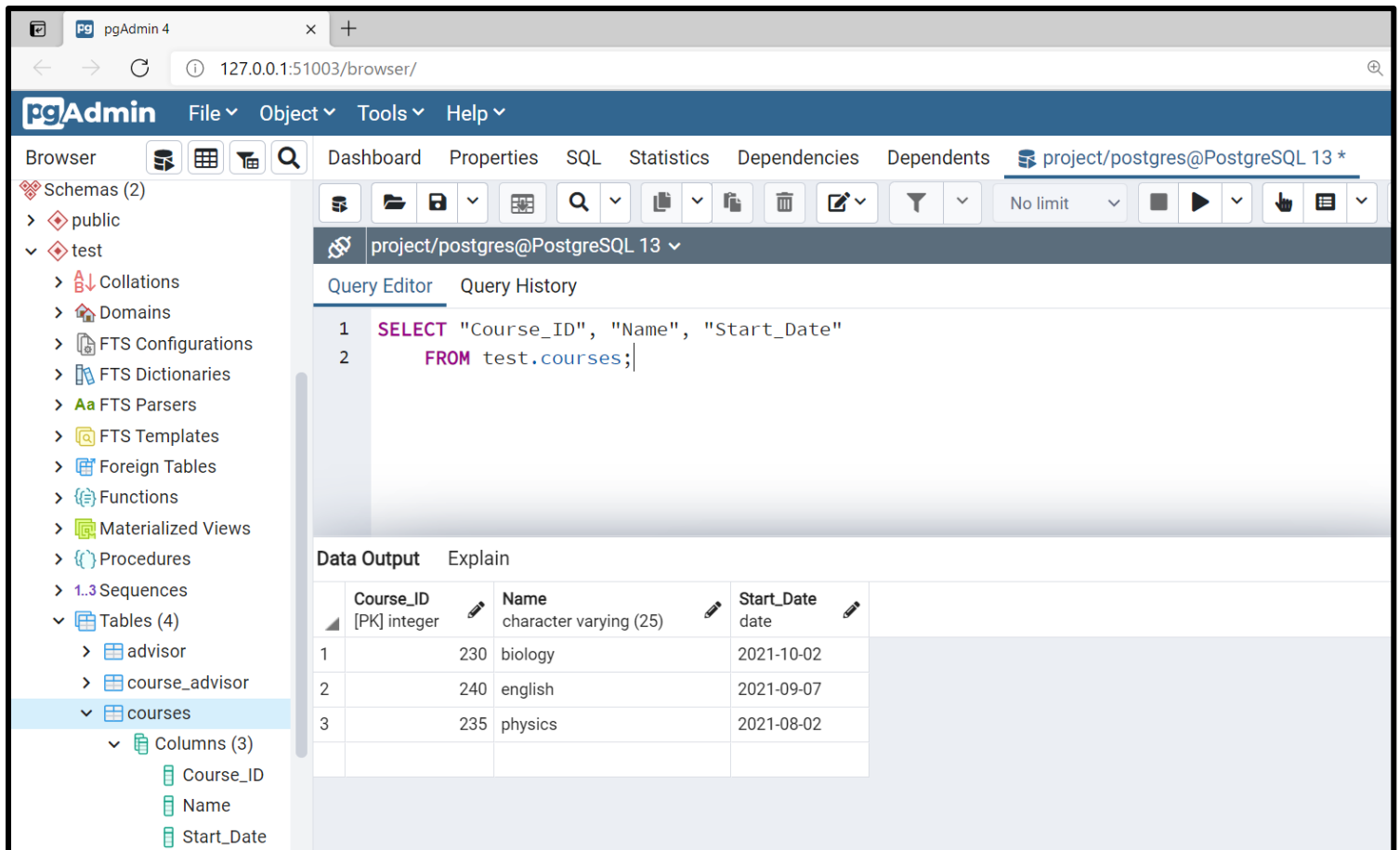


Figure 7: Displaying the data in *courses* table by using SELECT keyword

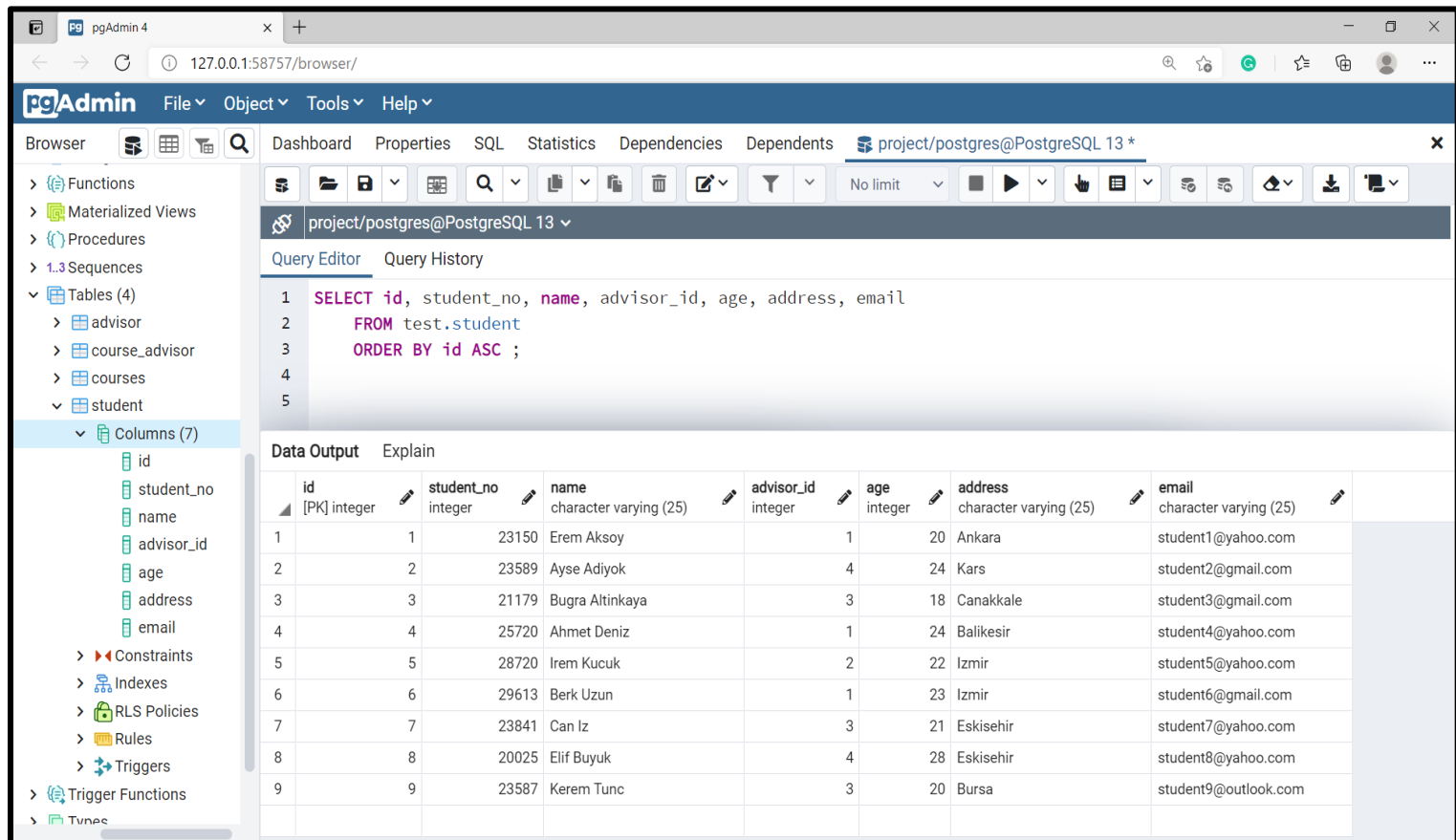


Figure 8: Displaying the data in *student* table by using SELECT keyword

9.2 Building a Maven Project in Spring Initializr

I have preferred to use the Maven project to create my web application in Spring Initializr. The reason why I have chosen Maven is that the Eclipse IDE has support for the Maven build. It provides an editor to adjust the pom file and downloads the required dependencies. There is a part called dependencies on the right side of the Spring Initializr page (see Figure 4). It asks us to specify what kind of functionality we want for the application. I have chosen Spring Web (which includes RESTful) option as I wanted to have a web application. I have also selected Spring Data JPA option as my application is supposed to be backed by a relational database accessed with JPA. After choosing dependencies, I have clicked the 'Generate' button at the left bottom, and the zip file is downloaded. By unzipping the file, the application is ready to open in Eclipse.

9.3 Creating a Maven Project in Eclipse

After creating the application by using Spring Initializr, we can easily add it to Eclipse. To do that, I have followed these steps: File -> New -> Project -> Maven -> Maven Project. Then by right clicking on the project I have created, I have selected 'import' option, and followed these steps: Existing Maven Project -> Add External Jar File. After choosing this option, the Maven project is now available on Eclipse platform.

You can see the pom.xml file for my project called demo and the JPA dependency that I have added in Spring Initializr, and Maven dependencies in Figure 6 below:

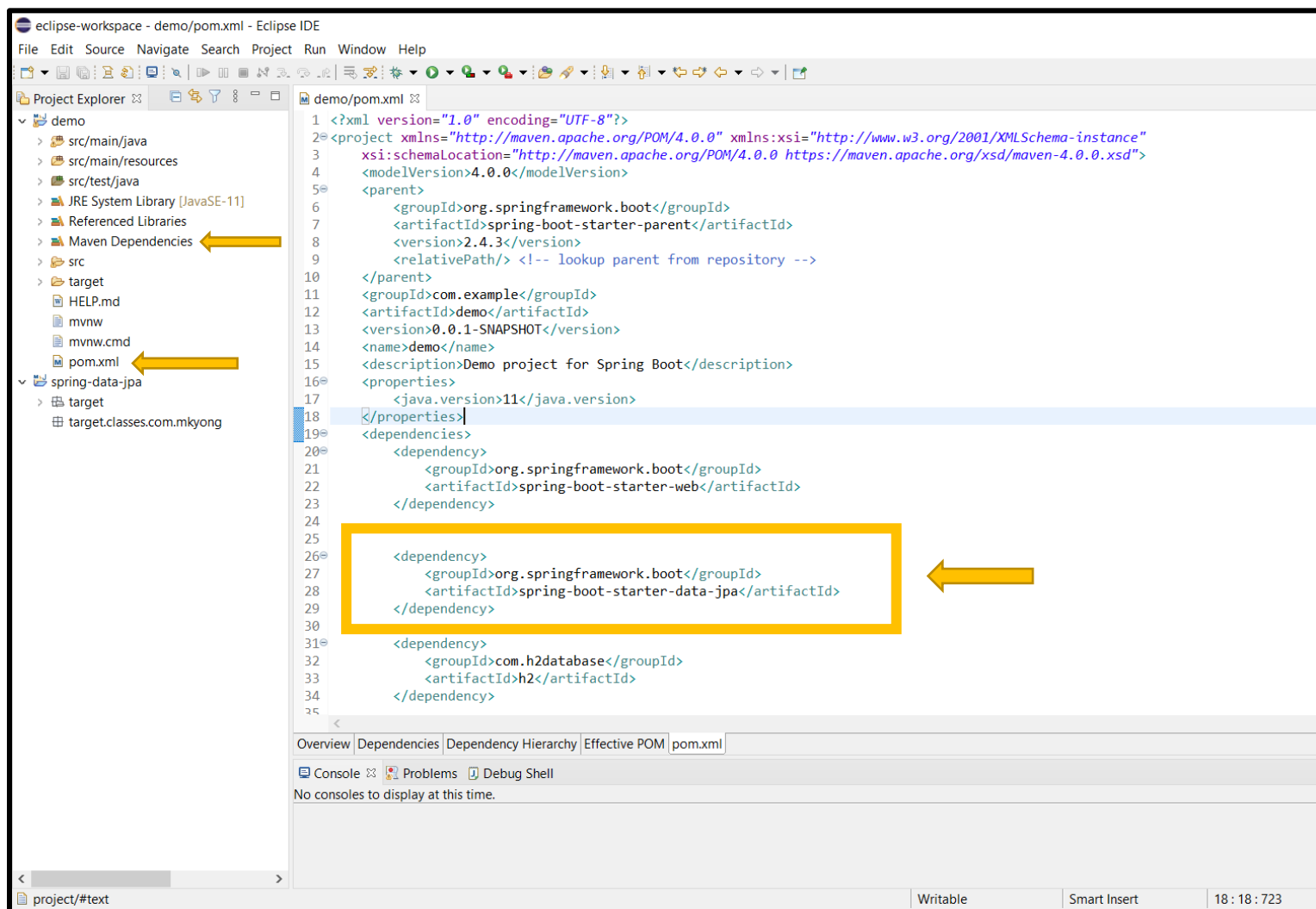


Figure 9: Project demo and its pom.xml file with JPA dependency added

9.4 Creating Packages, Classes and Objects

Java is an object-oriented language, and in object-oriented languages, everything is associated with classes and objects, meaning that the program is designed by classes and objects. Before moving on to the classes and objects in my projects, I want to summarize some basic information about Java's key terms. You can check the website [13] for more detailed information.

A class is a blueprint from which objects are created, so an object is a class instance. A class in Java can contain fields, methods, constructors, blocks, and interfaces. Class is a logical entity, and it cannot be physical.

An object is an entity that is created from a class. It can be logical or physical (tangible and intangible).

A method is a collection of instructions that performs a particular task or operation. It is reusable, so once it is created, it can be used many times, making the method time-saving. It is executed only when it is called. Java provides four different access types of the method, but I will only explain two of them as I have used these two in my project. The first one is *public*, accessible by all classes, and the second one is *private*, accessible only in the classes where it is defined.

The static keyword is mainly used for memory management to save memory. It can be used with variables, methods, blocks, and classes. Using the static class is a way of grouping classes, and it is also used to access the primitive member of the enclosed class through the object reference. As it is also clustered the objects with common attributes, this field gets the memory only once, making the program memory effective. Using the static keyword for objects, we do not need to 'new' anything, which the constructor usually does. If we do not make it static, we need to define everything before calling the method, so it is very beneficial to use the static keyword, but we need to be careful to decide where to use it. It can be used for methods as well, and one advantage of a static method is that it can be called without creating an object. It can also access static data fragments and alter their value.

An instance variable is a variable created inside the class, but it is known as an instance variable outside the class. It gets the memory at runtime, not at compile time.

The instance method is a non-static method that is defined in the class. An object of its class should be created before calling the instance method.

Constructor is a specialized method that is utilized to initialize the object. So, it is used to provide initial values to the instance variables in the class. Constructors are called every time an object is created by using the new keyword. *The new keyword* is used to allocate memory at runtime, and all objects obtain memory in the Heap

memory area. Here, it is essential to say that all classes have constructors even though there is no constructor available in the class. It is because Java provides a default constructor automatically, and by calling it, the default constructor initializes all variables to zero. We also need to know that the default constructor is no longer used once we define our constructor.

A dependency: Most of the time, an object performs its operations with objects of other classes, and it is called object dependencies.

An injection: It is the process of supplying the demanded dependencies to an object.

The dependency injection aids to implement inversion of control (IoC), meaning that object creation responsibility and injecting dependencies will be given to the framework, which is Spring in my case. There are three ways to handle dependency injection in Spring Boot and in Java in general; constructor-based injection, setter-based injection, and field-based injection.

Constructor injection: The required dependencies for the class are provided as arguments to the constructor. If we have more than one constructor for the class, we need to add an `@Autowired` annotation (see 9.5 for the detailed information) so that Spring can understand which constructor to use to inject the dependencies. Constructor injection is way safer than field injection as it does not allow the dependencies to change after setting them. Moreover, they can be *final*, so it cuts the physical connection which is very beneficial for the safety.

Setter injection: The required dependencies are provided as field parameters to the class, and the values are set using the setter methods. We need to add an `@Autowired` annotation.

Field injection: The required dependencies are allocated directly to the fields by Spring with `@Autowired` annotation. The real dependencies are hidden from the outside, and immutable objects cannot be created.

Before we create our classes and objects, we need to create a main and add some annotations before main, outside the class for Spring Boot. The required annotations are `@SpringBootApplication`, `@EnableAutoConfiguration`, and `@ComponentScan` (see 9.5 for detailed information about the annotations). You can look at Figure 10 below to see the imported version.

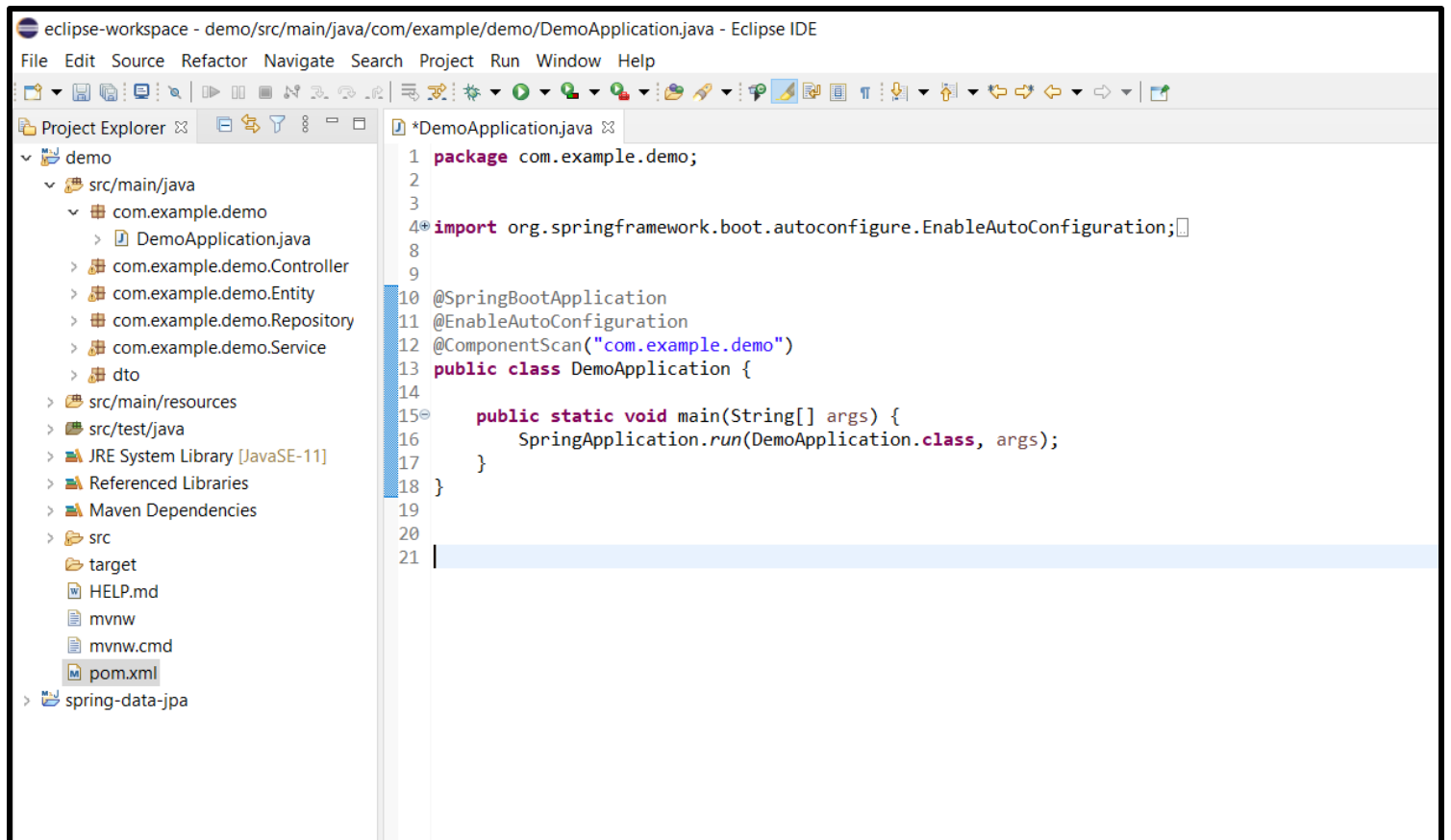


Figure 10: DemoApplication / Main of the project

After adding the necessary annotations, we can create our classes and objects. Firstly, I have created a package called Entity to create classes for all the tables I have in the *project* database. After that, I have created separate classes for all entities and then the objects in these classes. For each class, @Entity annotation should be added before the class implementation to show that the class is an entity and is mapped to a table in the database. Also, to map the table in the database, @Table annotation should be added, and the name of the schema and the table need to be specified for the @Table annotation inside the braces. As the next step, the primary key of an entity needs to be added; for example, for all the tables, primary keys are id, so the annotations should be in @Id format.

I have added all the required annotations and generated Setters and Getters by right-clicking on the class name and finding 'Generate Setters and Getters' from the 'Source' option. You can find how I have created all the necessary classes and objects for the tables below.

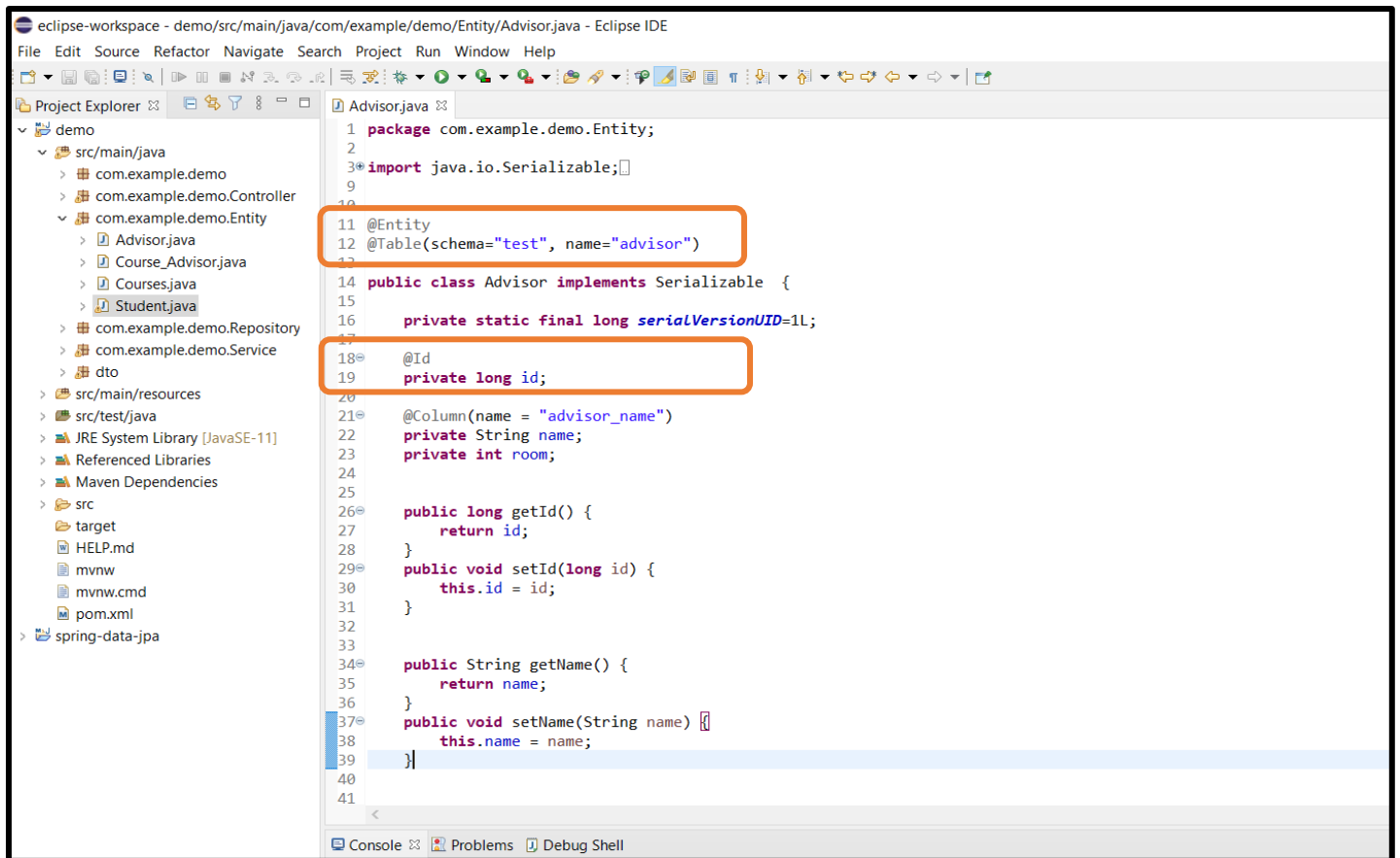


Figure 11: *Advisor* class and necessary annotations

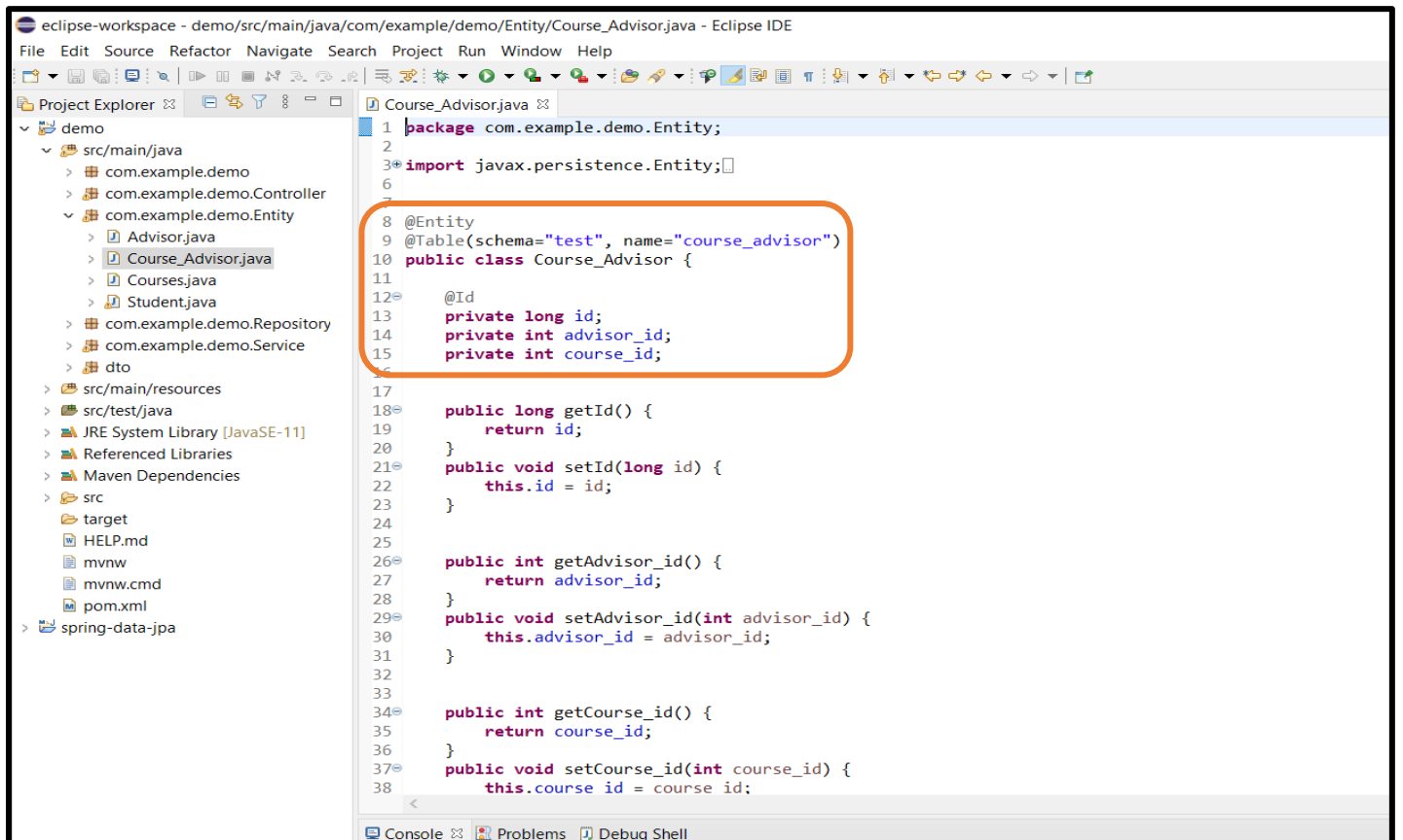


Figure 12: *Course_Advisor* class and necessary annotations

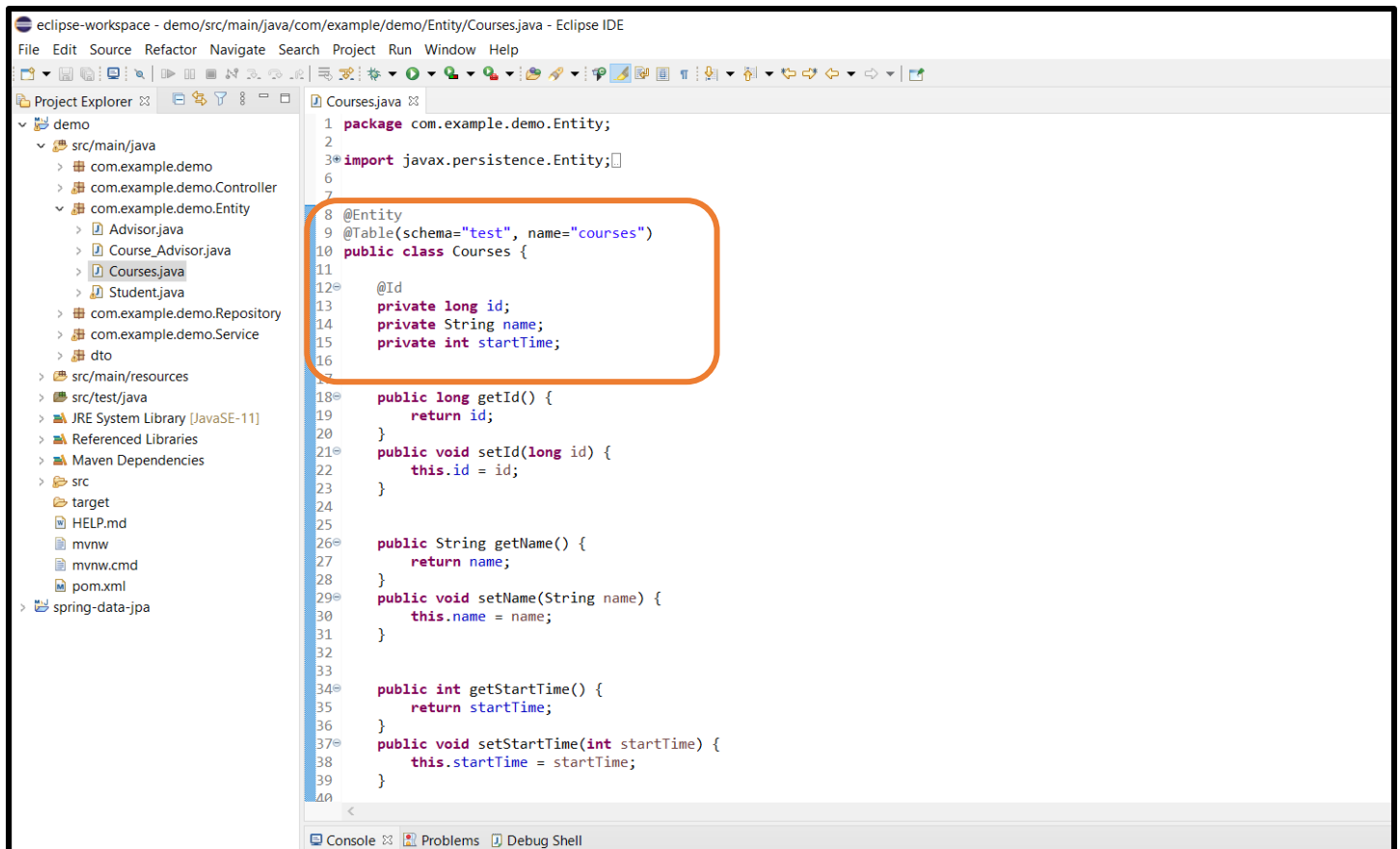


Figure 13: *Courses* class and necessary annotations

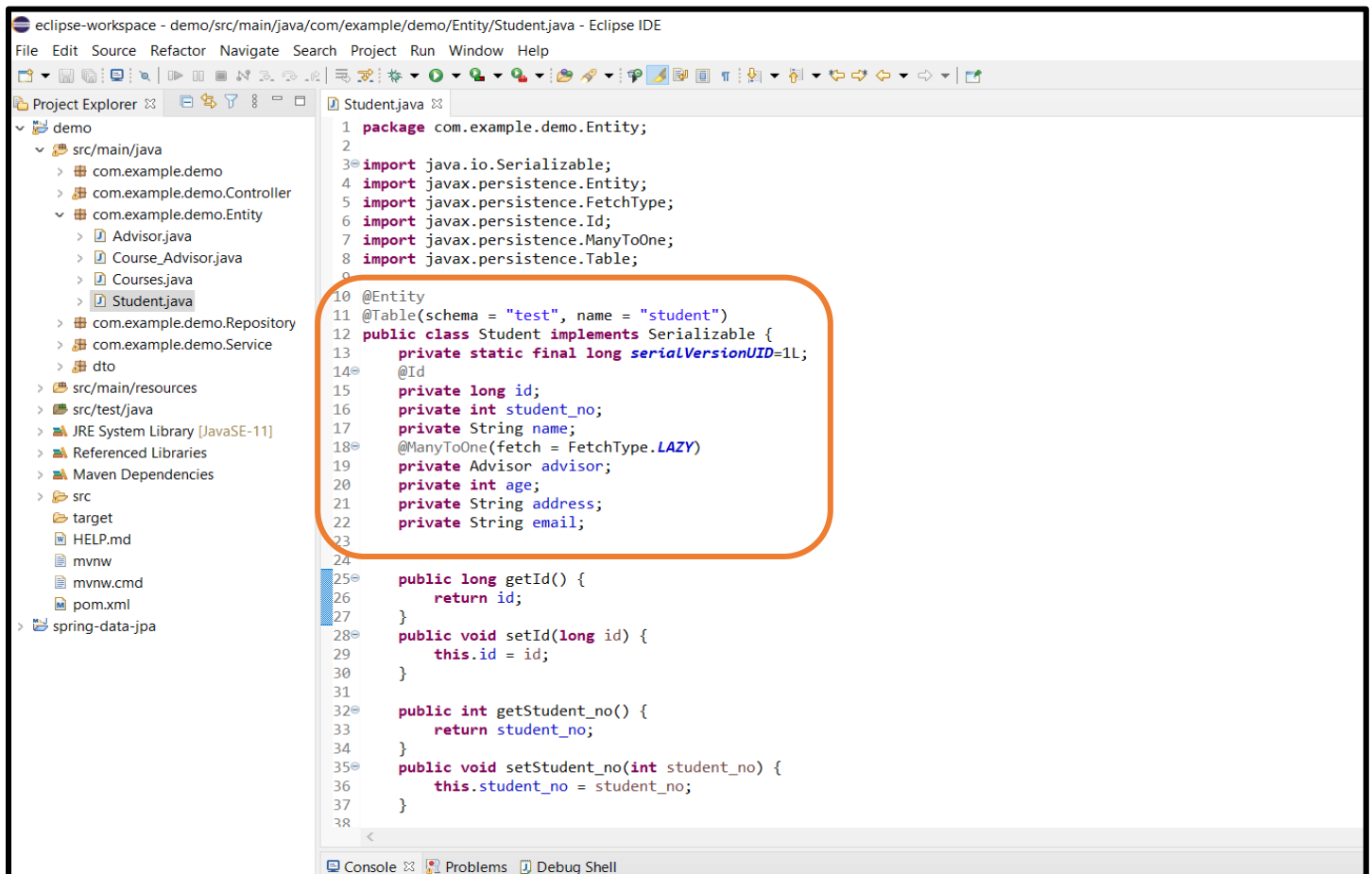


Figure 14: *Student* class and necessary annotations

In the *student* class, it can be seen that there is an extra annotation, which is `@ManyToOne`. This annotation indicates the connection between the *student* and *advisor* table, meaning that many students can have the same advisor, but one student cannot have more than one advisor. For this annotation, we can use either Eager Loading or Lazy Loading. Eager and Lazy are two types of data loading strategies in ORMs, such as Hibernate and Eclipse. When we have one entity that has references to other entities (like student and advisor), we use these data loading strategies.

Eager Loading: Data initialization occurs on the spot, and it can be used when the relations are not too much. However, it is associated as a bad practice in Hibernate as it can result in many data being fetched from the database and stored in the memory, so it can consume so much memory and also affect the performance.

Lazy Loading: It does not initialize any data and load into memory until an explicit call is made to it, so it has less memory consumption. Nevertheless, it might cause a delayed initialization which impacts performance. I have used LAZY loading to fetch the data because I wanted to get the result quickly and save memory.

9.5 Spring Boot Annotations

The primary purpose of Spring Boot annotations is to provide auxiliary data information about a program. So, it is not a part of the application we have developed, and they cannot alter the way the compiled program works. The core annotations for Spring Framework are listed below as (also see Figure 10):

- **@SpringBootApplication:** It allows Spring Boot autoconfiguration and component scanning. It encapsulates `@Configuration`, `@EnableAutoConfiguration`, and `@ComponentScan` annotations with their default attributes.
- **@EnableAutoConfiguration:** It enables auto configuration, meaning that it tells Spring Boot to start search for beans and automatically add them based on the classpath settings. So, when we use this annotation, we do not need to define any XML Configuration as auto configurator will do that.
- **@ComponentScan:** It enables Spring Boot to know which packages to scan for beans, which are annotated components, so we need to add the package name inside the braces.
- **@Autowired:** It is used on setter methods, constructors, and instance variables to inject object dependency utterly by matching data type. In this version of Spring I have used, it is optional to use this annotation for the classes that have only one constructor.

9.5.1 Spring Framework Stereotype Annotations

- **@Component:** It is a class-level annotation and used to sign a class as a bean. It makes the annotated class easy to find during the classpath. The Spring Framework detects and records only custom beans, which are with @Component annotation, and does not look for others. By using this annotation, Spring can inject any specified dependencies into the classes annotated with @Component. However, we can use this annotation only when a class's source code is editable, meaning that it is not possible to use @Component on classes that we do not have the source code. Instead, we should use @Bean annotation, which is at the method level. Spring also provides some specialized stereotype annotations, such as @Controller, @Repository, and @Service. They all use @Component as a function, but the reason they are specialized is that there are some areas that Spring searches for specialized annotations specifically to provide extra automation profits.

→ **@Controller:** It is often used to serve web pages as a request handler, so it marks the class as a web controller and passes user input data to service. It is generally used with @RequestMapping annotation.

→ **@Repository:** It is a DAOs (Data Access Object) that access the databases directly, so it interacts with the data source to perform all the operations related to the database. Its main job is to catch specific exceptions and rethrow them as a Spring's unified unchecked exception.

→ **@Service:** It is called the business logic that helps to create a class that implements an interface and stores, retrieves, deletes, and updates the data. It provides the methods, and when we use @Autowired annotation with the interface, it calls the methods. The benefits of using @Service annotation are that it makes the code easier to read and debug, and it marks the class as a service provider. It gathers data from controller and after performing validations, calls repositories for data manipulation.

We can summarize the relation between these specialized components as below:

CONTROLLER *calls* **SERVICE(S)** *who calls* **REPOSITOR(IES)**

I have created separate packages for Controller, Repository, and Service. I have added two classes in each of them (except Service); one for advisor and one for the student object. I have also created some random methods to play with the data and see how differently they work in different methods. You can see the methods written in Controller classes and the content of each package below:

```

1 package com.example.demo.Controller;
2
3 import java.util.List;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.web.bind.annotation.DeleteMapping;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PostMapping;
8 import org.springframework.web.bind.annotation.RequestBody;
9 import org.springframework.web.bind.annotation.RequestMapping;
10 import org.springframework.web.bind.annotation.RequestParam;
11 import org.springframework.web.bind.annotation.RestController;
12
13 import com.example.demo.Entity.Student;
14 import com.example.demo.Service.StudentService;
15
16 import dto.StudentDTO;
17
18
19 @RestController
20 @RequestMapping("/student")
21
22 public class StudentController {
23
24
25     @Autowired
26     StudentService studentService;
27
28     @PostMapping(value="/save")
29     public Student save(@RequestBody Student student) {
30         return studentService.saveStudent(student);
31     }
32
33     @GetMapping(value="/list") //Listing the students
34     public List<Student> getStudentList() {
35         return studentService.getStudentList();
36     }
37
38
39     @GetMapping(value="/idbulma") //Find a student by id
40     public StudentDTO getStudentbyID(@RequestParam("id") long id) {
41         return studentService.getStudentById(id);
42     }
43
44     @GetMapping(value="/advisorbulma") //Find a student by advisor
45     public List<Student> getStudentbyAdvisor(@RequestParam("name") String name){
46         return studentService.getStudentByAdvisor(name);
47     }
48
49     @PostMapping(value="/update") //Update the address of a student
50     public Student updateAddress(@RequestBody Student student) {
51         return studentService.saveStudent(student);
52     }
53
54     @DeleteMapping(value="/delete") //Delete a student
55     public Student delete(@RequestParam("id") long id) {
56         return studentService.deleteStudent(id);
57     }
58 }
59

```

Console Problems Debug Shell

Figure 15: StudentController class

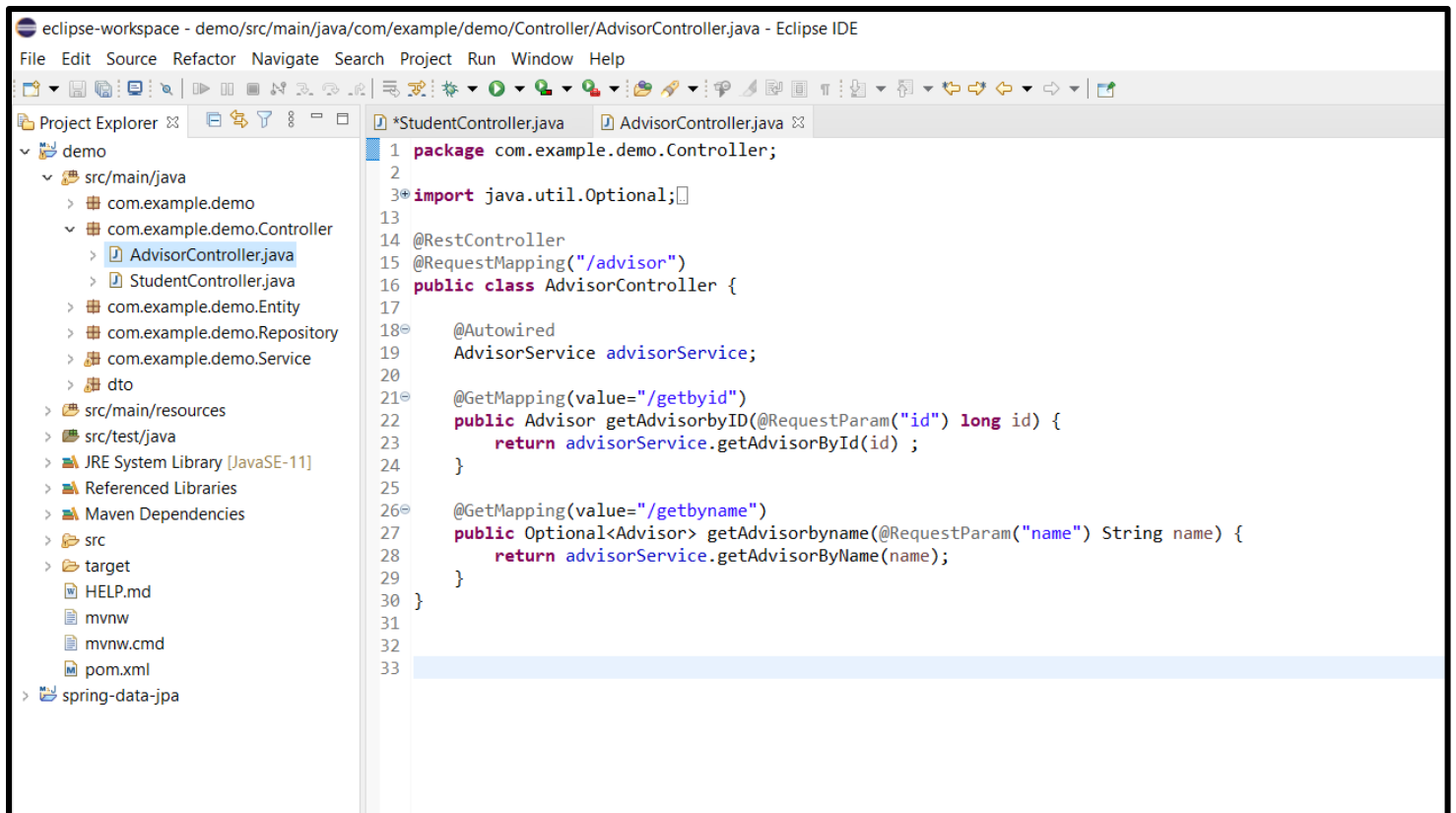


Figure 16: AdvisorController class

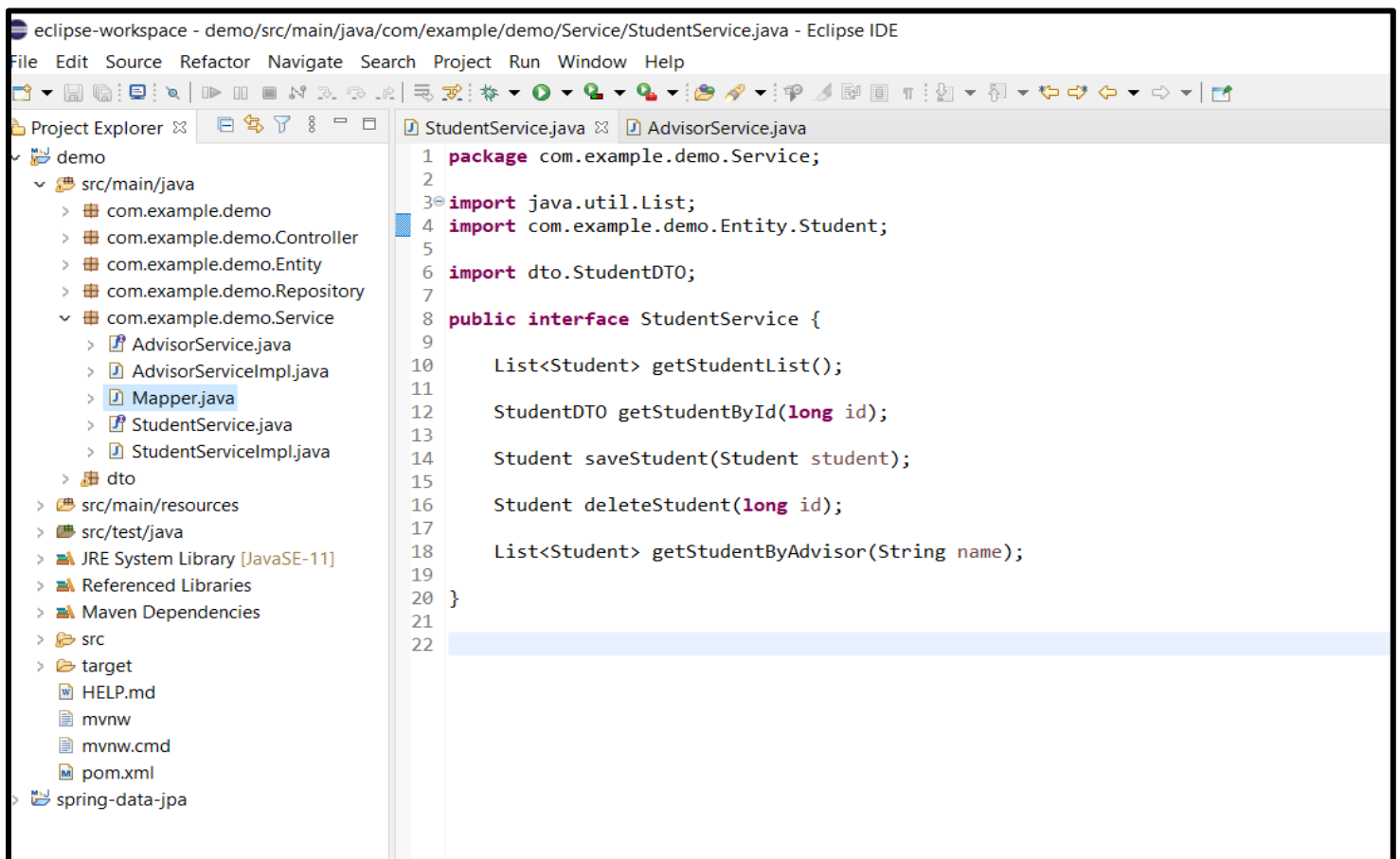


Figure 17: StudentService interface

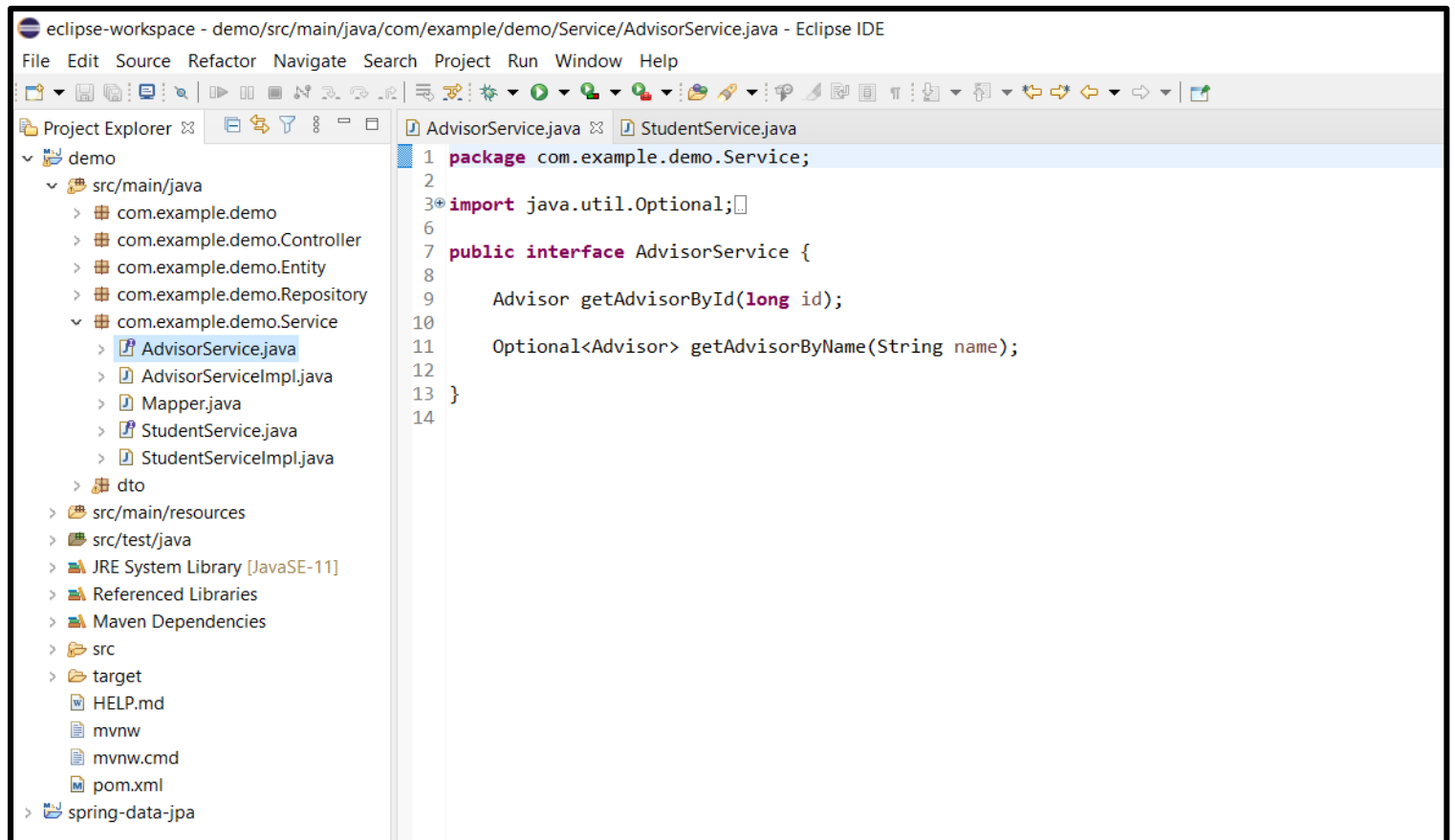


Figure 18: AdvisorService interface

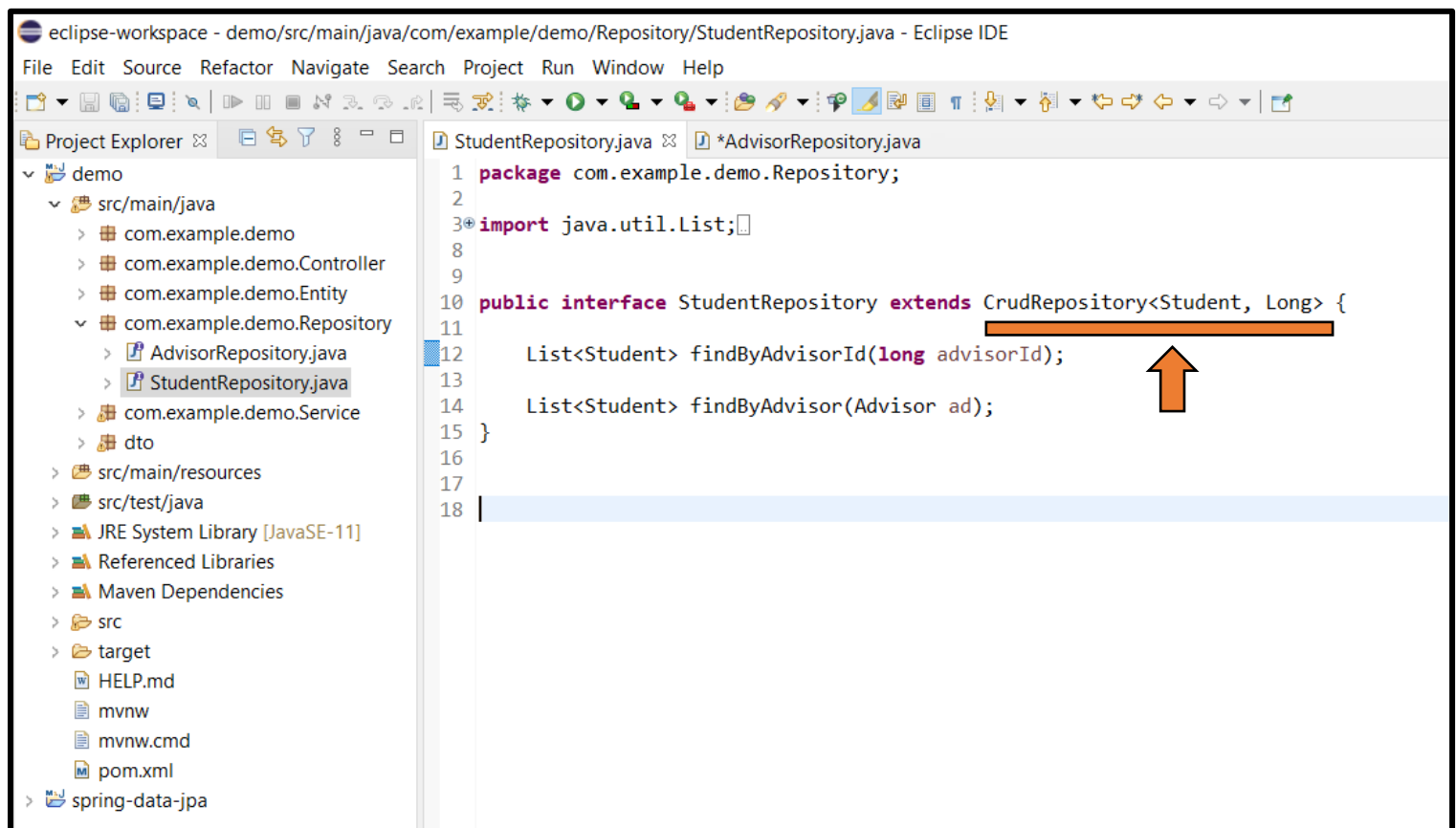


Figure 19: StudentRepository interface

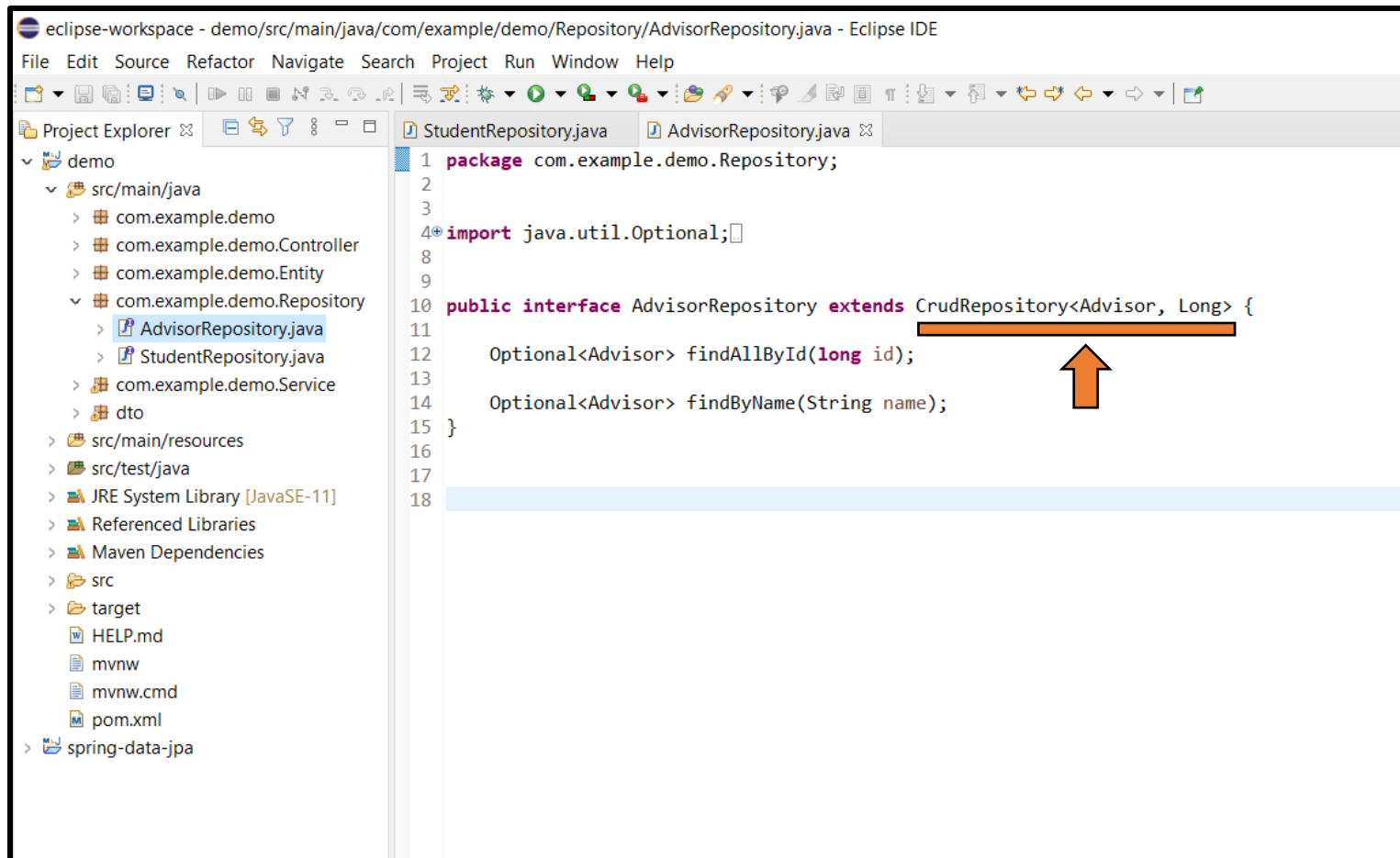


Figure 20: AdvisorRepository interface

Service interfaces (Figures 17 and 18) allow us to create an interface and define the methods we use in Controller classes (Figures 15 and 16). This way, the Service can hold the definitions for the methods, so it gathers data from Controller and calls repositories for data manipulation.

There is a statement in the Repository interfaces saying 'CrudRepository ...'. This statement holds the information about the entity and its corresponding primary key. The first one (Student & Advisor) is the name of the entity and the second one (Long) is the data type of the primary key of this entity. I basically entries the database and contacts with the data. The operations defined there belong to the Service implementation classes of advisor and student (see 9.8.1), and it shows that Repository is called by Service and then operates the requirements.

9.5.2 Spring REST Annotations

- **@RequestController:** It is a specialized version of the controller and is a combination of @Controller and @ResponseBody annotations. It streamlines the controller implementation, and there is no need for @ResponseBody annotation when @RequestController is annotated.
- **@RequestMapping:** It is used to map the web requests, and it can be used with both classes and methods. We need to add this annotation to the controller class as it is where the request is taken from. @RequestMapping have several variants specified for particular requests. According to the request we send from Postman, we need to add appropriate mapping annotation to be sure the requests are matching on both platforms. I have used @GetMapping, @PostMapping, and @DeleteMapping.
 - **@GetMapping:** It does not update the data, so we do not need to send any value from Postman, we just need to send the request directly. I have used 'GET' request for searching particular value: for example, printing the student list, finding a student with specific id or an advisor.
 - **@PostMapping:** It enables to update the data, so we need to specify the parts that we want to update on Postman and then send the request. I have used 'POST' request to save a new student and update any information of a student. It is also possible to update many data of a particular student or more than one student with one request. I have also observed that when we do not send all the values (all the columns including the one we want to update specifically), it assigns a 'NULL' value for the columns that we have not entered any value. It does not take the necessary information itself so it means that we should send all the information with the one we want to update as an input from Postman.
 - **@DeleteMapping:** It deletes the data requested. I have used "DELETE" request to delete a certain student.
- **@RequestBody:** It is used to bind the incoming request to the specific parameter that we indicate. We use this annotation when we send an object as a parameter as @RequestBody converts the request to an object by HTTP MessageConverter.
- **@RequestParam:** It is used to extract the query parameters from the URL. It can specify default values if the query parameter is not present in the URL.

You can check the website [14] for more annotations.

9.6 Connecting to Database via Eclipse

To connect to the database, I have followed these steps:

- Go to src/main/resources,
- Click application.properties,
- Enter all the necessary information,
- Save the changes,
- Connection to the database is completed.

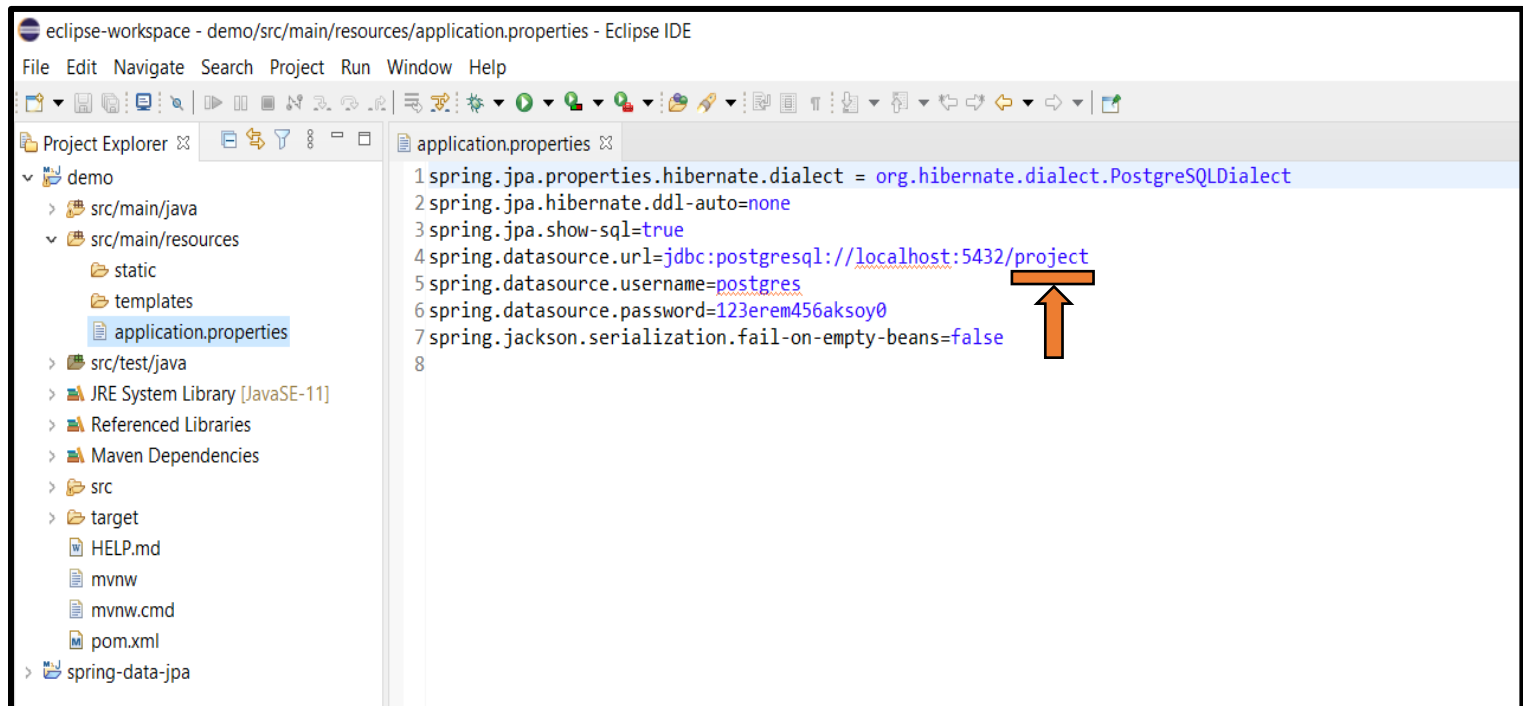


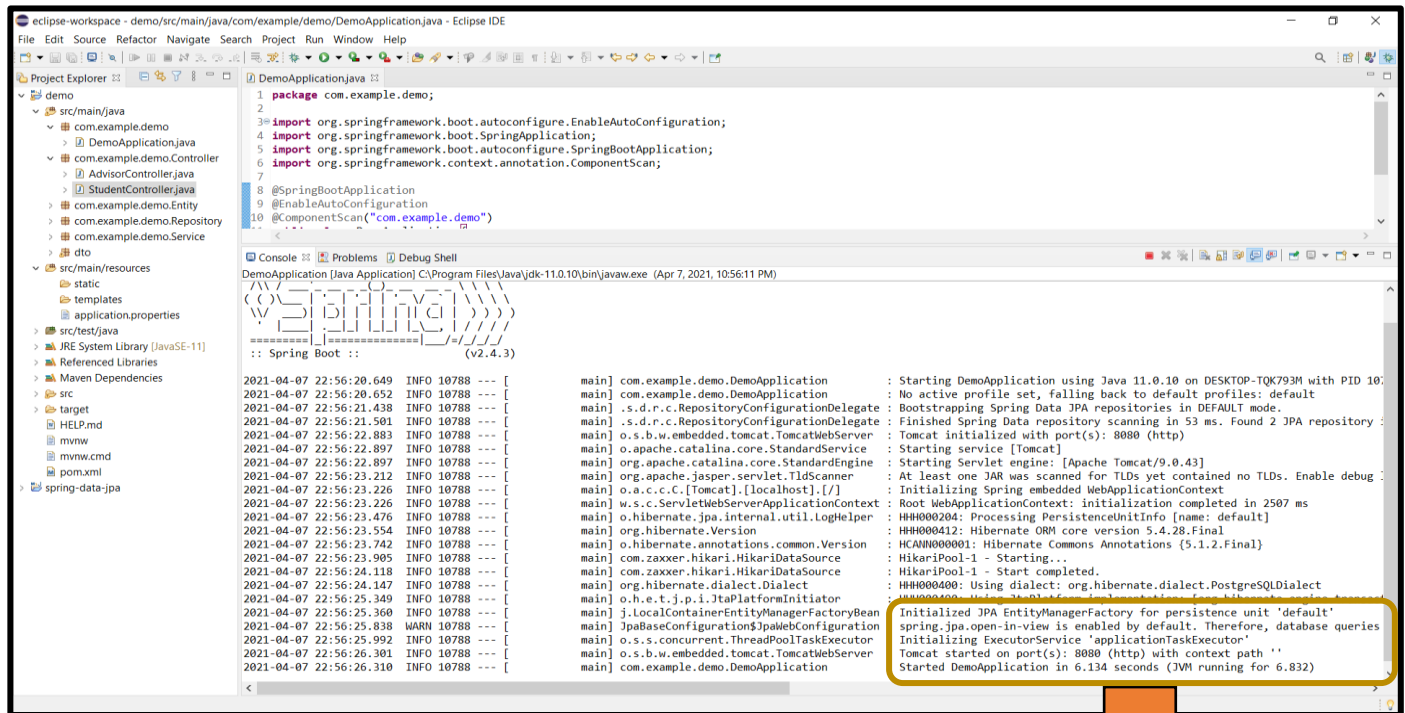
Figure 21: To connect to the database: application.properties

In figure 21, the name 'project' refers to the database that we want to connect. As I have created this database and worked on it, I have connected to this one. It was 'postgres' by default, so I needed to change it to 'project'. I also needed to enter the password and all the other information was already entered, but to check if they are correct, I have followed these steps:

- Go to pgAdmin,
- Open the 'Servers' part,
- Select the server we are connected (PostgreSQL 13),
- Right click and go to properties,
- Go to Connection section,
- Check the entries there with the ones in Figure 21.

9.7 Sending Request from Postman

After connecting to the database successfully, we need to send the request via Postman. Before sending the request, we need to run the program (DemoApplication.java) and be sure that we have connected to the database. After clicking the run button, we can track the steps by opening the console and go to Postman after seeing that our connection is made successfully. You can see how it looks like after connecting in Figure 22 below:



```
1 package com.example.demo;
2
3 import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
4 import org.springframework.boot.SpringApplication;
5 import org.springframework.boot.autoconfigure.SpringBootApplication;
6 import org.springframework.context.annotation.ComponentScan;
7
8 @SpringBootApplication
9 @EnableAutoConfiguration
10 @ComponentScan("com.example.demo")
11 public class DemoApplication {
12     public static void main(String[] args) {
13         SpringApplication.run(DemoApplication.class, args);
14     }
15 }
```

```
2021-04-07 22:56:20.649 INFO 10788 --- [main] com.example.demo.DemoApplication : Starting DemoApplication using Java 11.0.10 on DESKTOP-TQK793M with PID 10788
2021-04-07 22:56:20.652 INFO 10788 --- [main] com.example.demo.DemoApplication : No active profile set, falling back to default profiles: default
2021-04-07 22:56:21.438 INFO 10788 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2021-04-07 22:56:21.501 INFO 10788 --- [main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 53 ms. Found 2 JPA repository :
2021-04-07 22:56:22.883 INFO 10788 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-04-07 22:56:22.897 INFO 10788 --- [main] o.apache.catalina.core.StandardEngine : Starting service [Tomcat]
2021-04-07 22:56:23.212 INFO 10788 --- [main] org.apache.jasper.servlet.TldScanner : At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for this logger to get more detail
2021-04-07 22:56:23.476 INFO 10788 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-04-07 22:56:23.742 INFO 10788 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2507 ms
2021-04-07 22:56:23.754 INFO 10788 --- [main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2021-04-07 22:56:23.905 INFO 10788 --- [main] com.zaxxer.hikari.HikariDataSource : HHH0000412: Hibernate ORM core version 5.4.28.Final
2021-04-07 22:56:24.118 INFO 10788 --- [main] org.hibernate.dialect.Dialect : HHH000000001: Hibernate Commons Annotations [5.1.2.Final]
2021-04-07 22:56:24.147 INFO 10788 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2021-04-07 22:56:25.349 INFO 10788 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HikariPool-1 - Start completed.
2021-04-07 22:56:25.360 INFO 10788 --- [main] j.LocalContainerEntityManagerFactoryBean : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
2021-04-07 22:56:25.838 WARN 10788 --- [main] JpaBaseConfiguration$JpaWebConfiguration : Initialized JPA EntityManagerFactory for persistence unit 'default'
2021-04-07 22:56:25.992 INFO 10788 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be executed during request processing (this will log this warning on every request, it can be disabled by setting spring.jpa.open-in-view=false)
2021-04-07 22:56:26.301 INFO 10788 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Initializing ExecutorService 'applicationTaskExecutor'
2021-04-07 22:56:26.310 INFO 10788 --- [main] com.example.demo.DemoApplication : Tomcat started on port(s): 8080 (http) with context path ''
2021-04-07 22:56:26.310 INFO 10788 --- [main] com.example.demo.DemoApplication : Started DemoApplication in 6.134 seconds (JVM running for 6.832)
```

Figure 22: Connection to the database

```
main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be executed during request processing (this will log this warning on every request, it can be disabled by setting spring.jpa.open-in-view=false)
main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
main] com.example.demo.DemoApplication : Started DemoApplication in 6.134 seconds (JVM running for 6.832)
```

Tomcat started on port(s): 8080 (http) with context path '' means that the connection is successful. Now we can send the requests via Postman. You can see the requests for all possible cases that I have created in the figures below:

http://localhost:8080/student/save

POST http://localhost:8080/student/save

Params Authorization Headers (8) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1 {
2   "id": 10,
3   "student_no": 22889,
4   "name": "Oyku Bozan",
5   "advisor": {
6     "id": 1,
7     "name": "Ahmet Kocak",
8     "room": 75
9   },
10  "age": 20,
11  "address": "Istanbul",
12  "email": "student10@outlook.com"
13 }

```

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 332 ms Size: 354 B Save Response

Pretty Raw Preview Visualize JSON

```

1 {
2   "id": 10,
3   "student_no": 22889,
4   "name": "Oyku Bozan",
5   "advisor": {
6     "id": 1,
7     "name": "Ahmet Kocak",
8     "room": 75,
9     "hibernateLazyInitializer": {}
10  },
11  "age": 20,
12  "address": "Istanbul",
13  "email": "student10@outlook.com"
14 }

```

Bootcamp Runner Trash

Query Editor Query History

```

1 SELECT id, student_no, name, advisor_id, age, address, email
2 FROM test.student
3 ORDER BY id ASC;

```

Data Output Explain

	id [PK] integer	student_no integer	name character varying (25)	advisor_id integer	age integer	address character varying (25)	email character varying (25)
1	1	23150	Erem Aksoy	1	20	Ankara	student1@yahoo.com
2	2	23589	Ayse Adiyok	4	24	Kars	student2@gmail.com
3	3	21179	Bugra Altinkaya	3	18	Canakkale	student3@gmail.com
4	4	25720	Ahmet Deniz	1	24	Balikesir	student4@yahoo.com
5	5	28720	Irem Kucuk	2	22	Izmir	student5@yahoo.com
6	6	29613	Berk Uzun	1	23	Izmir	student6@gmail.com
7	7	23841	Can Iz	3	21	Eskisehir	student7@yahoo.com
8	8	20025	Elif Buyuk	4	28	Eskisehir	student8@yahoo.com
9	9	23587	Kerem Tunc	3	20	Bursa	student9@outlook.com
10	10	22889	Oyku Bozan	1	20	Istanbul	student10@outlook.com

Figure 23: POST request to save a new student and updated table in the database

In Figure 23, to send the request properly, I have chosen 'POST' for the request type as 'save' method that I have created updates data, so the most proper request type for this method is POST. After I

have sent the request with table and method name (see the structure of the request), I have entered some random value for the student in JSON format and clicked ‘Send’ button then she has been added to the table in the database.

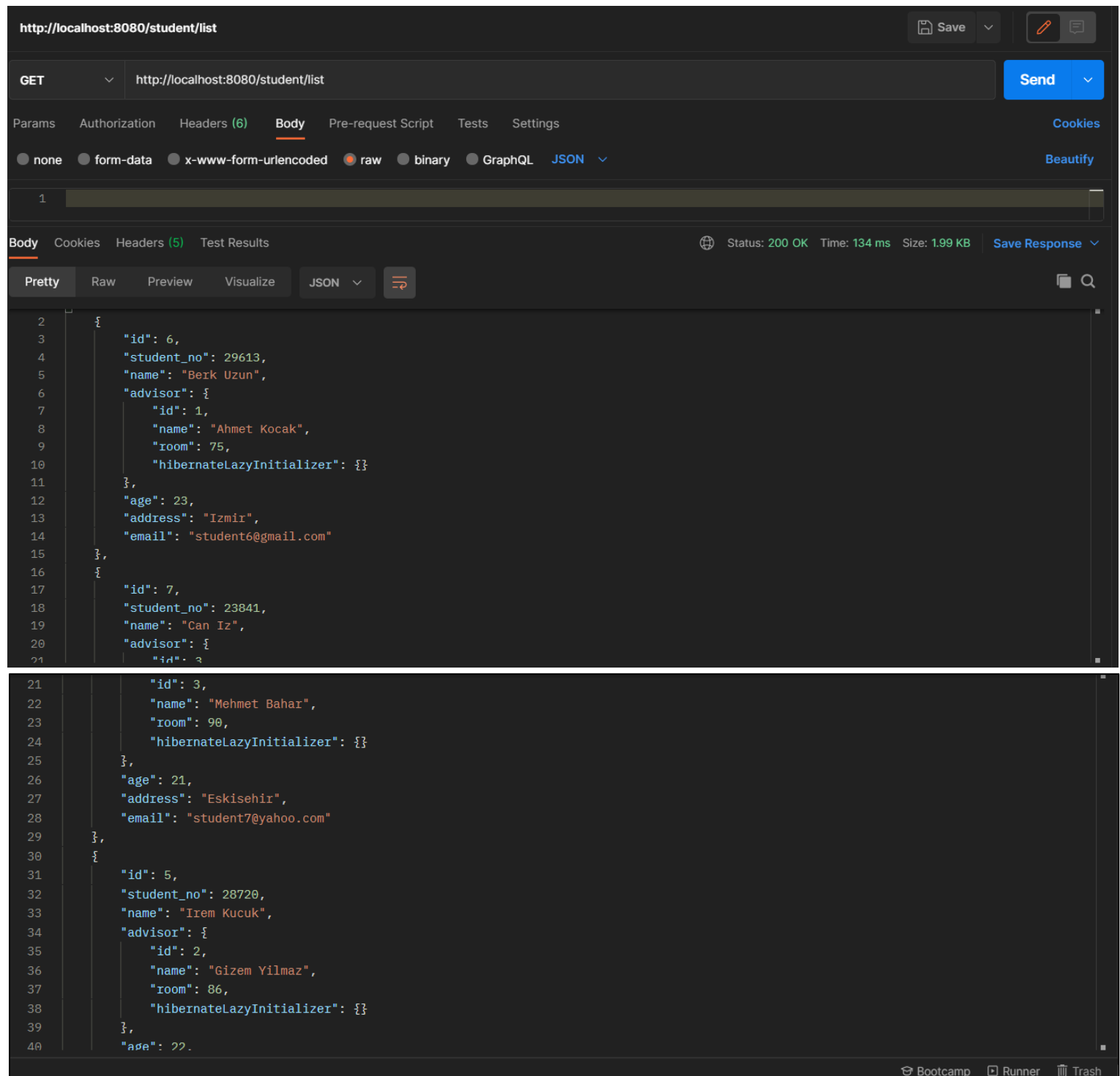


Figure 24: GET request to print the student list

For the GET request, again we need to specify the method name, which is list this time, then as it is a GET request, we do not need to write/send any information. As seen in Figure 24, I have managed to get the student list with all members in the order of actual table (not ascending order).

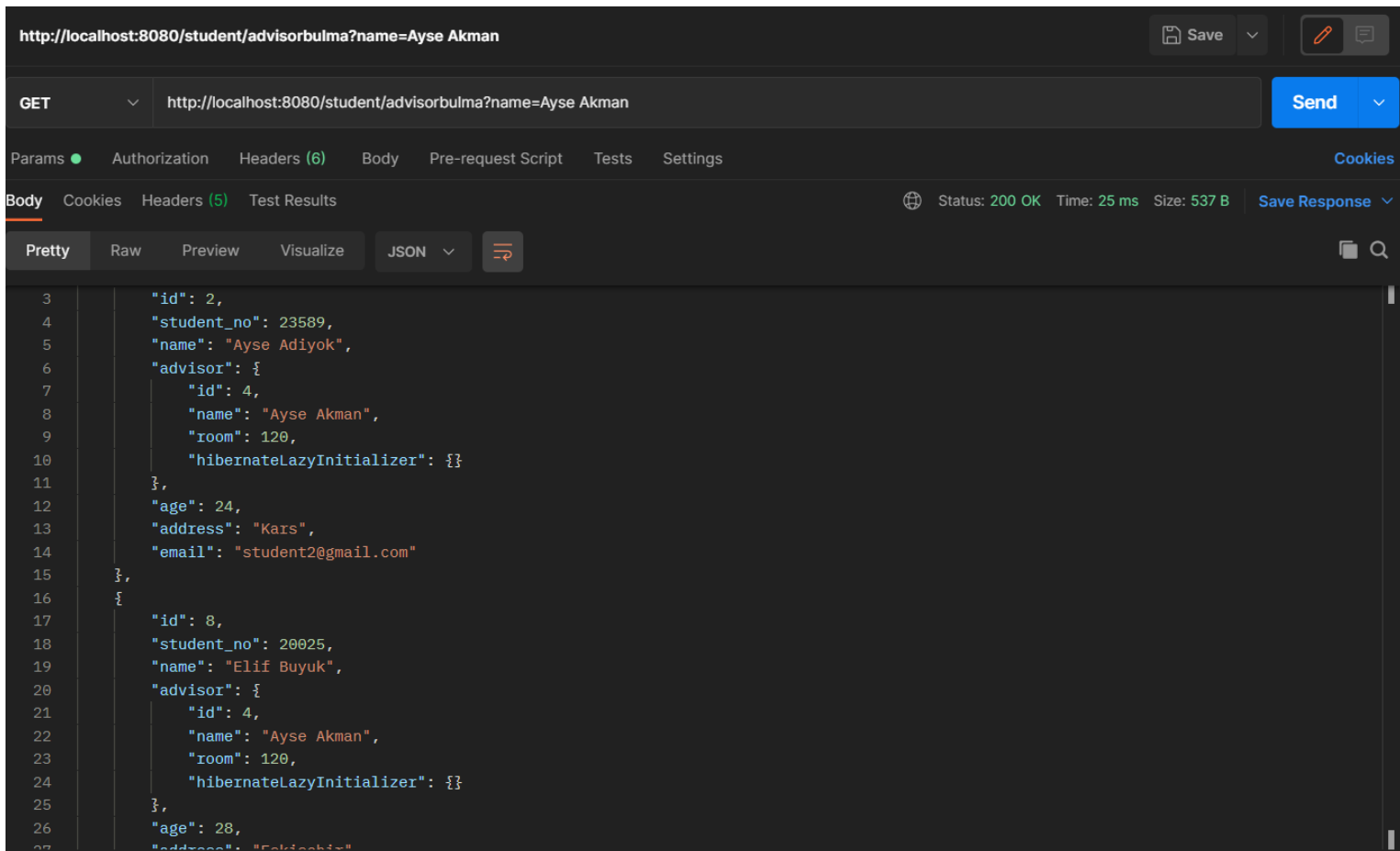
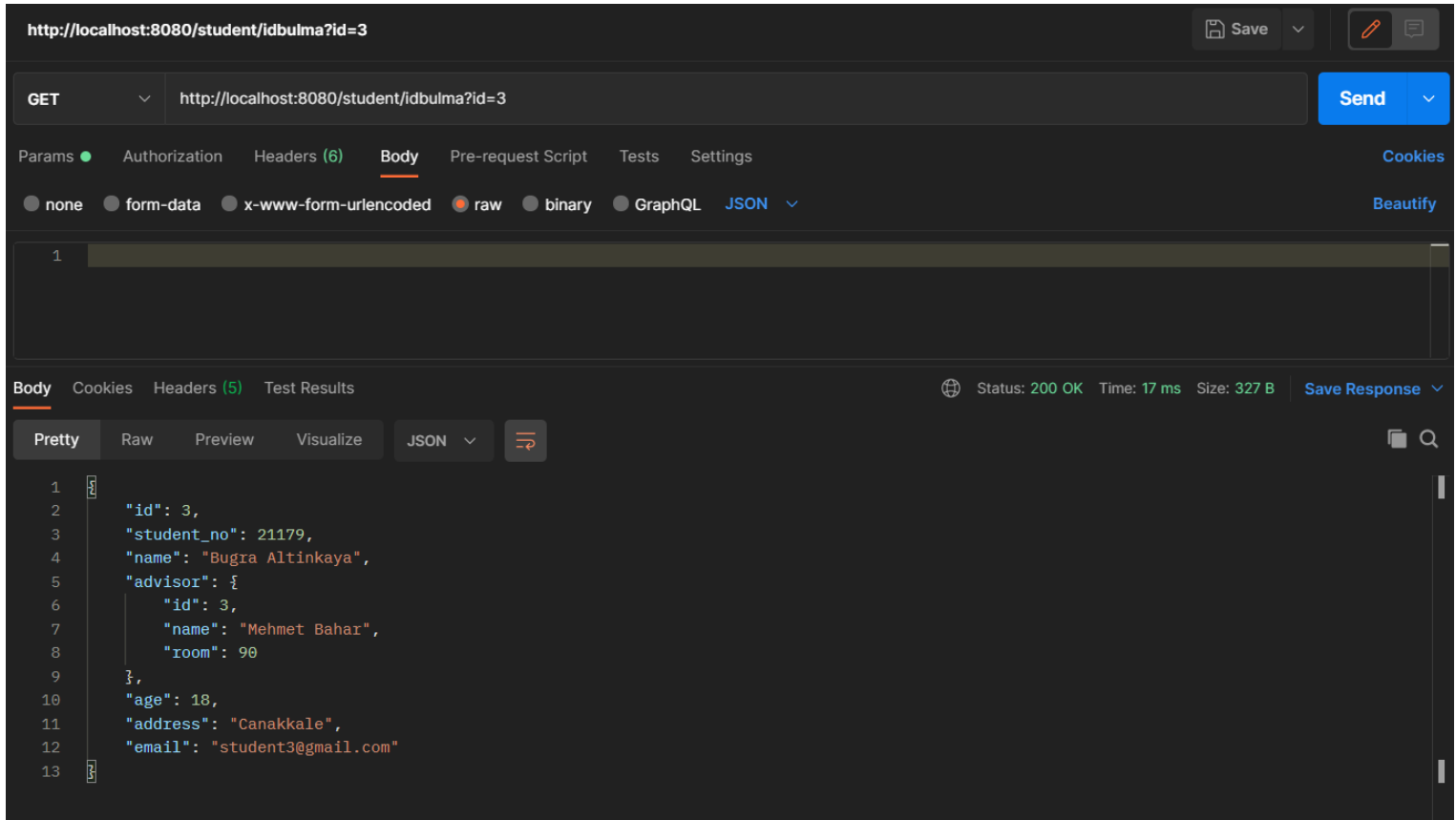


Figure 25: GET request to find student(s)

In Figure 25, I have sent two GET requests. One of them finds a student by the specific id (id=3) that I have entered, and the second request finds all the possible students according to their advisor name that is entered. It can be seen that the outputs are id=2 and id=8, which are the students of advisor called Ayse Akman (from the request).

http://localhost:8080/student/update

POST http://localhost:8080/student/update

Body

```

1 { "id": 10,
2   "student_no": 21111,
3   "name": "Oyku Bozan",
4   "advisor": {
5     "id": 1,
6     "name": "Ahmet Kocak",
7     "room": 75,
8     "hibernateLazyInitializer": {} },
9   "age": 21,
10  "address": "Ankara - Cankaya",
11  "email": "student10@outlook.com" }

```

Status: 200 OK Time: 18 ms Size: 362 B

Query Editor Query History

```

1 SELECT id, student_no, name, advisor_id, age, address, email
2 FROM test.student
3 ORDER BY id ASC;

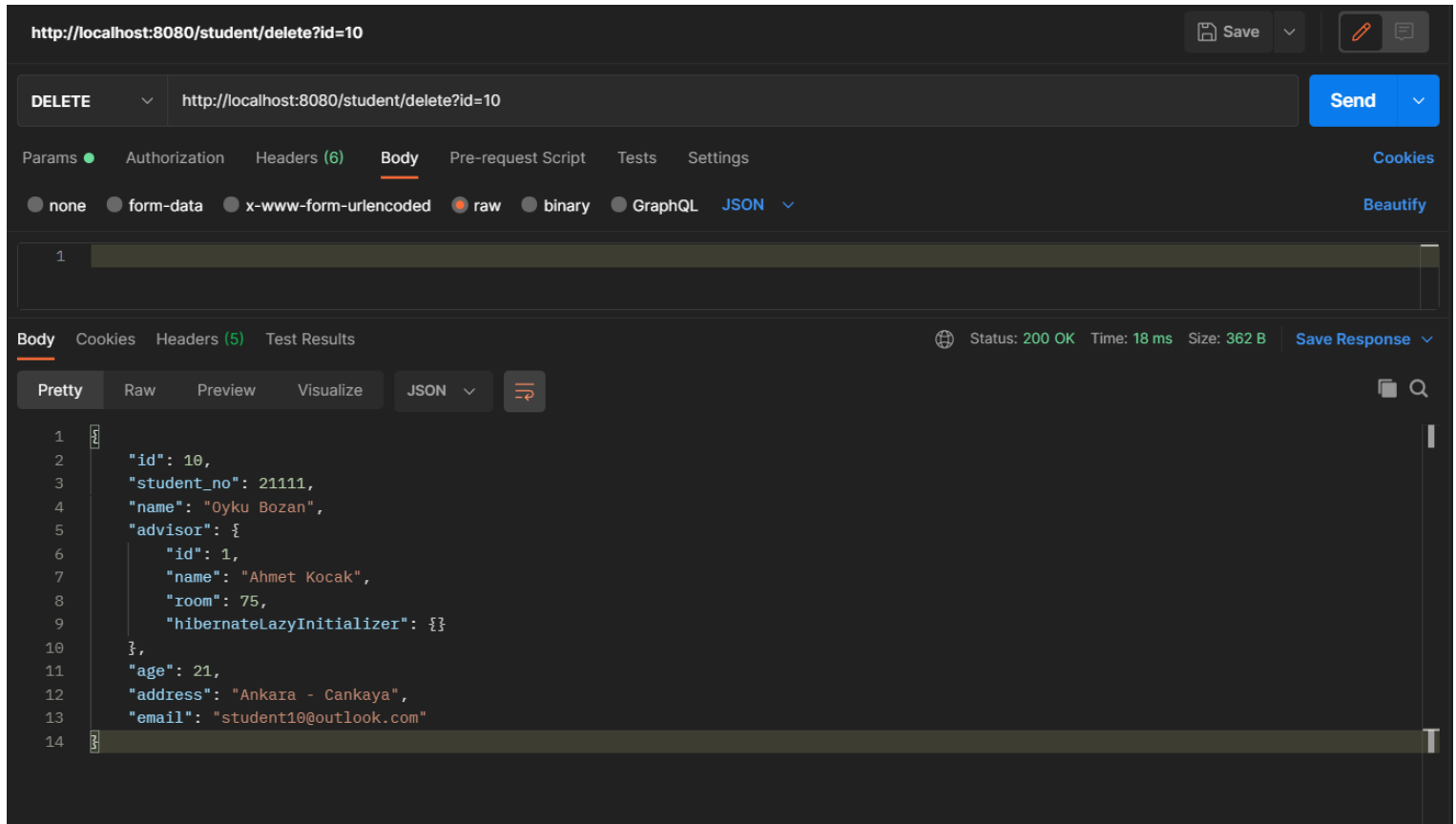
```

Data Output Explain

	id [PK] integer	student_no integer	name character varying (25)	advisor_id integer	age integer	address character varying (25)	email character varying (25)
1	1	23150	Erem Aksoy	1	20	Ankara	student1@yahoo.com
2	2	23589	Ayse Adiyok	4	24	Kars	student2@gmail.com
3	3	21179	Bugra Altinkaya	3	18	Canakkale	student3@gmail.com
4	4	25720	Ahmet Deniz	1	24	Balikesir	student4@yahoo.com
5	5	28720	Irem Kucuk	2	22	Izmir	student5@yahoo.com
6	6	29613	Berk Uzun	1	23	Izmir	student6@gmail.com
7	7	23841	Can Iz	3	21	Eskisehir	student7@yahoo.com
8	8	20025	Elif Buyuk	4	28	Eskisehir	student8@yahoo.com
9	9	23587	Kerem Tunc	3	20	Bursa	student9@outlook.com
10	10	21111	Oyku Bozan	1	21	Ankara - Cankaya	student10@outlook.com

Figure 26: POST request to update student's information

In Figure 26, I have updated the student_no, age, and address information by sending POST request. As it is a POST request, I have entered some input with the values that I wanted to change and copied the information of other columns as they are. You can see the student table with the updated data.



Query Editor

Query History

```
1 SELECT id, student_no, name, advisor_id, age, address, email
2 FROM test.student
3 ORDER BY id ASC;
```

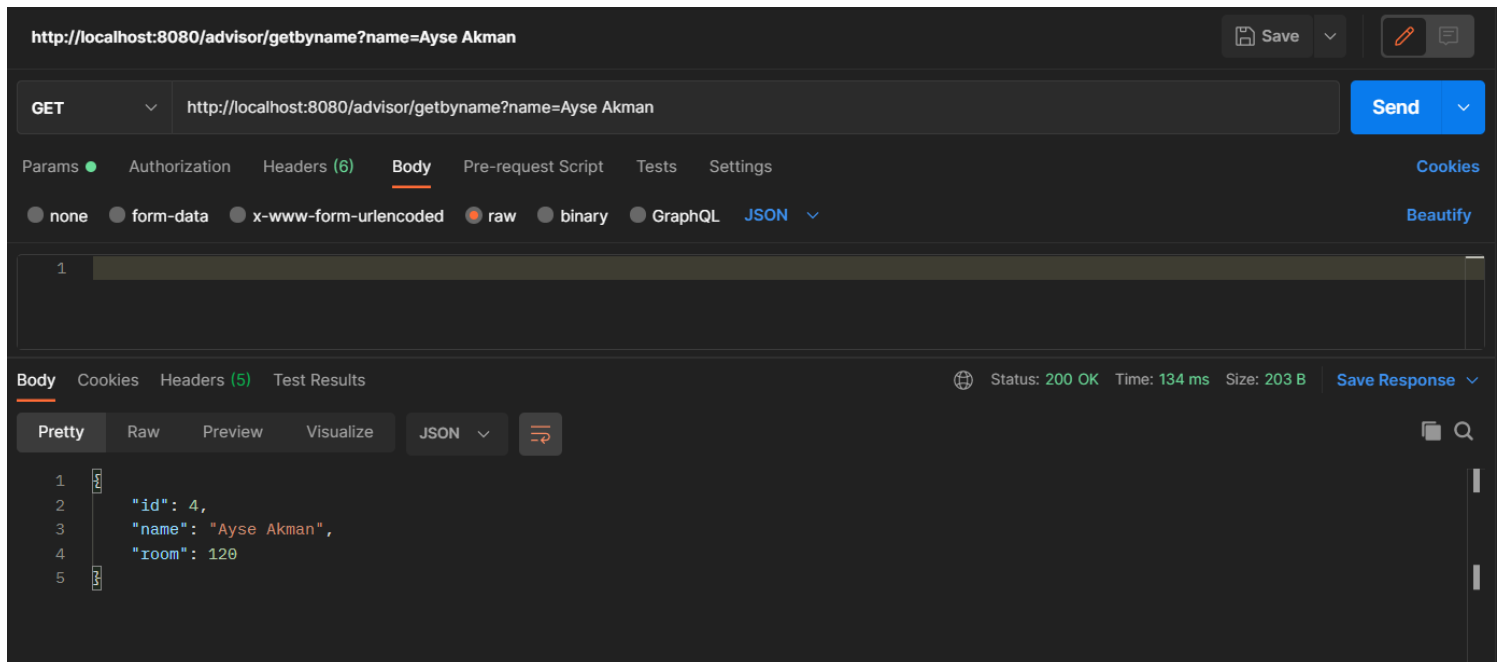
Data Output

Explain

	<div>id</div> <div>[PK] integer</div>	<div>student_no</div> <div>integer</div>	<div>name</div> <div>character varying (25)</div>	<div>advisor_id</div> <div>integer</div>	<div>age</div> <div>integer</div>	<div>address</div> <div>character varying (25)</div>	<div>email</div> <div>character varying (25)</div>	
1		1	23150	Erem Aksoy	1	20	Ankara	student1@yahoo.com
2		2	23589	Ayşe Adiyok	4	24	Kars	student2@gmail.com
3		3	21179	Bugra Altinkaya	3	18	Canakkale	student3@gmail.com
4		4	25720	Ahmet Deniz	1	24	Balikesir	student4@yahoo.com
5		5	28720	Irem Kucuk	2	22	Izmir	student5@yahoo.com
6		6	29613	Berk Uzun	1	23	Izmir	student6@gmail.com
7		7	23841	Can Iz	3	21	Eskisehir	student7@yahoo.com
8		8	20025	Elif Buyuk	4	28	Eskisehir	student8@yahoo.com
9		9	23587	Kerem Tunc	3	20	Bursa	student9@outlook.com

Figure 27: DELETE request to delete a specific student

As seen in Figure 27, I have sent a DELETE request with the id of a particular student that I wanted to delete, and the student is deleted from the table in the database.



Query Editor

Query History

1 SELECT id, advisor_name, room

2 FROM test.advisor;

Data Output

Explain

	id [PK] integer	advisor_name character varying (25)	room integer
1	1	Ahmet Kocak	75
2	2	Gizem Yilmaz	86
3	3	Mehmet Bahar	90
4	4	Ayse Akman	120

Figure 28: GET request for an advisor

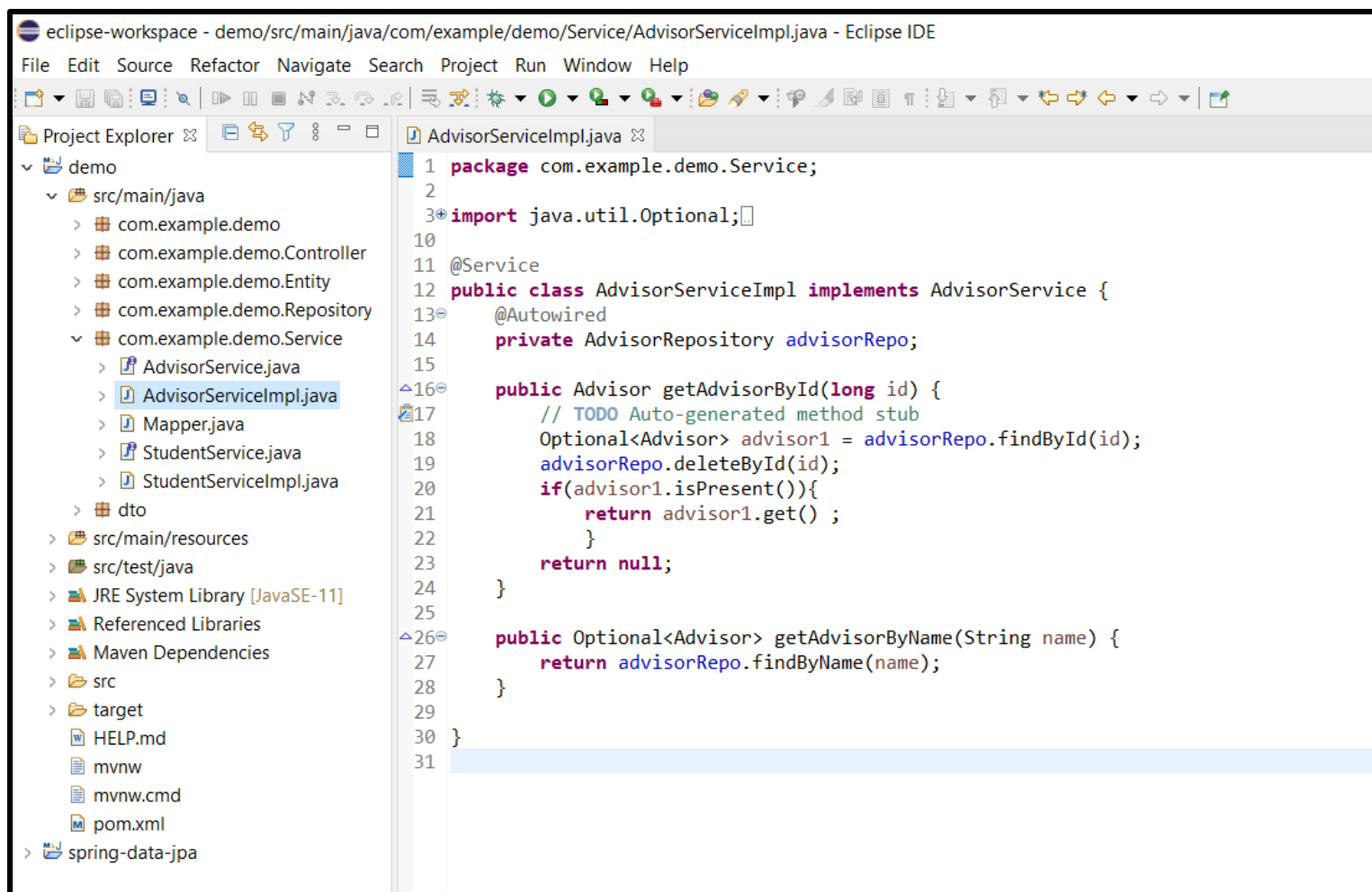
Here in Figure 28, I have sent GET request for an advisor this time. What it does is that it gives the information of an advisor according to the name that I have entered. You can check the output in Postman and the actual information in the advisor table.

9.8 Service

This section will introduce some terms that I have used to make the code easy to read and debug. They are Service Implementation, Mapper, and DTO classes. As the Service interface itself is discussed before in section 9.5.1, it is known that Service helps the code to be cleaner, and it is a connection between Controller and Repository. Moreover, it creates a safe environment as it cuts the physical connection between the user and the program, so it protects the program by not allowing any change.

9.8.1 Service Implementation

The prime purpose of the Service Implementation class is to provide the implementations for the Service interface. An instance of the implementation class needs to be compatible with the Service interface, and it must be a public concrete class in order to reach the object from outside. I have created two different Implementation classes, one for student and one for the advisor entities. You can see the implementation of these classes below:



The screenshot shows the Eclipse IDE with the file `AdvisorServiceImpl.java` open. The Project Explorer on the left shows the project structure, with `AdvisorServiceImpl.java` selected under `com.example.demo.Service`. The main editor displays the following code:

```
1 package com.example.demo.Service;
2
3 import java.util.Optional;
4
5
6
7
8
9
10
11 @Service
12 public class AdvisorServiceImpl implements AdvisorService {
13     @Autowired
14     private AdvisorRepository advisorRepo;
15
16     public Advisor getAdvisorById(long id) {
17         // TODO Auto-generated method stub
18         Optional<Advisor> advisor1 = advisorRepo.findById(id);
19         advisorRepo.deleteById(id);
20         if(advisor1.isPresent()){
21             return advisor1.get() ;
22         }
23         return null;
24     }
25
26     public Optional<Advisor> getAdvisorByName(String name) {
27         return advisorRepo.findByName(name);
28     }
29
30 }
31
```

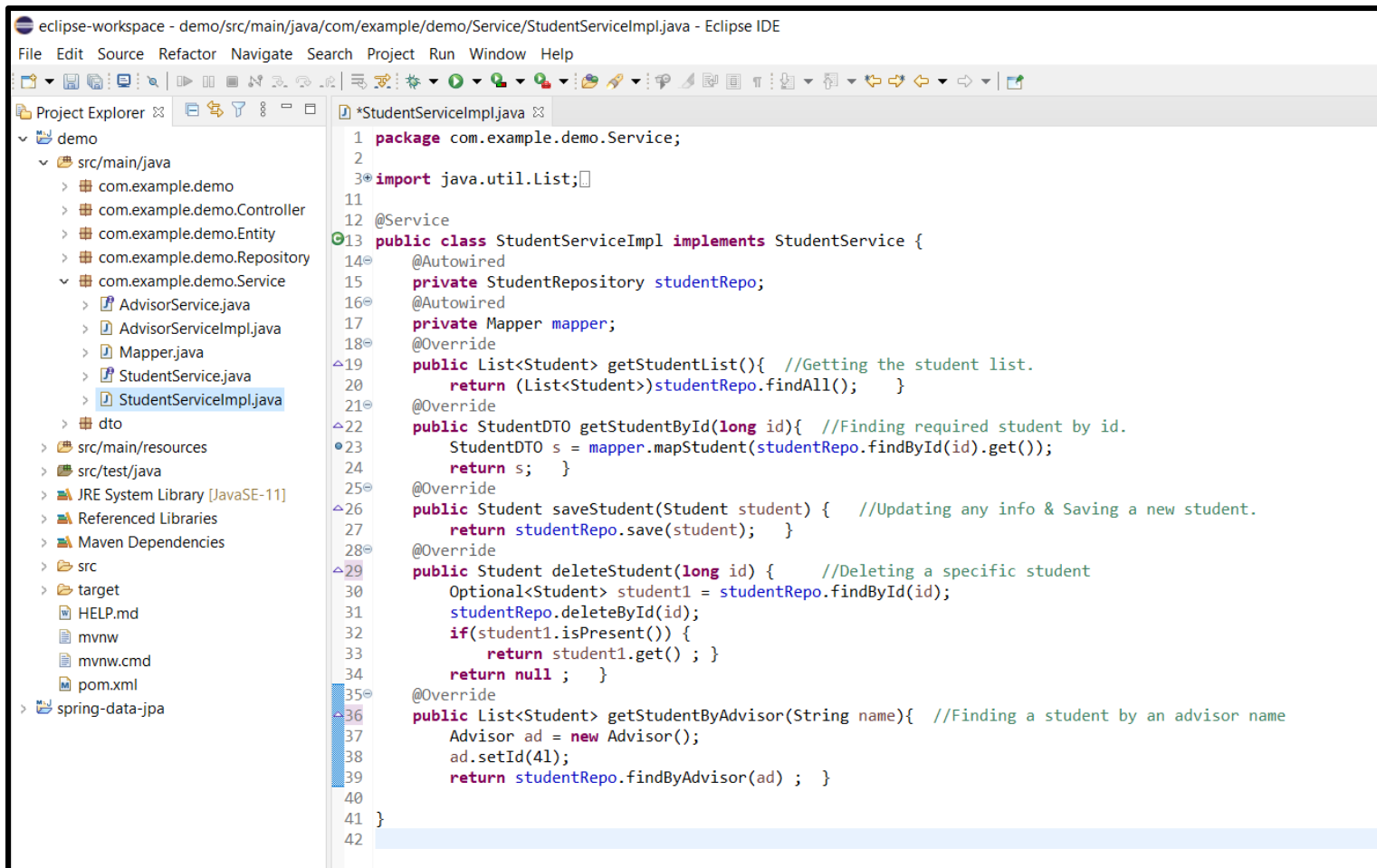



Figure 29: AdvisorServiceImplementation and StudentServiceImplementation classes

9.8.2 Mapper

When the user sends a request from Postman, a method for the corresponding request on the mapper is called to load the required data from the database. If the data is not loaded already, it loads it. A mapper creates communication between objects and a database while holding them independent of each other.

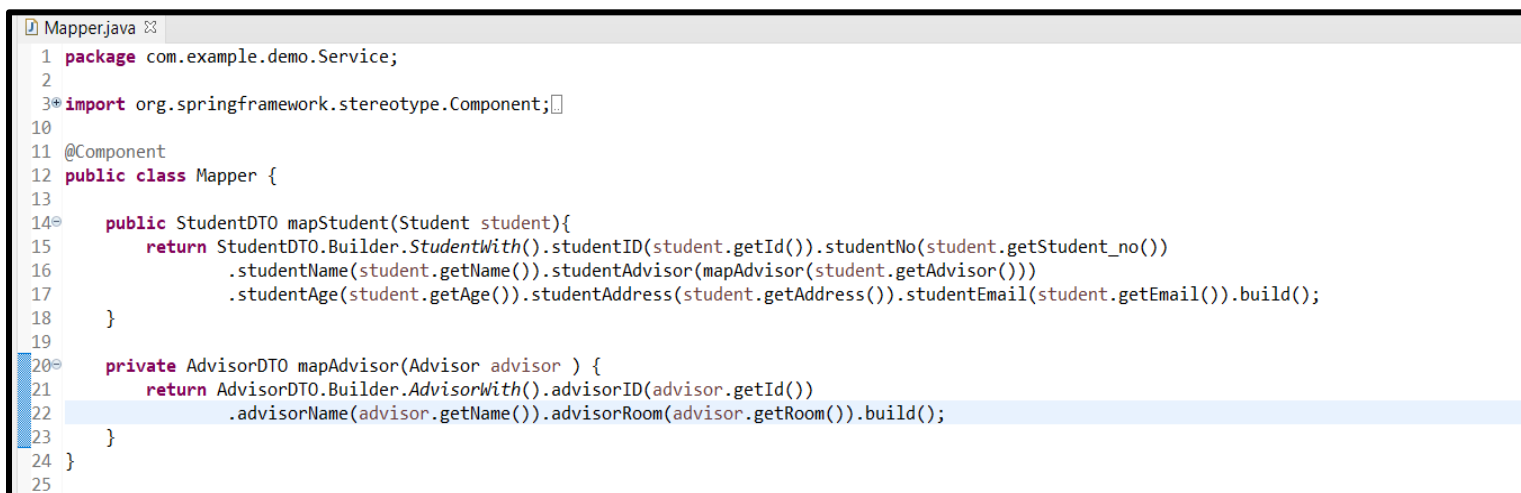


Figure 30: The Mapper Class

In Figure 30, there is the Mapper class that I have created, and I have used it to convert an entity object to a DTO for both student and advisor. I have also created a Builder class (see 9.8.3), and the Mapper class takes methods from that class. I have called the methods I want, and at the end of these method calls, I have also called the build() method to create a new object. All the methods that are called in the Mapper class are inside the Builder class.

9.8.3 Design Patterns: The Build Pattern

As I have mentioned before, it is all about classes in object-oriented languages. We create objects from classes, and we need constructors to do that. The more we have fields, the more constructors we need. If not all the fields are mandatory, we need another constructor which takes the mandatory fields as a parameter. The problem is that when we do not know which fields are mandatory, we need to create constructors for all the possible fields. It would cause a big problem when an object has many fields. By using a build pattern, this problem with a large number of optional parameters is solved. It provides a way to build the object step-by-step and a method that returns the final object.

Here, I want to highlight that the Student class has only getter methods and no public constructor, so the only way to get a student object is through the builder.

DTO (Data Transfer Object): It is a design pattern used to transfer data between software systems. It is used to conjunct the data access objects to retrieve data from a database. It differs from business objects as DTO does not have any functionality except for storing and retrieving of its data. Using DTO, the number of calls can be reduced as DTO aggregates the data that many calls could transfer, but that is served by only one call. So, it can be said that the DTO simplifies the code with fewer classes, and they can also hold data from single or multiple sources. [15]

I have created two builder classes, one for student and one for an advisor object. You can see their implementation in Figures 31 and 32 below:



Figure 31: StudentDTO and Builder Classes for a Student Entity

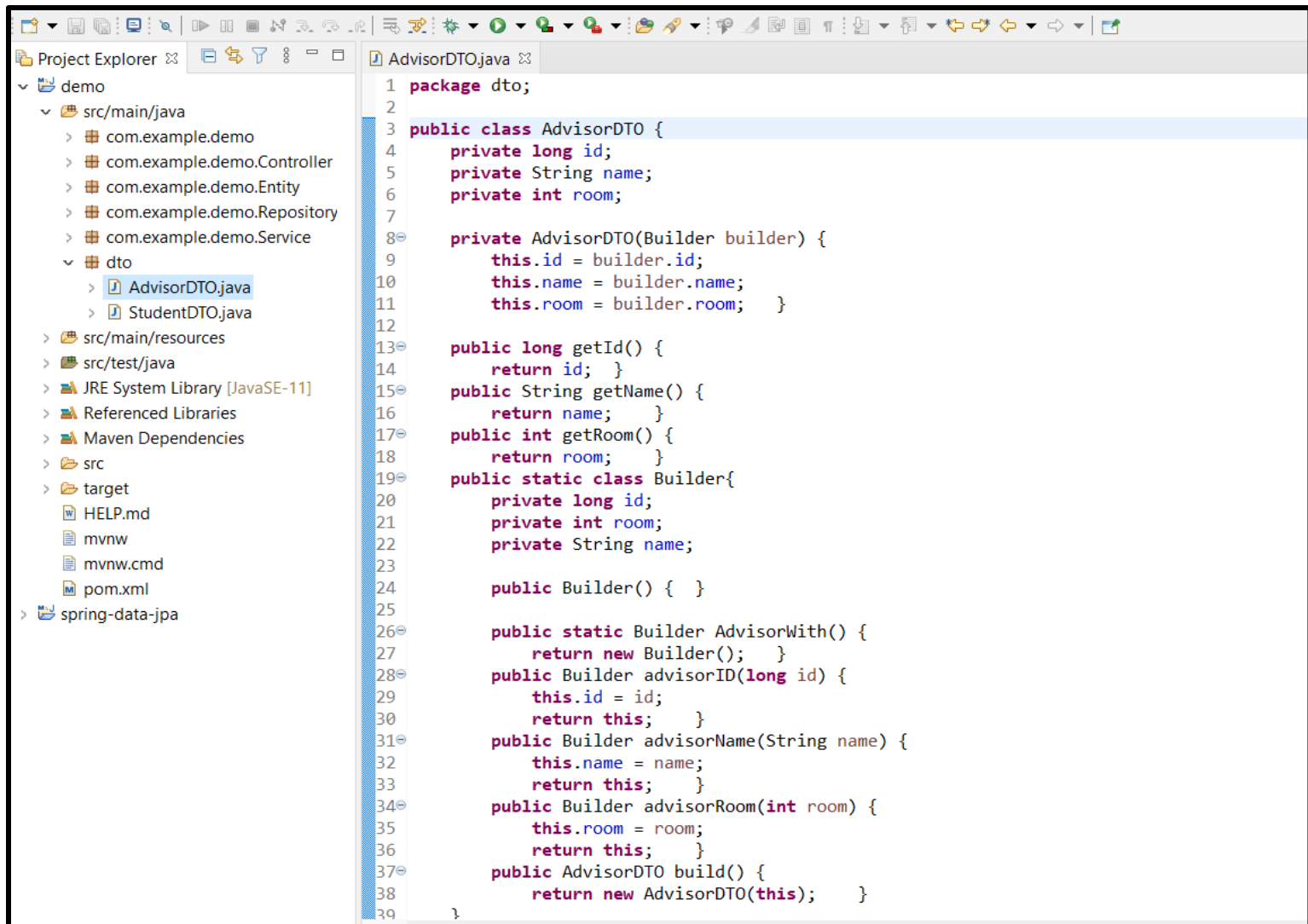


Figure 32: AdvisorDTO and Builder Classes for an Advisor Entity

In Figures 31 and 32, I have used DTO and Builder classes. The logic behind them is that I have assigned the fields that I want to a new object in DTO classes and created a new object in the Builder class with the build() method with all the fields that I have wanted to add. Here, it is crucial for both classes to be public to get the objects student and advisor, which are private. So, it basically takes the fields that we required and creates an object accordingly, then assigns all the values that we have entered to the appropriate fields. It is very beneficial to avoid creating many constructors for all possible cases because it can be done by only one constructor that saves time and memory and makes the code cleaner and easier to read/debug.

10. CONCLUSION

During 21 days of my internship, I have worked on testing and analyzing practices. I have learned a new language, Java, and using some software tools/frameworks, such as Spring Boot and Postman. I have also used the PostgreSQL database management system to enhance my existing knowledge of SQL. I have combined all these and learned how to get a request from the user at the frontend, process data, make necessary changes in that data in a database, and test the software by using tools. I have learned how to make the code cleaner and easy to read and debug and apply this knowledge to my code.

After the summer practice, I have gained good experience in a real work environment and improved my skills, such as problem-solving, critical thinking, and coming up with a solution quickly. I have also improved my communication skills, met many people, and made new connections. Besides, I can say that the most significant gain from this practice for me has been developing my software-related skills like coding, testing, and analyzing.

I am interested in database management for a few years now, and I believe that this practice helped me a lot to decide if I want to pursue my career or focus on this area more. As I have worked on analyzing as well and enjoyed it very much, I realize that I would like to concentrate on the data science area, and during my education in METU NCC, I will make sure that I make use of all the sources that my school provides.

The knowledge that I have before my internship, thanks to my instructors and the courses I have taken so far, helped me to manage many problems that I have faced. I have managed to learn a new programming language as I have already known one, C, and knowing about data structures also assisted me at some point in my struggles, understanding DTO, which is a data structure.

As my final words, I want to thank my supervisor Alper Çepni for his support and kindness during my practice and for how he guided me. I also want to thank Ümit Ateş for accepting me and giving me the opportunity to be an intern in their company.

REFERENCES

1. The official website of VBT Yazılım A.Ş, <https://www.vbt.com.tr/>
2. Wikipedia for Object-Oriented Language, https://en.wikipedia.org/wiki/Object-oriented_programming
3. The official website of Java, <https://www.java.com/tr/>
4. The official website of Eclipse, <https://www.eclipse.org/ide/>
5. The official website of PostgreSQL, <https://www.postgresql.org/>
6. The official website of pgAdmin, <https://www.pgadmin.org/>
7. Wikipedia for REST, https://en.wikipedia.org/wiki/Representational_state_transfer
8. Wikipedia for JSON, <https://en.wikipedia.org/wiki/JSON>
9. The official website of Postman, <https://www.postman.com/>
10. The official website of Spring, <https://spring.io/>
11. Wikipedia for Hibernate ORM, [https://en.wikipedia.org/wiki/Hibernate_\(framework\)](https://en.wikipedia.org/wiki/Hibernate_(framework))
12. The official website of Hibernate, <https://hibernate.org/orm/>
13. Javatpoint website for Java contents, <https://www.javatpoint.com/>
14. Website for Spring Framework annotations, <https://springframework.guru/spring-framework-annotations/>
15. Wikipedia for DTO, https://en.wikipedia.org/wiki/Data_transfer_object