Software Engineering for the WWW

Spring Boot

Dr. Vinod Dubey
SWE 642
George Mason University

Spring Boot

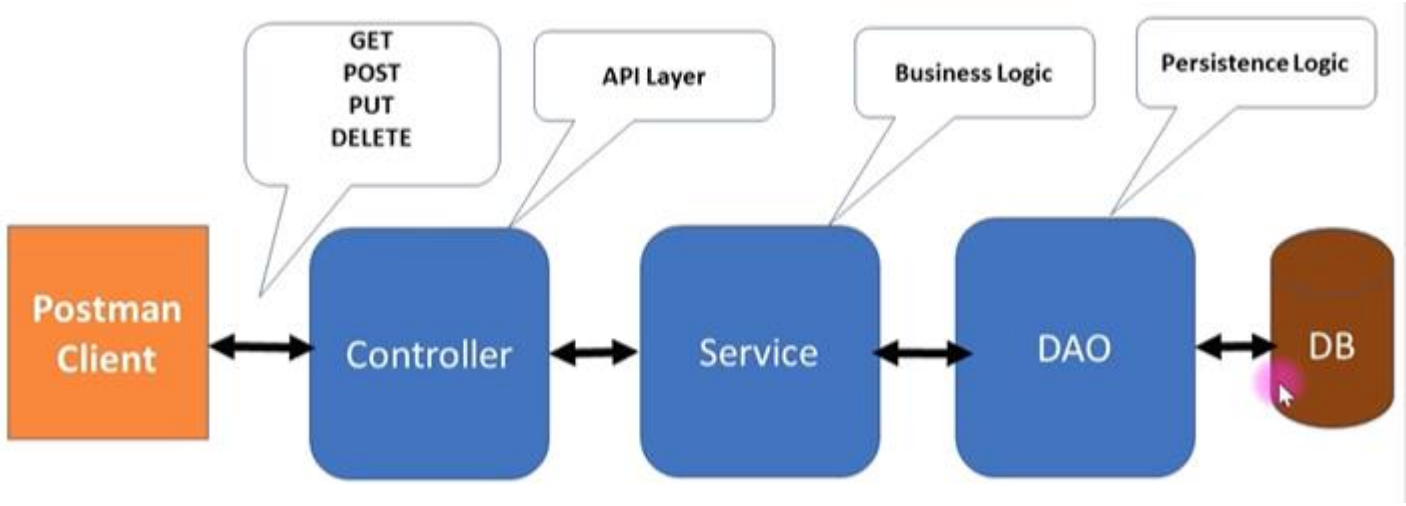
- **Agenda**
  - Installing JDK
  - Spring Boot
  - Creating RESTful and JPA application using Spring Boot
  - Installing MySQL server and Workbench

Spring Boot - Goal

- **How to implement REST APIs using**
  - Spring Boot,
  - Spring Data JPA (Hibernate), and
  - MySQL

Spring Boot Tutorial

- **Typical three-layer architecture to implement Spring Boot based applications**
  - Controller layer hosts REST APIs
  - Service layer encapsulates business logic, and
  - DAO that encapsulates data persistence logic.
  - Postman - REST client



Source: youtube

Spring Boot Tutorial

- **Student Management System that allows REST client to**
  - To get list of students
  - Get a single student by id
  - Create a new student
  - Update an existing student
  - Delete a student

Spring Boot Tutorial

- **REST APIs for Student resource**

HTTP Method	URL Path	Status Code	Description
GET	/api/students	200 (OK)	Get all students
GET	/api/students/{id}	200 (OK)	Get single student by Id
POST	/api/student	201 (Created)	Create a new student
PUT	/api/students/{id}	200 (OK)	Update an existing student with id
DELETE	/api/students/{id}	200 (OK)	Delete a student with Id

Spring Boot Tutorial

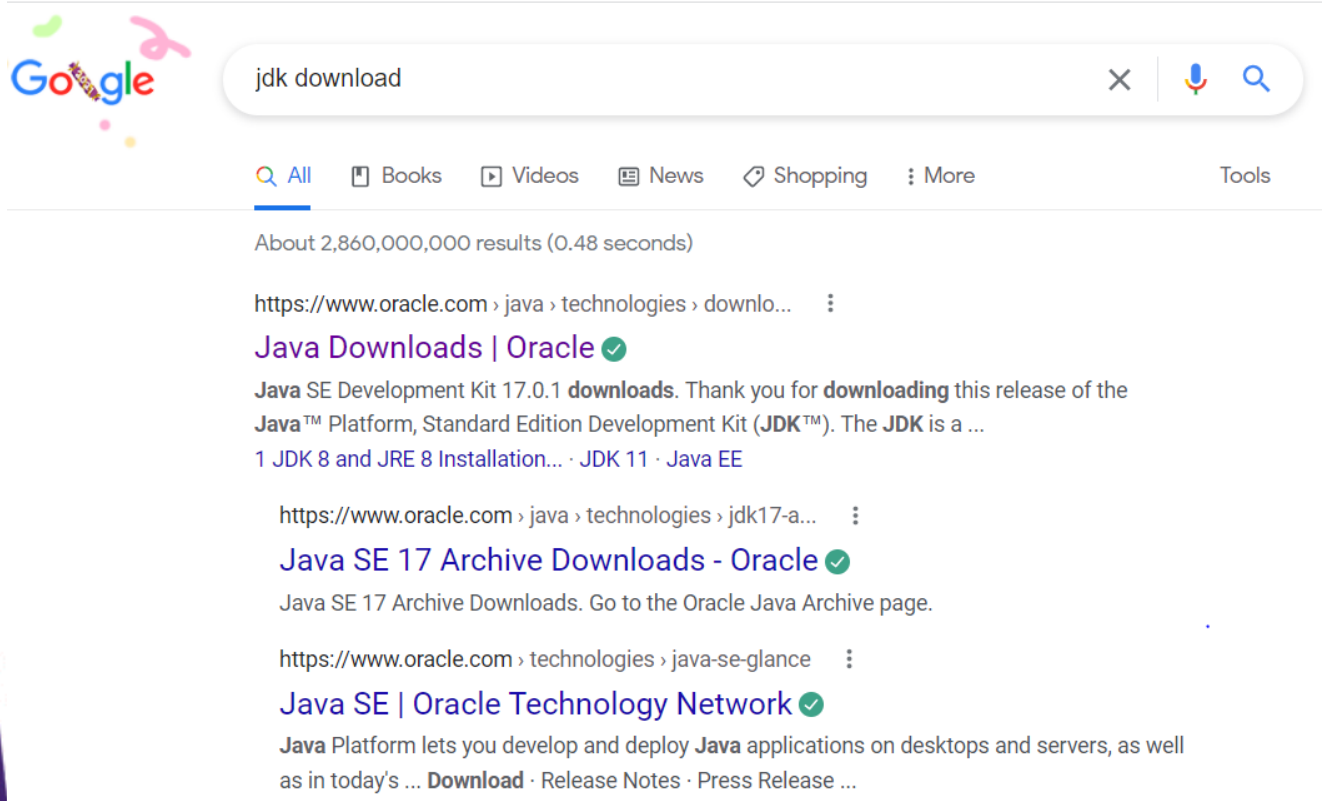
- **Tools and technologies used**
  - JDK 17
  - Spring Boot 2.5.0
  - Spring Data JPA, which internally uses Hibernate as the default JPA provider
  - Embedded Tomcat server 8.5+
  - MySQL database
  - Maven 3.2+
  - Eclipse STS

Spring Boot Tutorial

- **Creating STS (Spring Tool Suite) project**
  - STS IDE requires latest version of Java, e.g., JDK17 preinstalled
  - Download latest version of JDK and install on the machine where you will run the STS IDE
  - Download STS IDE as a jar file, extract, and launch STS IDE on local machine
  - Create a Spring Boot application

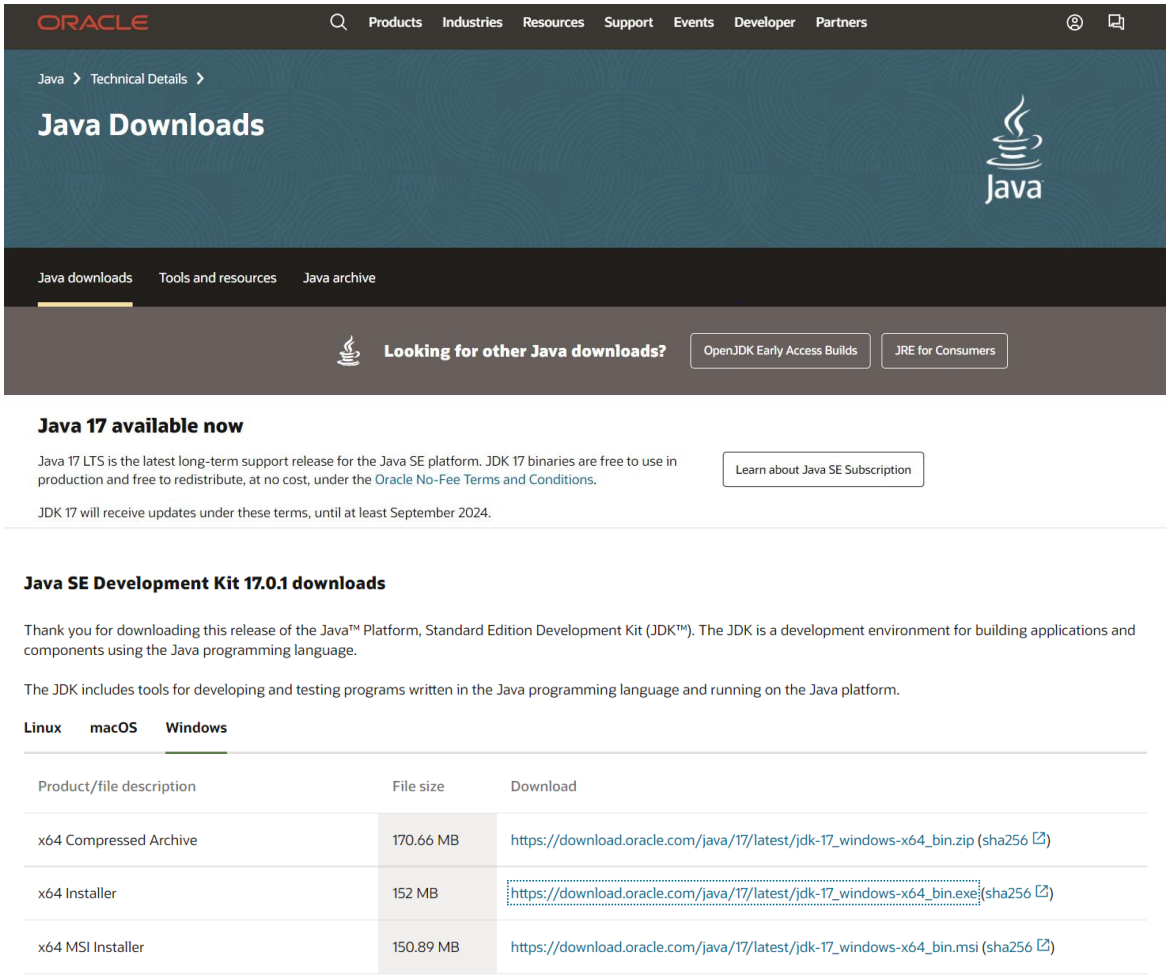
Spring Tool Suite requires Java JDK

- **Download and install JDK**



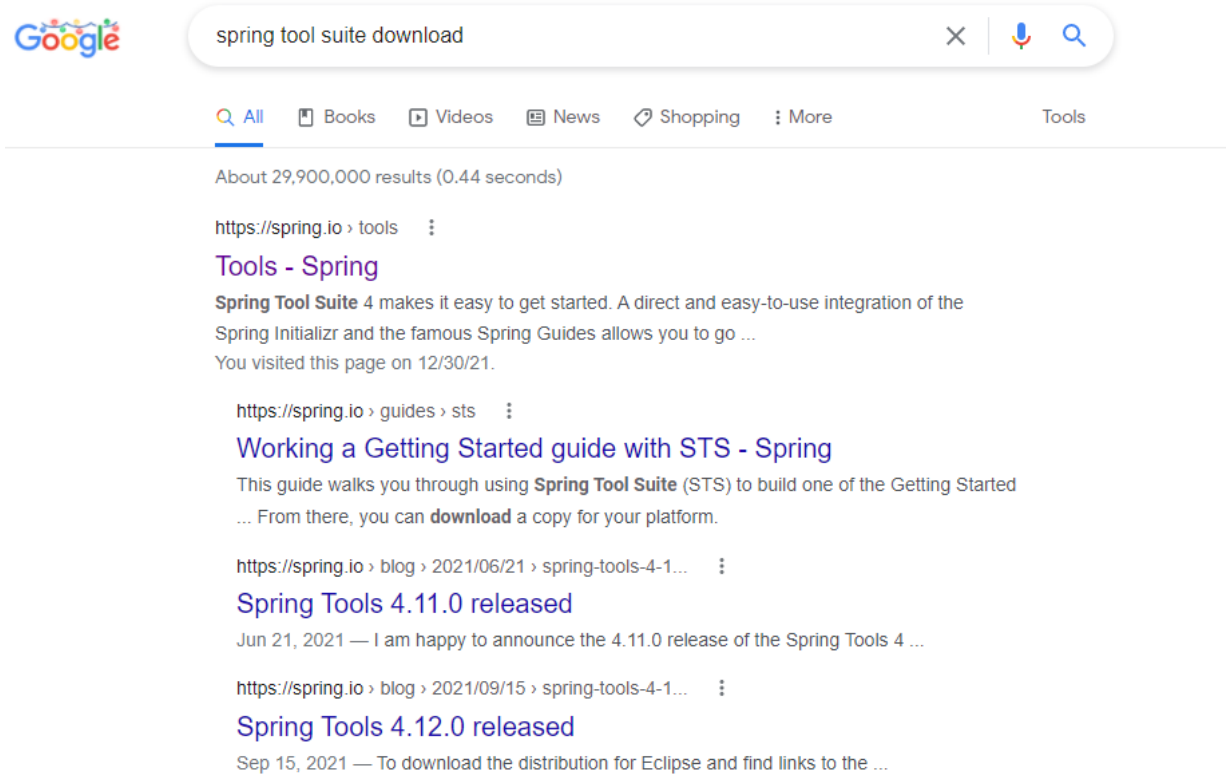
Spring Tool Suite requires Java JDK

- **Download and install JDK**



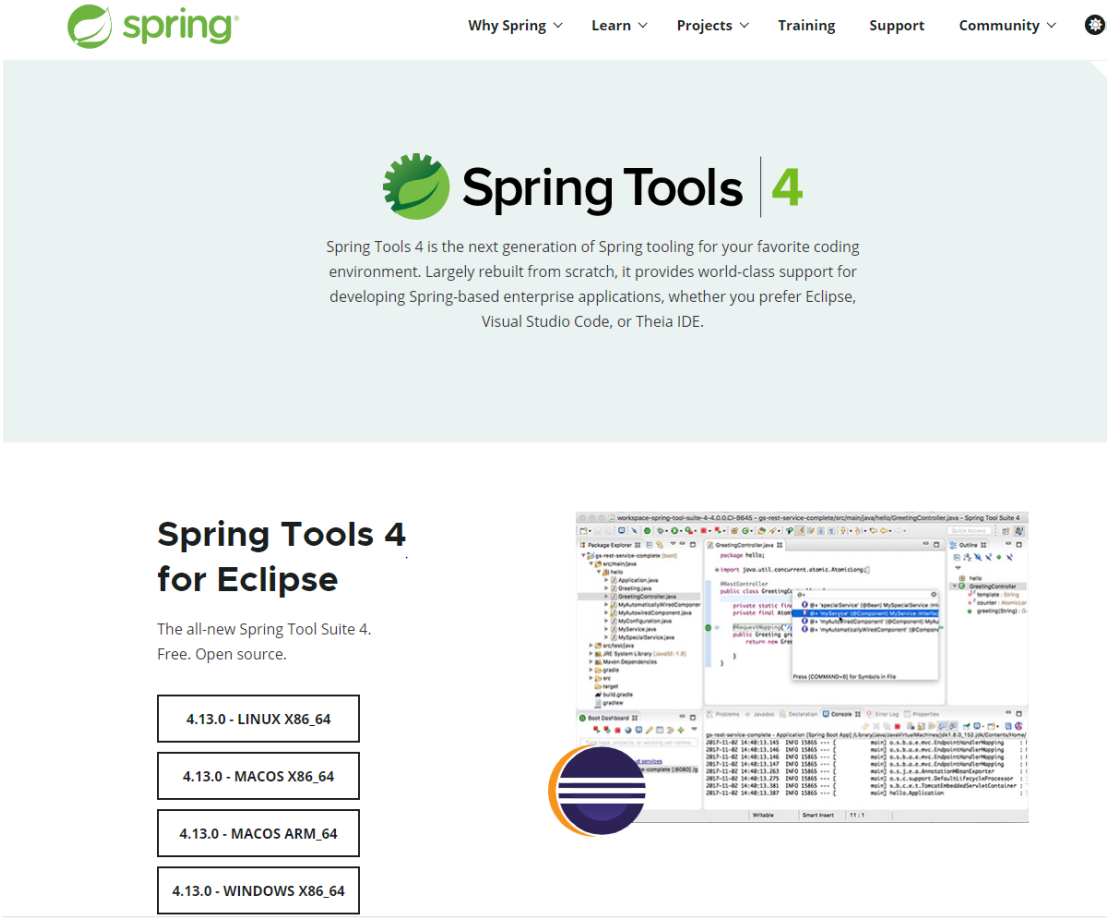
Download Spring Tool Suite

- **Click on the first link ‘Tools – Spring’**



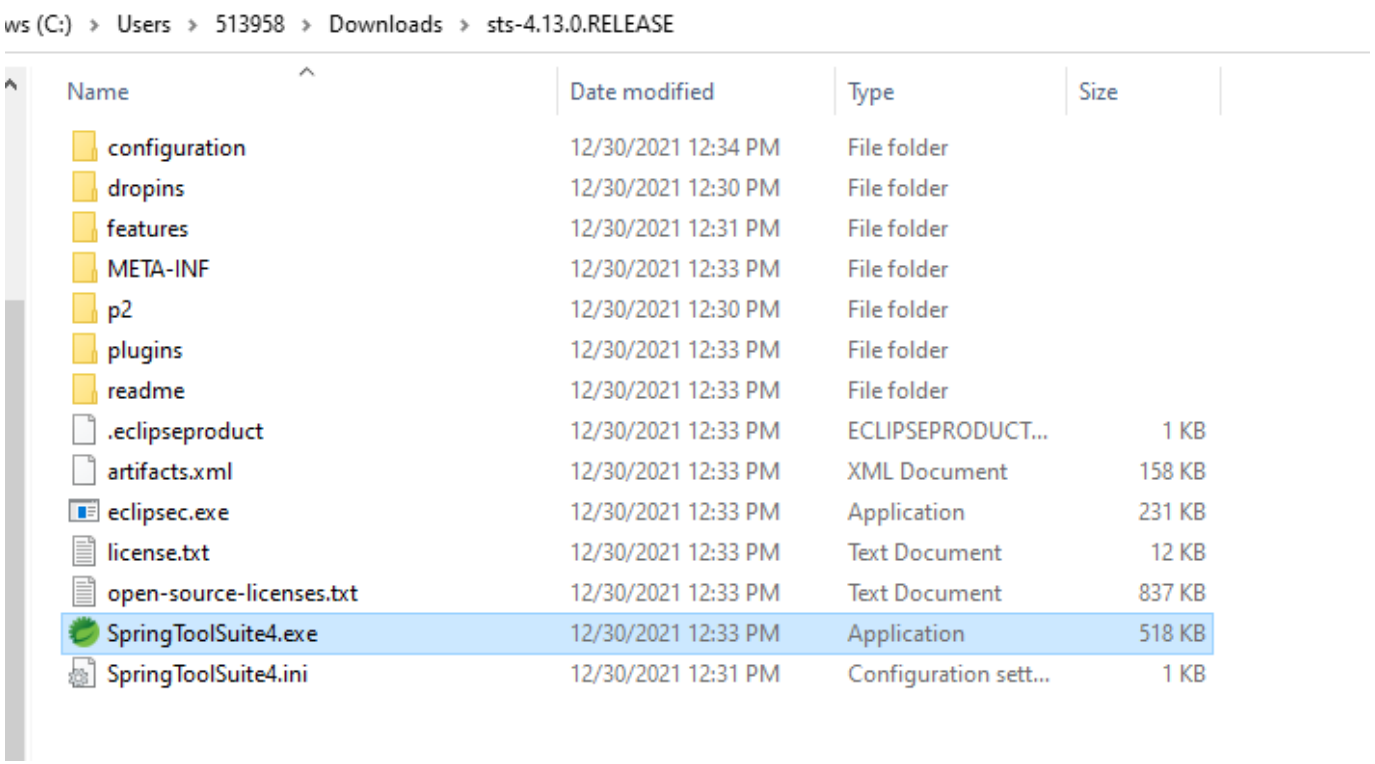
Download Spring Tools 4 for Eclipse for your O/S

- **Download the Spring Tools 4 for Eclipse for your OS**

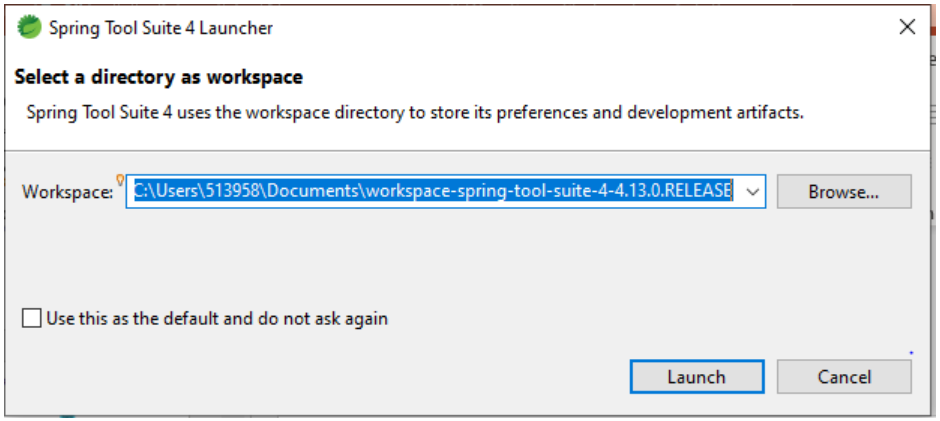


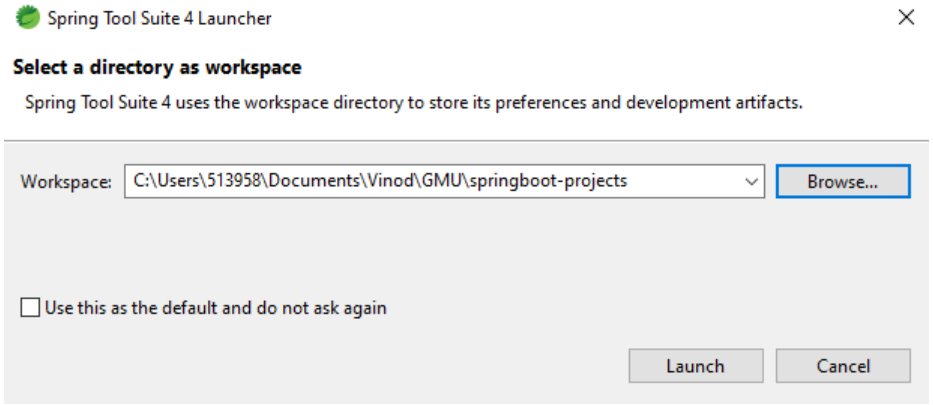
Extract the downloaded Jar

- **Double click on SpringToolSuite4.exe to open the IDE**

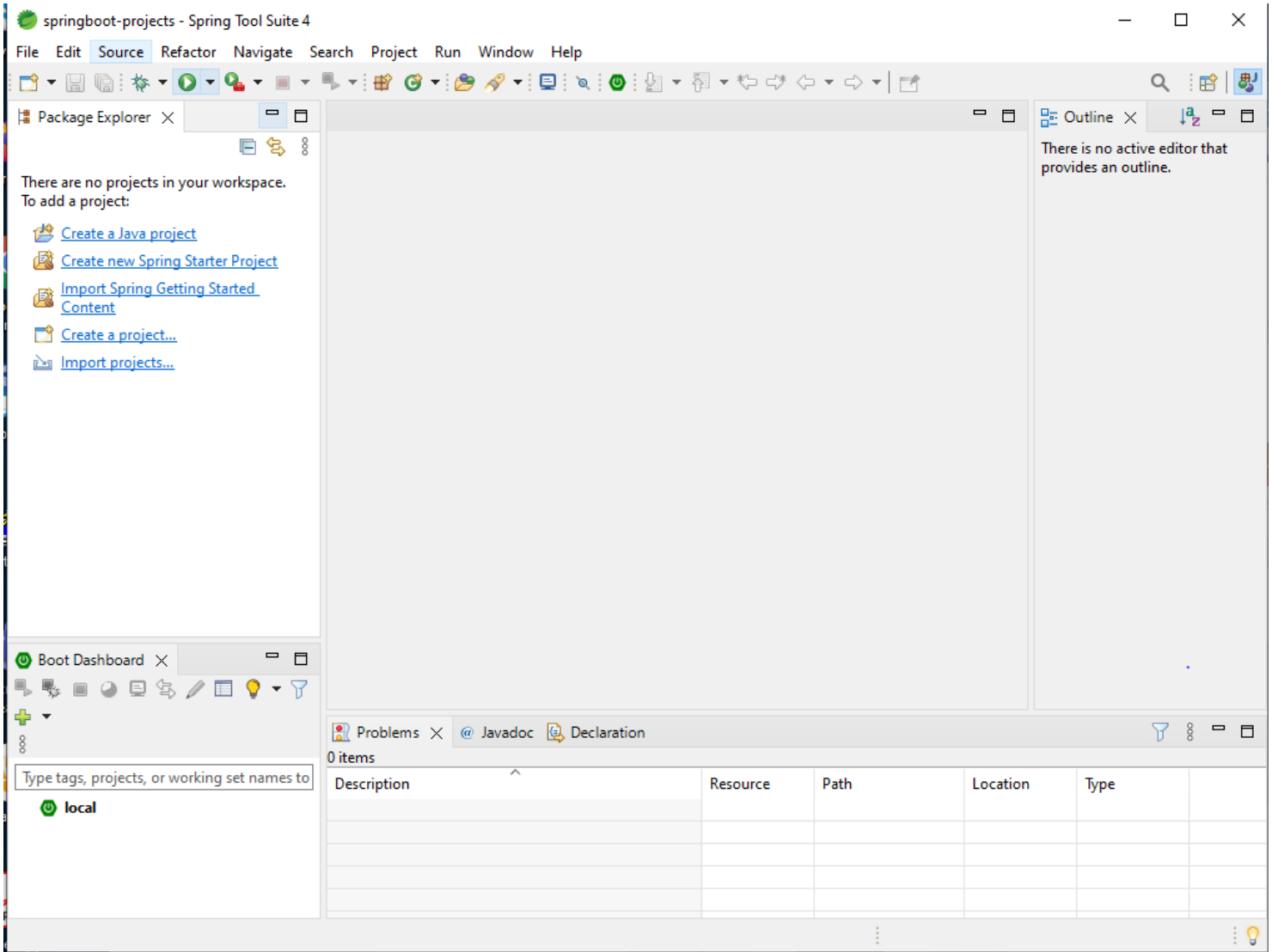


Select a directory as workspace and Launch STS IDE



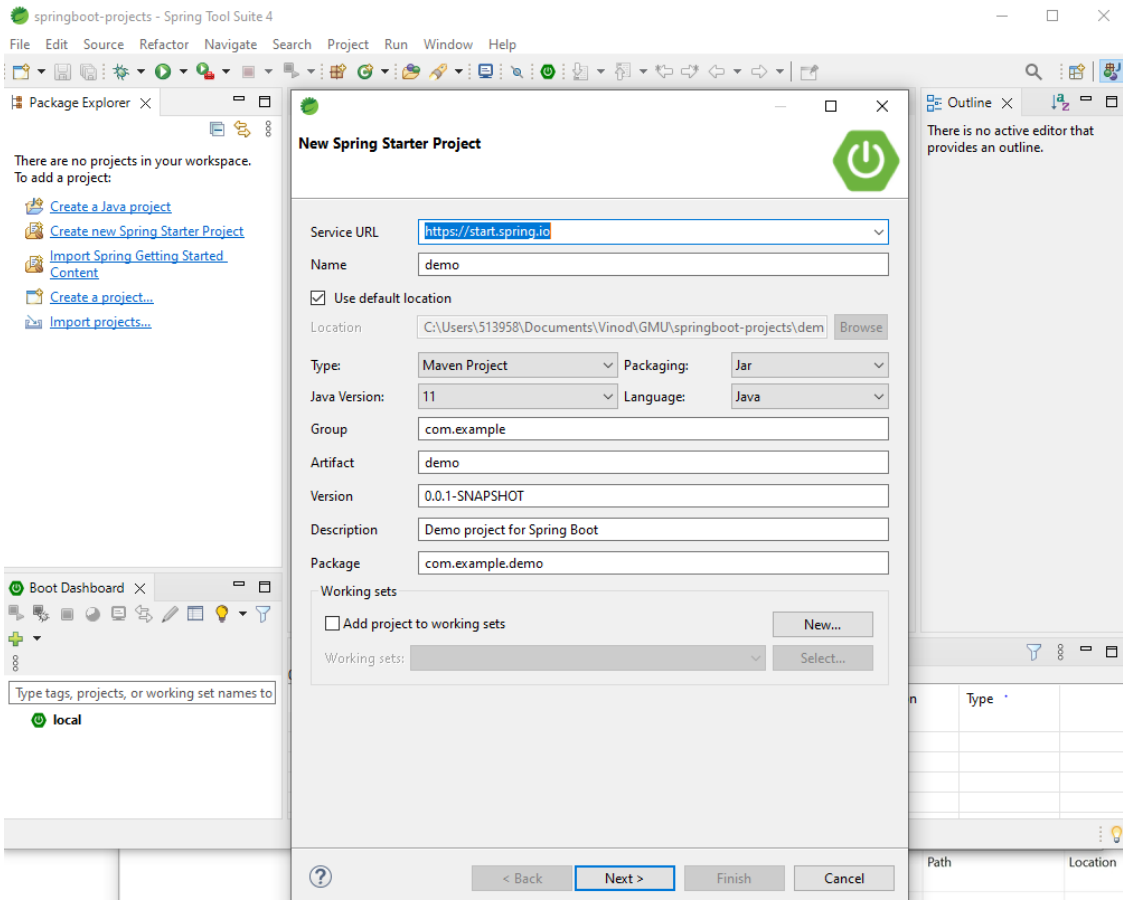


Click on “Create new Spring Starter Project”



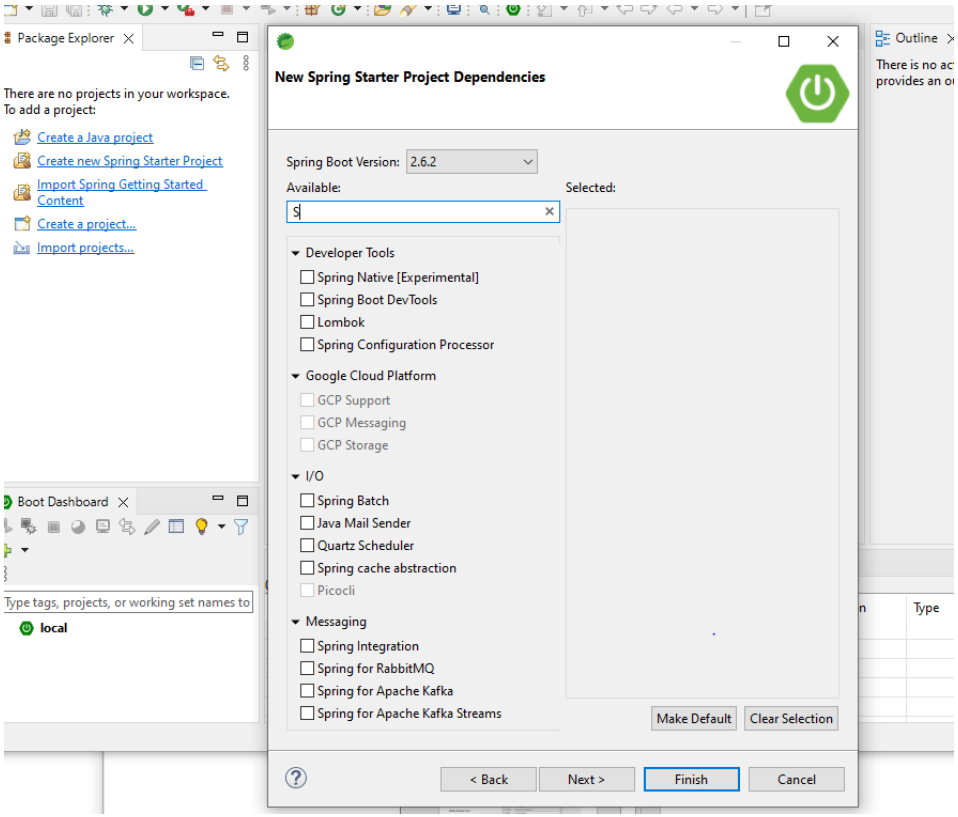
Click on “Create new Spring Starter Project”

- **Service URL refers to Spring Initializer – integrated in the STS IDE**
- **Click on Next and select dependencies**



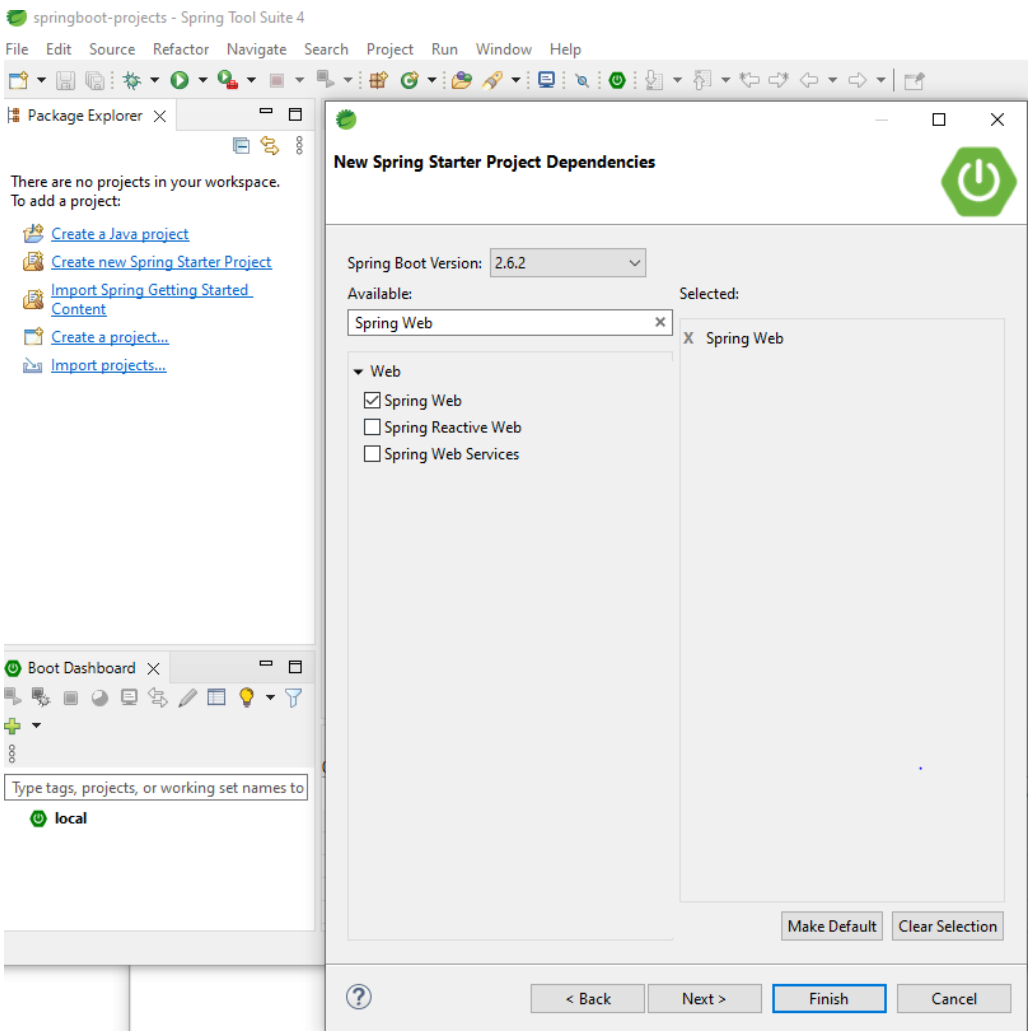
Select Version and project dependencies

- **Select latest Spring Boot Version and project dependencies, such as**
  - Spring Web, Spring Data JPA, MySQL Driver



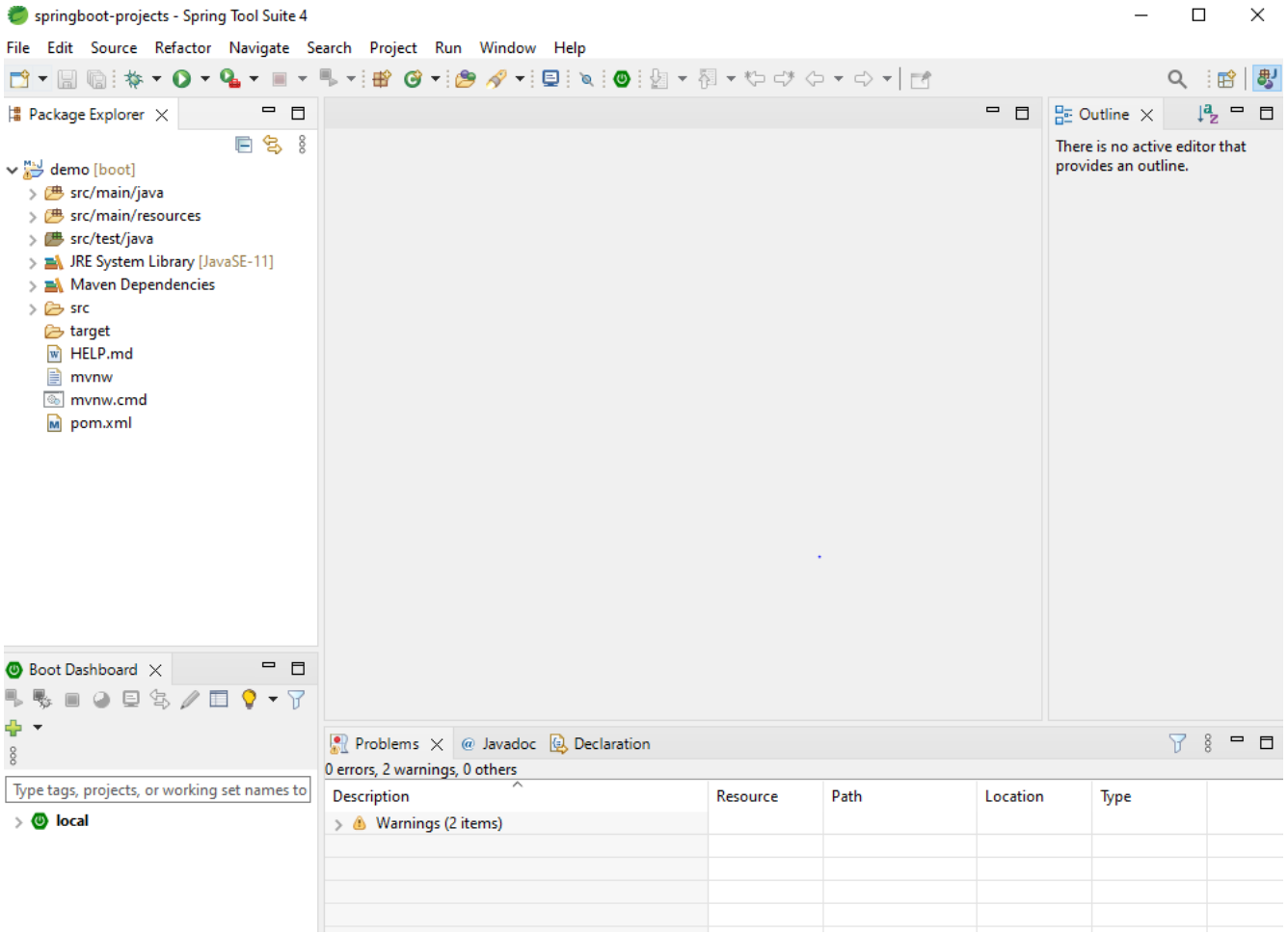
Select Version and project dependencies

- Search and choose ‘Spring Web’ for starter dependencies
- And click Finish



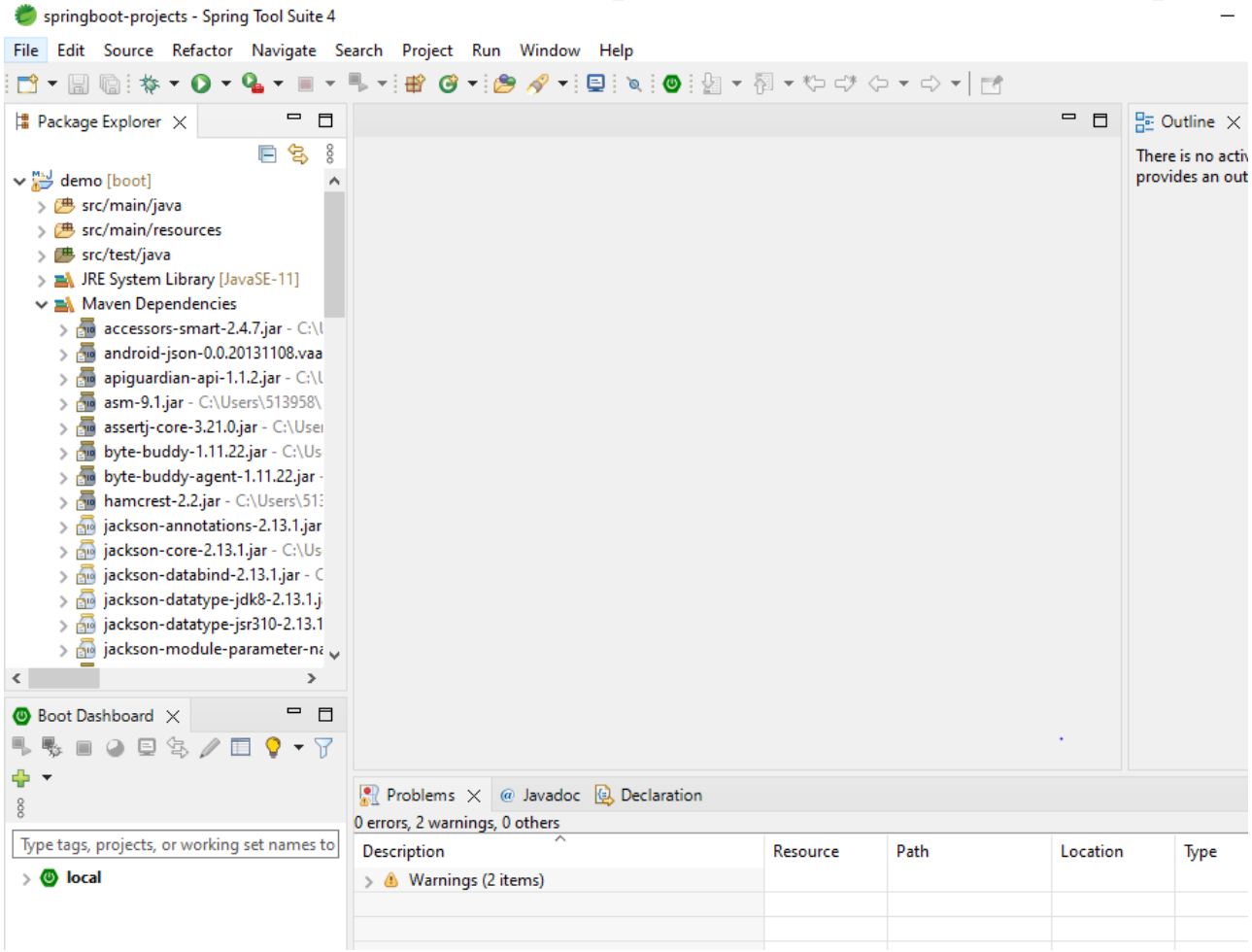
Select Version and project dependencies

- **Spring downloads all dependencies and stores in Maven Dependencies repository**



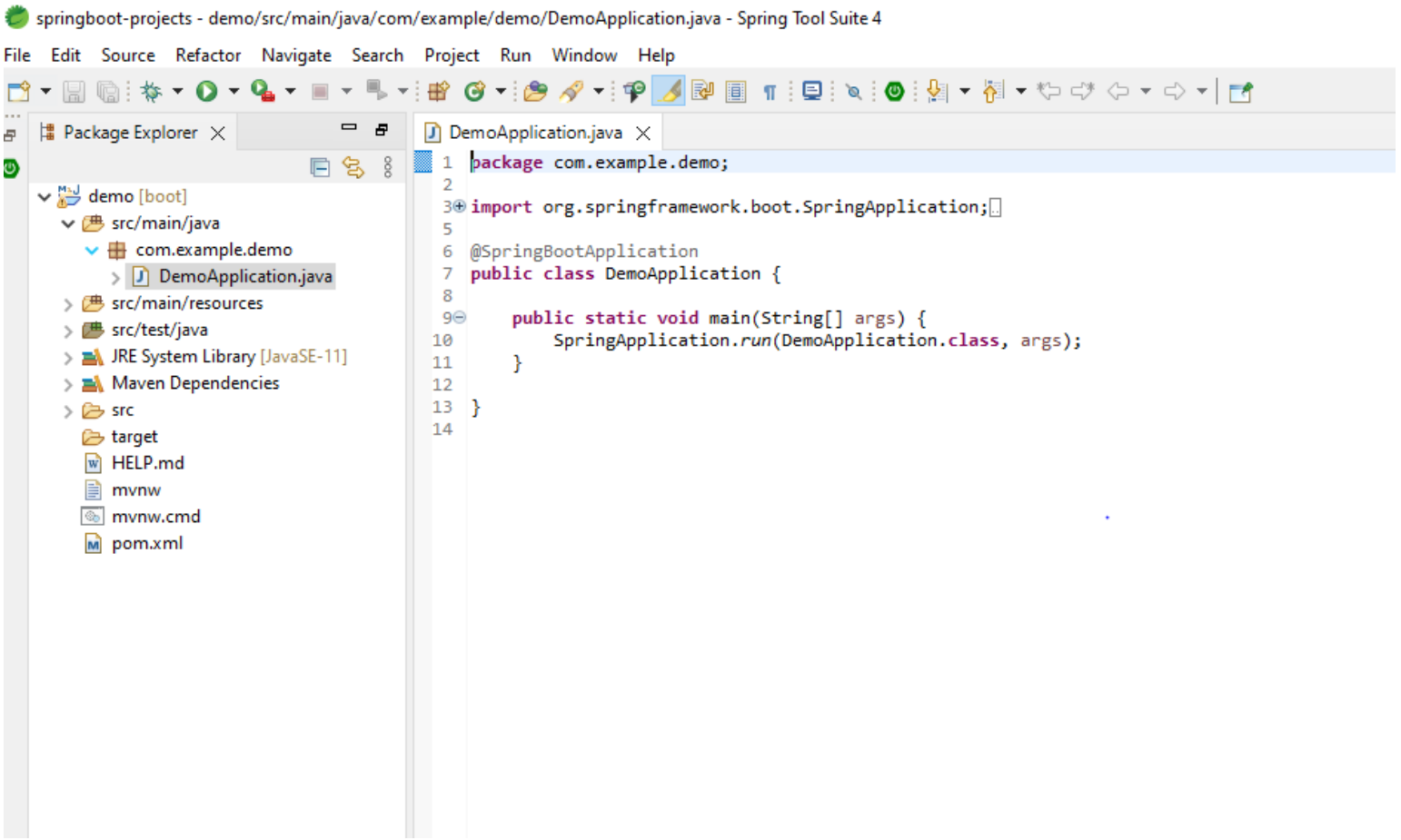
Select Version and project dependencies

- **Spring downloads all dependencies and stores in Maven Dependencies repository**



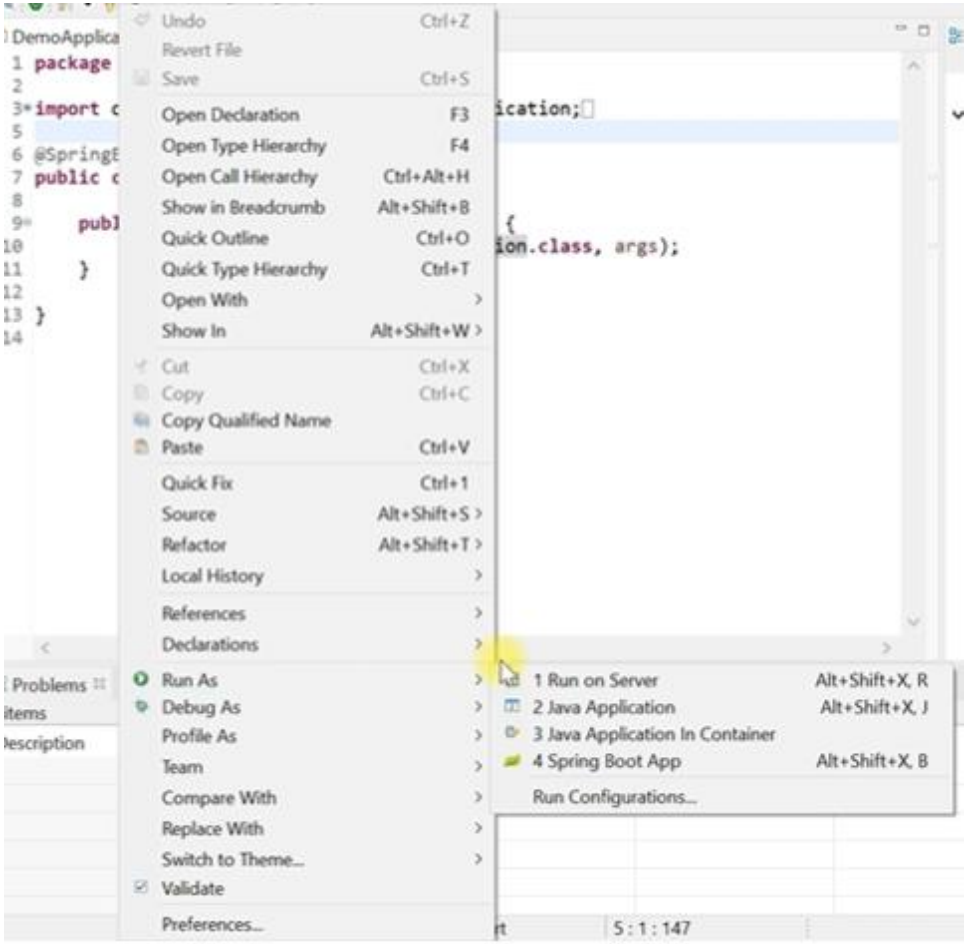
DemoApplication.java

- **Open DemoApplication.java** under /src/main/java/com.example.demo
- **This is Spring Boot main class**



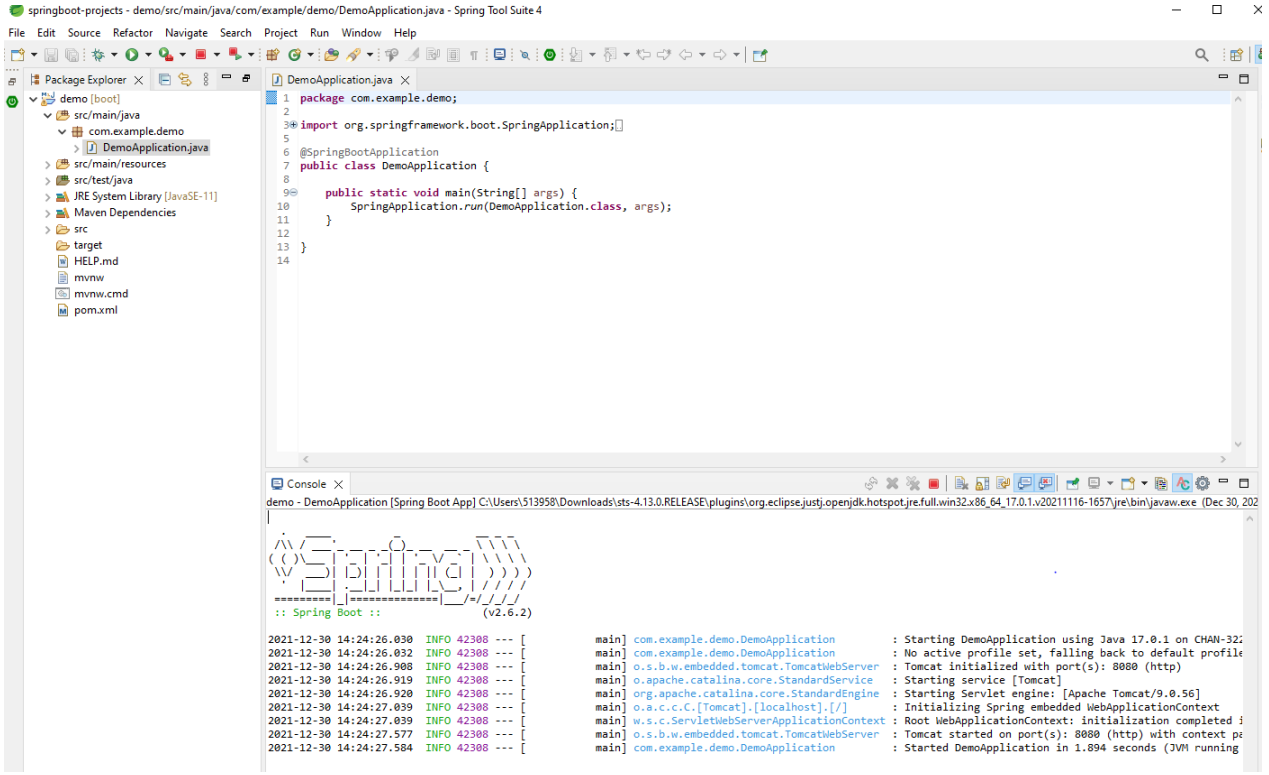
Right click in the righthand pane and select Run As Spring Boot Application
Review the console for output including a line saying 'Tomcat started'
This simple Spring Boot app is running on embedded Tomcat at port 8080.

- **Right click in the righthand pane and select Run As Spring Boot Application**



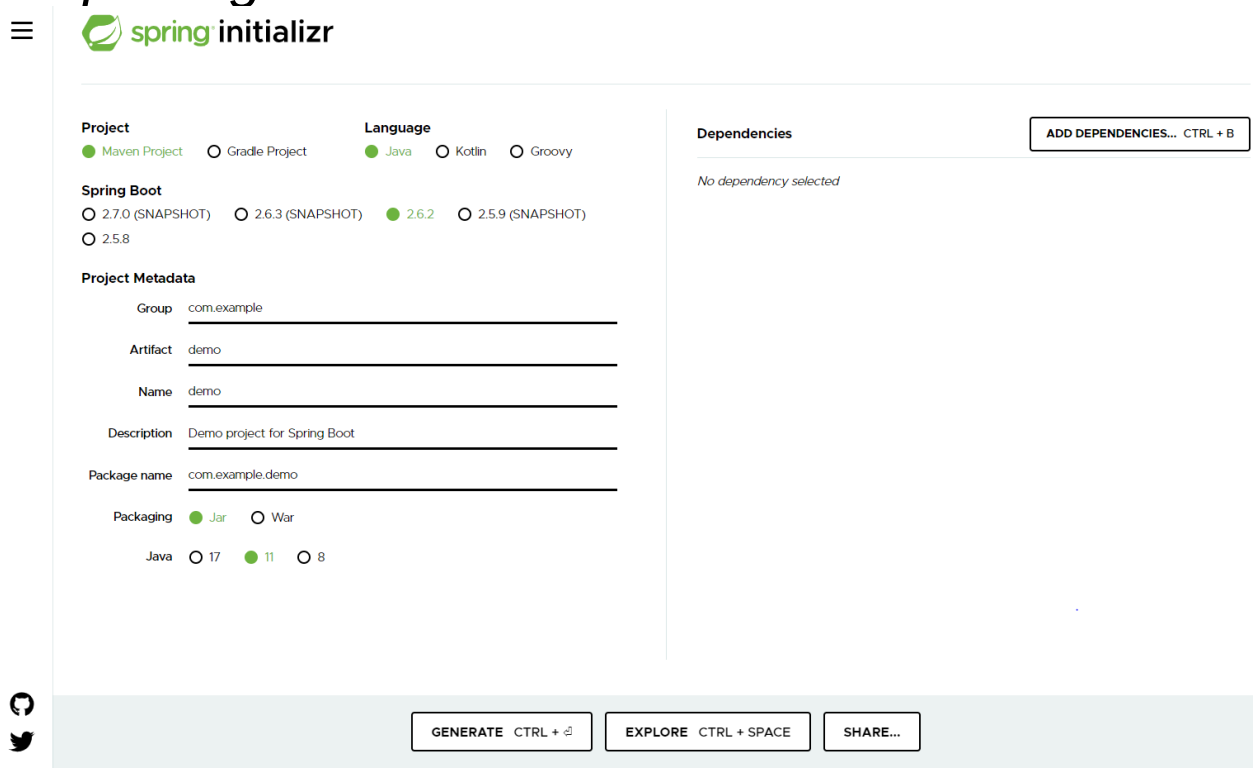
Right click in the righthand pane and select Run As Spring Boot Application Review the console for output including a line saying ‘Tomcat started’ This simple Spring Boot app is running on embedded Tomcat at port 8080.

- Review the console for output including a line saying ‘Tomcat started’
- This simple Spring Boot app is running on embedded Tomcat at port 8080.

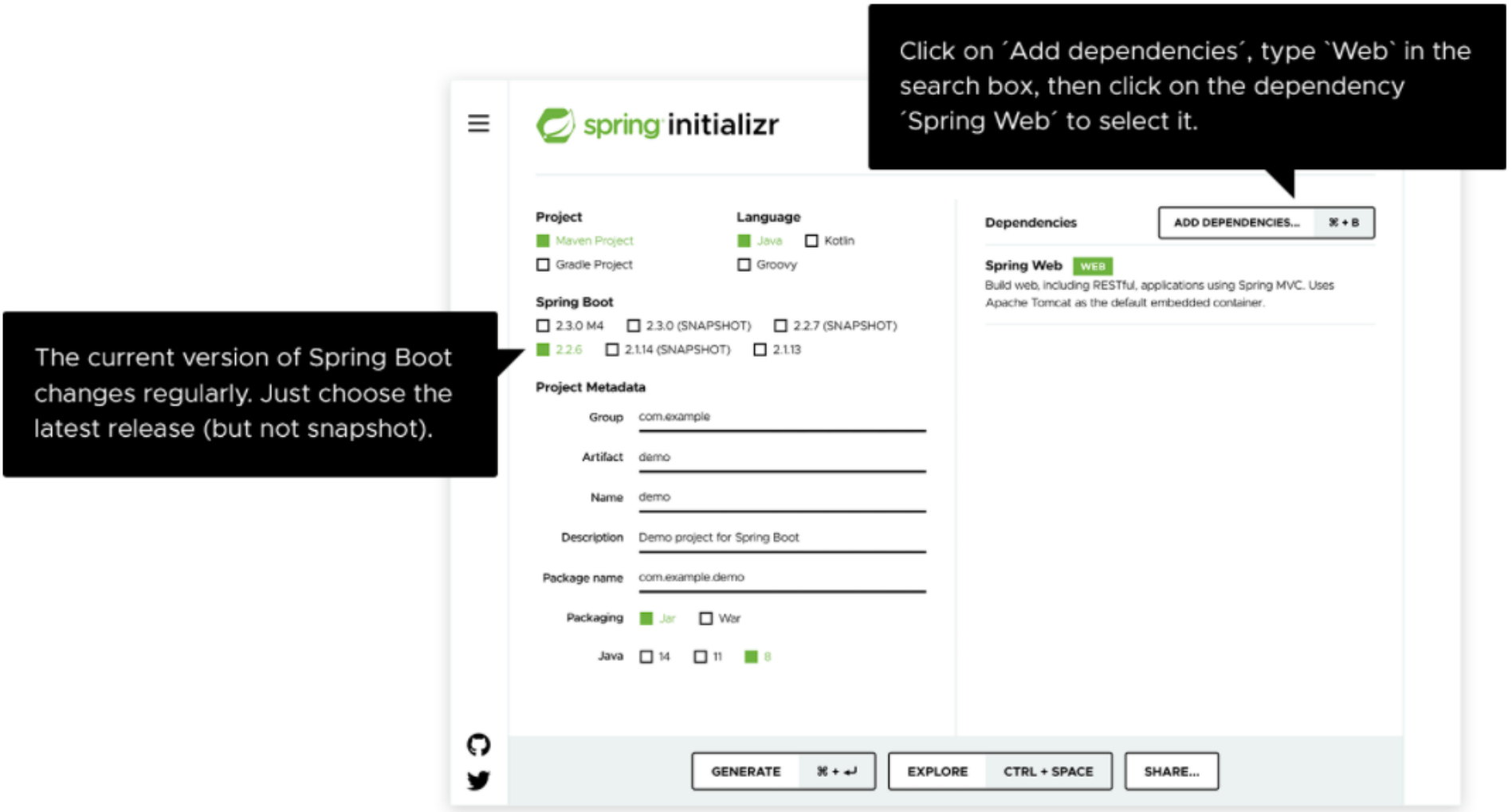


Spring Initializer (https://start.spring.io)

- **You can use Spring Initializer (https://start.spring.io) to create Spring Boot maven project, download and then import maven project in STS IDE**
- *Please note - Spring Initializer is now integrated with STS IDE, so you can create Spring Boot maven project directly inside STS IDE instead of importing it from outside.*

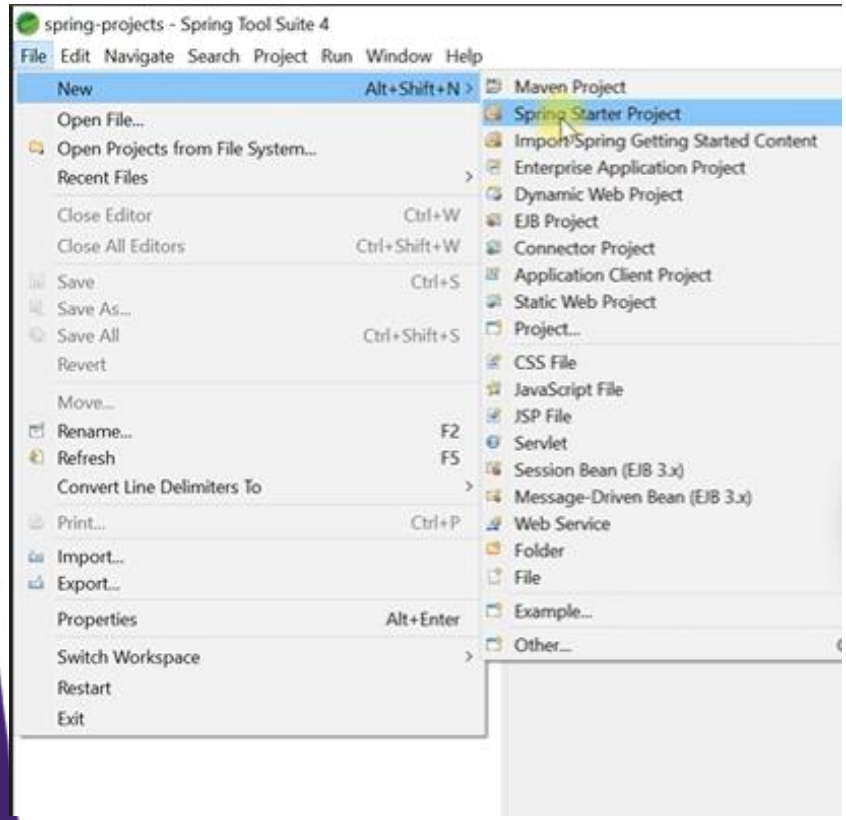


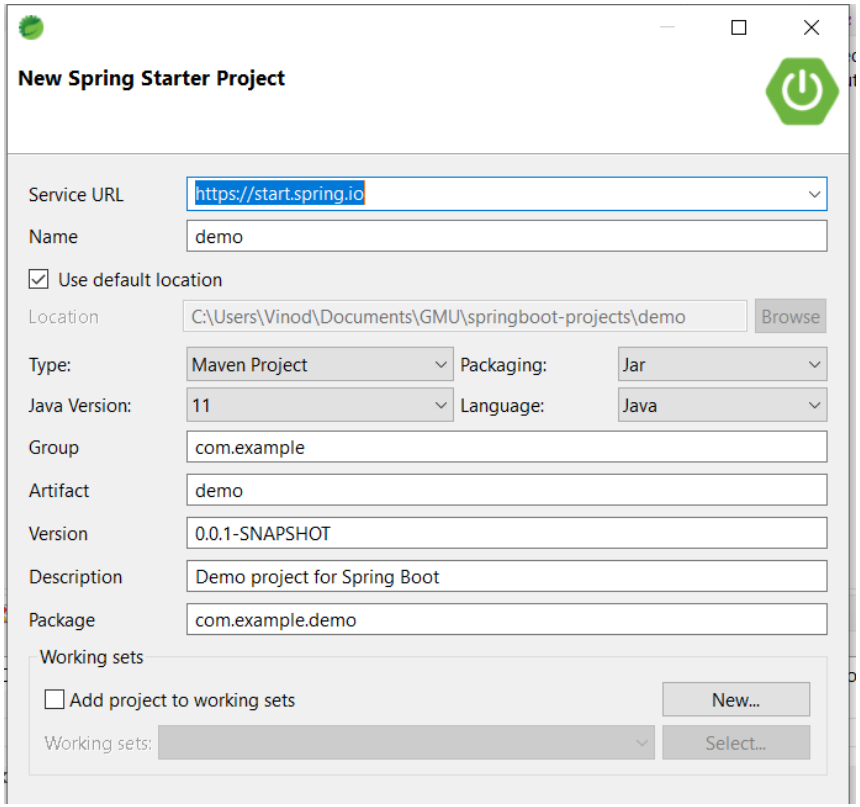
Spring Initializer (https://start.spring.io)



Creating Sping Boot Maven project using STS IDE

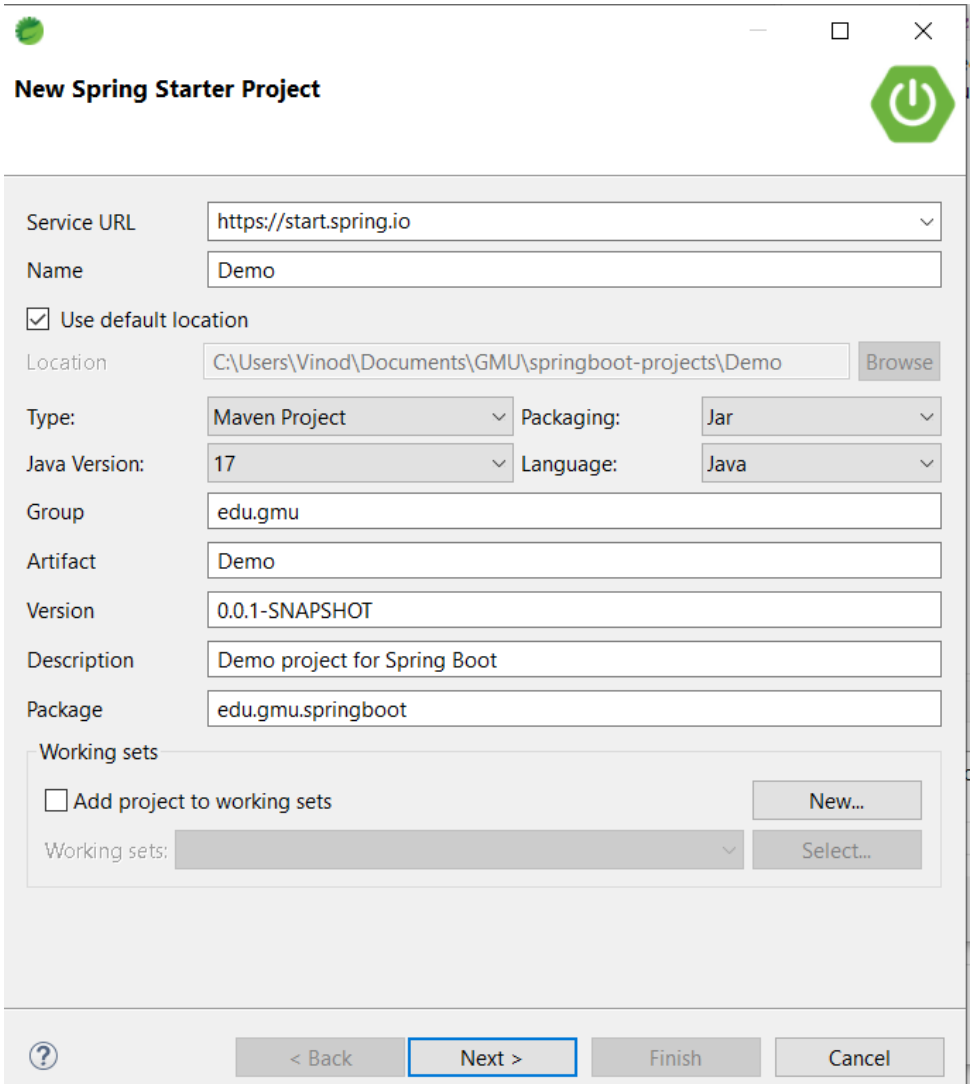
- From inside STS IDE, select:
- ***File -> New -> Spring Starter Project***





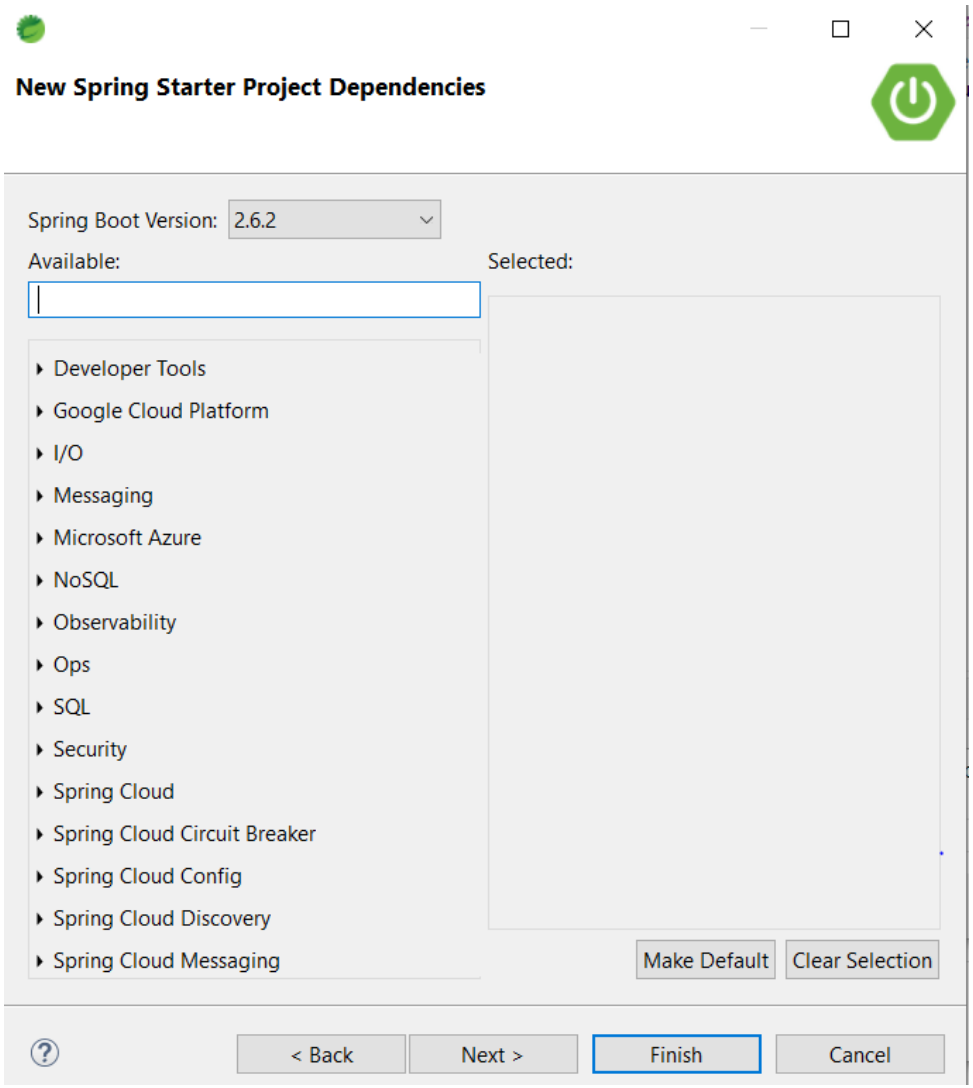
Creating Sping Boot Maven project using STS IDE

- **Select/update values for**
  - Name,
  - type,
  - Java Version,
  - packaging, language,
  - Group,
  - Description,
  - Package
- **Artifact** is same as Project Name
- **Press Next**



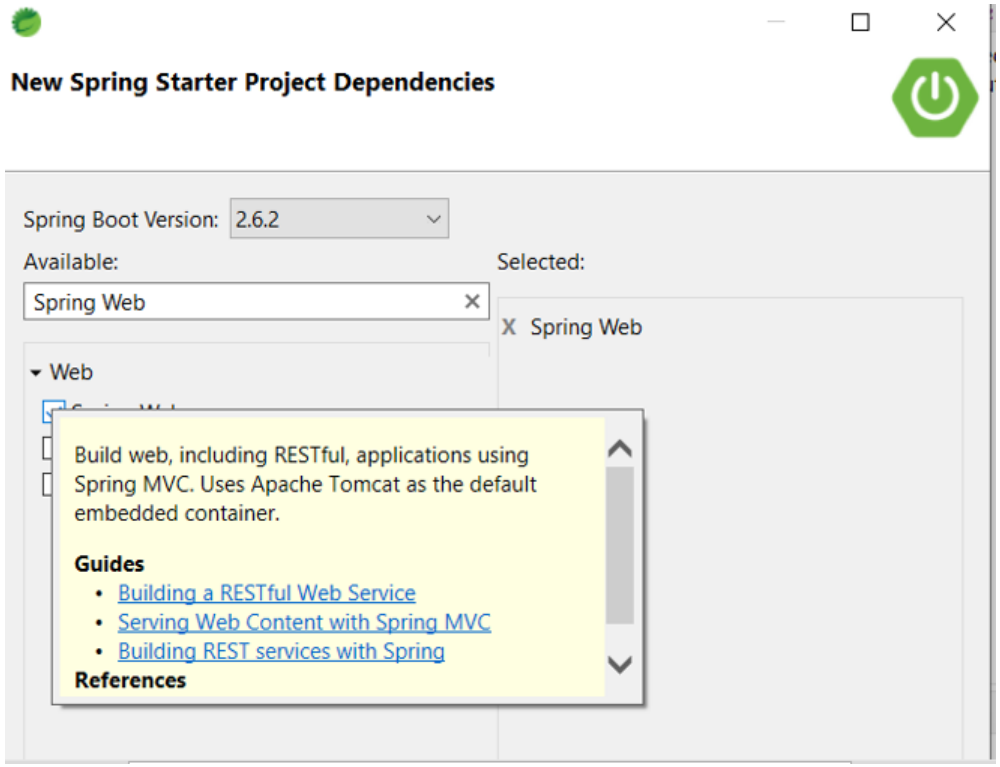
Select project dependencies

- **Select Spring Starter Project Dependencies**
- **For example,**
  - Select search and select **Spring Web** to create web application or RESTful web service.
- **And press Next**



Select project dependencies

- **Select Spring Starter Project Dependencies**
- **For example,**
  - Select search and select **Spring Web** to create web application or RESTful web service.
- **And press Next**



Select project dependencies

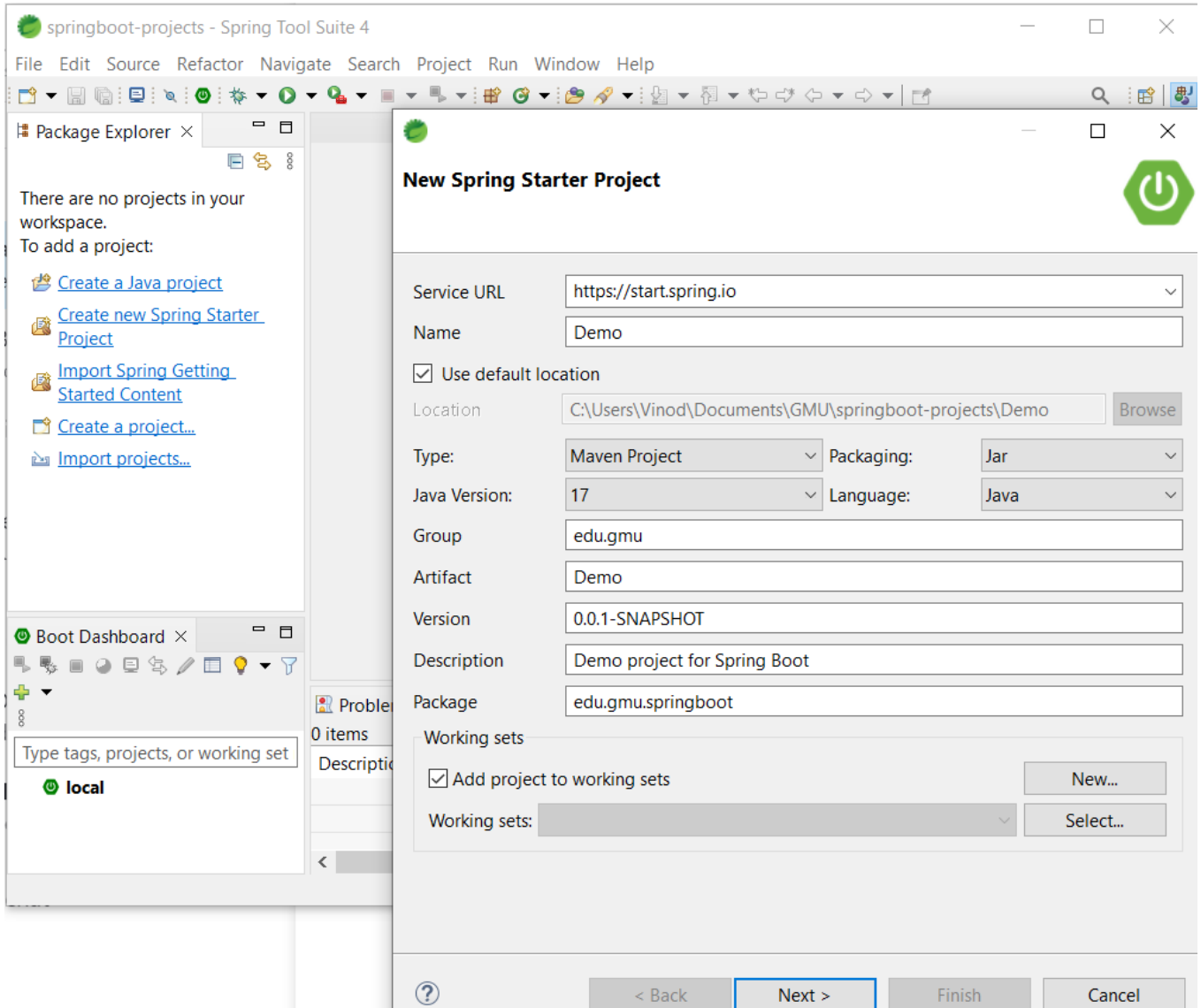
- ***Spring Web***
  - Build web, including RESTful applications using Spring MVC.
  - Uses Apache Tomcat as the default embedded container.
- **Guides**
  - Building a RESTful Web Service
  - Serving Web Content with Spring MVC
  - Building REST services with Spring
- **References**
  - Spring Boot Reference Doc

Select project dependencies

- ***Spring Data JPA***
  - is an abstraction on top of JPA and internally uses Hibernate as a JPA provider
  - Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
- **Guides**
  - Accessing Data with JPA
- **References**
  - Spring Boot Reference Doc
  - Accessing data with MySQL

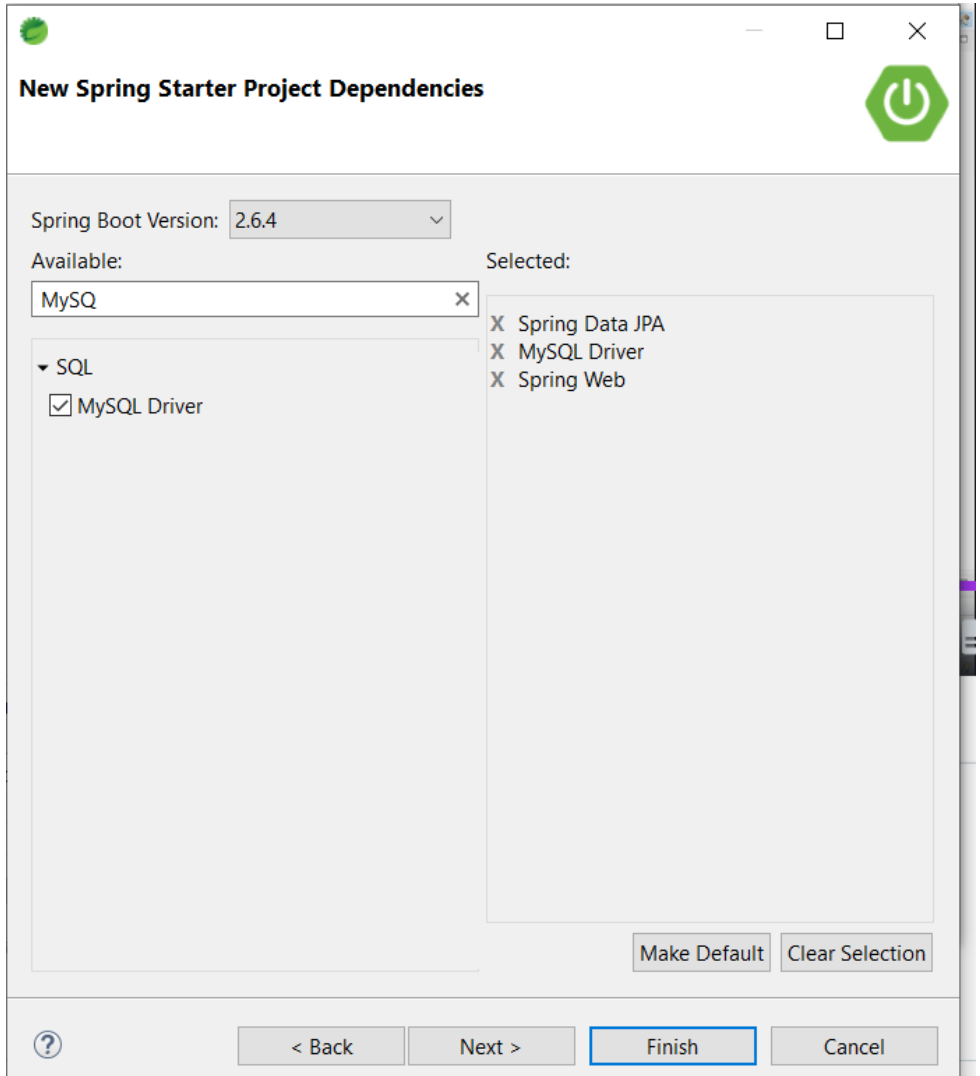
Let's create a Spring Starter Project

- *Select File->New -> Spring Starter Project,*
- *Select values for Name, Group, Select search and select Spring, Artifact, package, java version, and then And press Next*



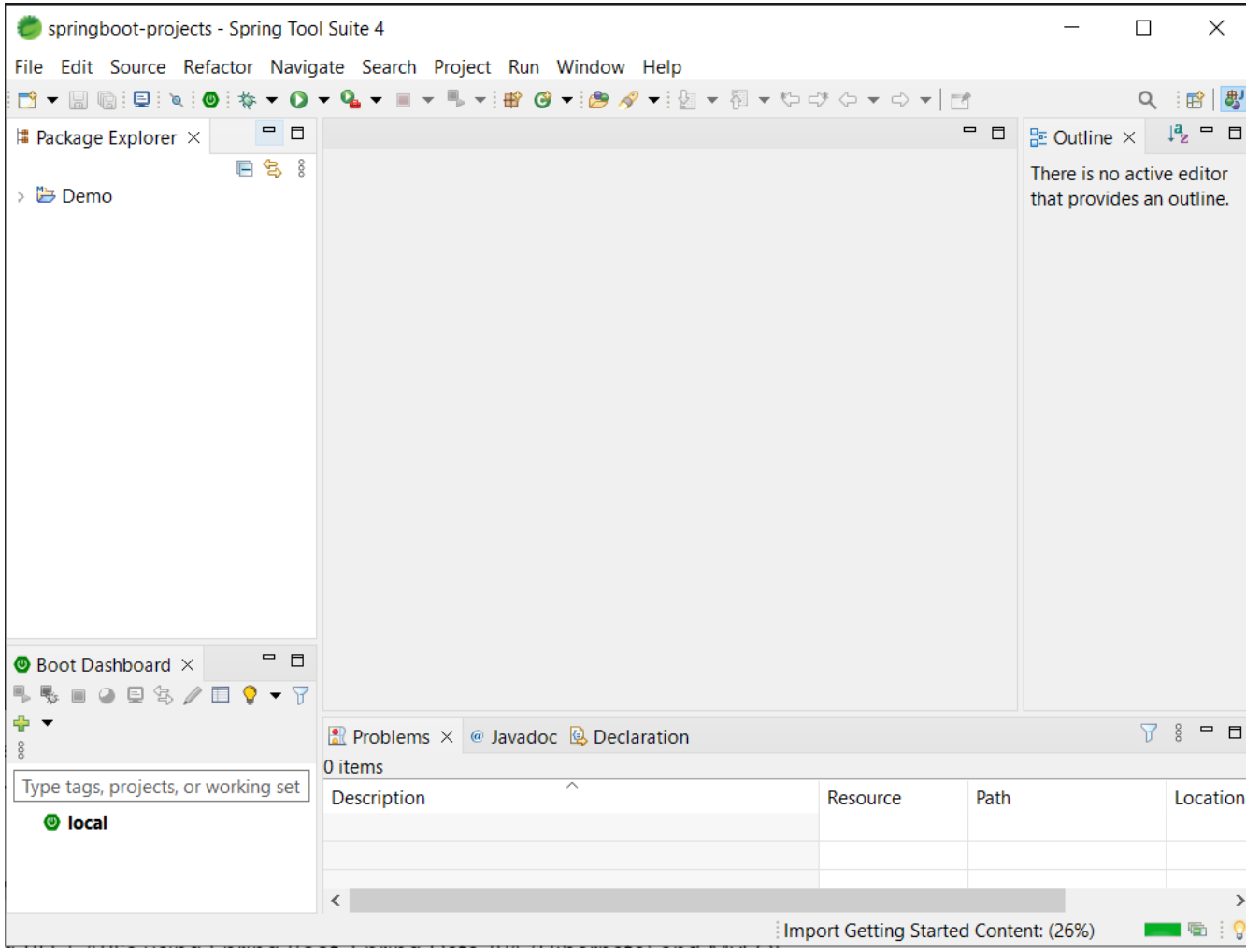
Create Spring Starter Project

- *Select Spring Starter Project Dependencies – e.g., In the Available text box, search and select Spring Web to create web application or RESTful web service.*
- *Also, select Spring Data JPA as well as MySQL Driver*
- *And press Finish*



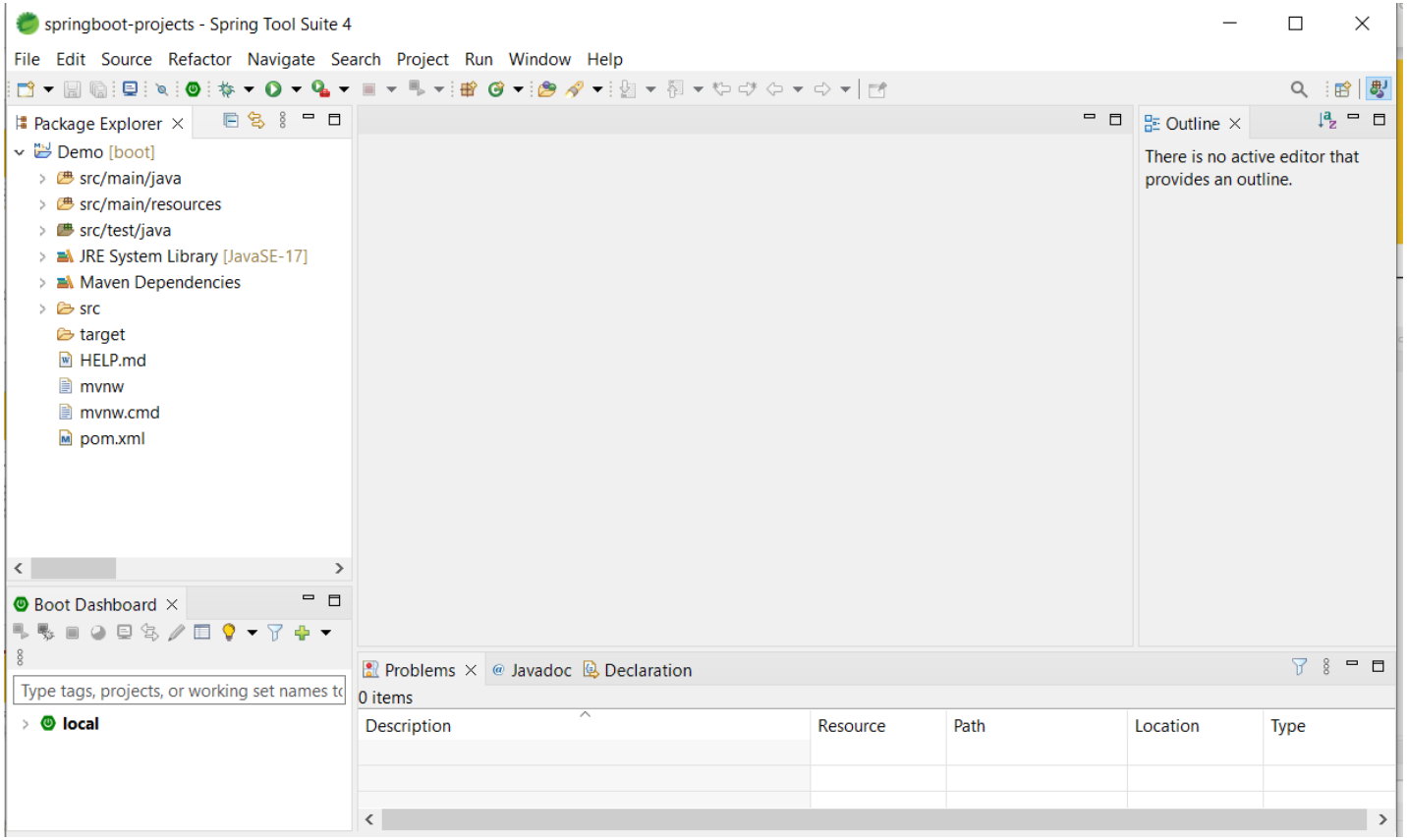
Create Spring Starter Project

Maven downloads all dependencies from the Internet

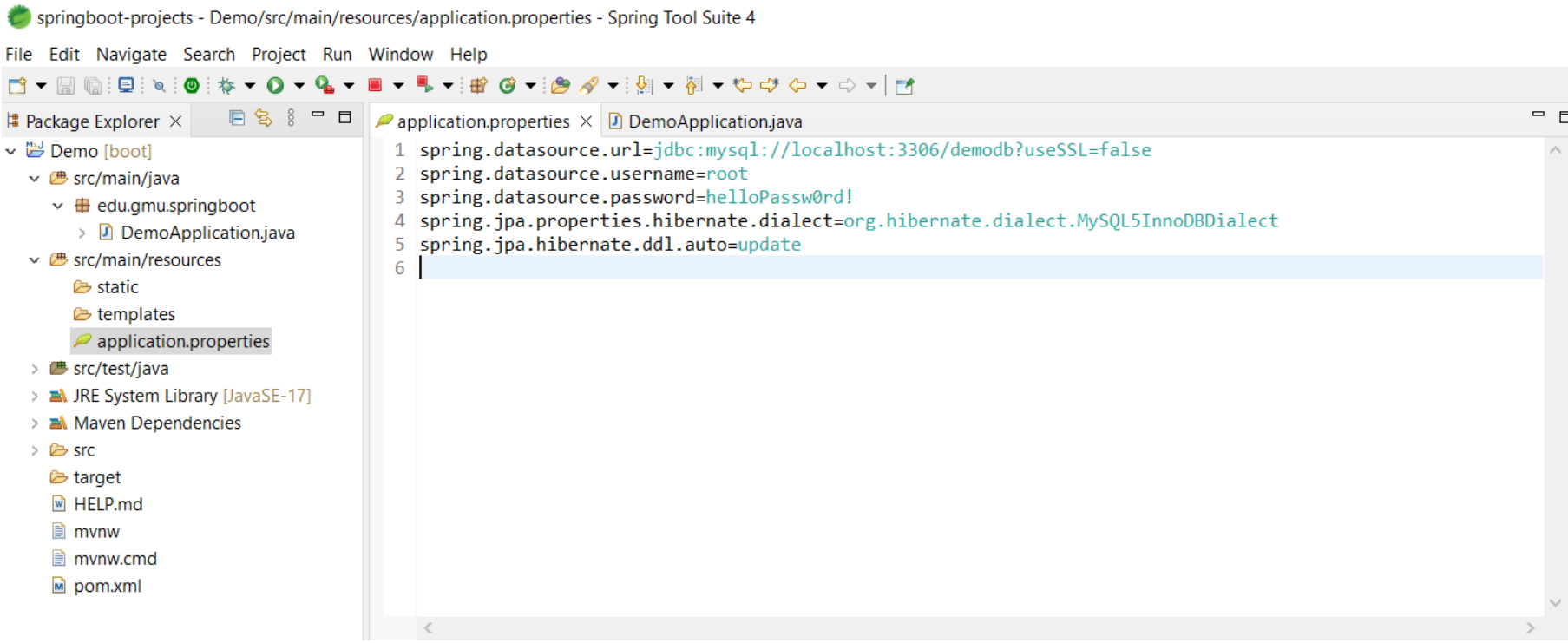


Create Spring Starter Project

Maven downloads all dependencies from the Internet and stores them under Maven Dependencies folder



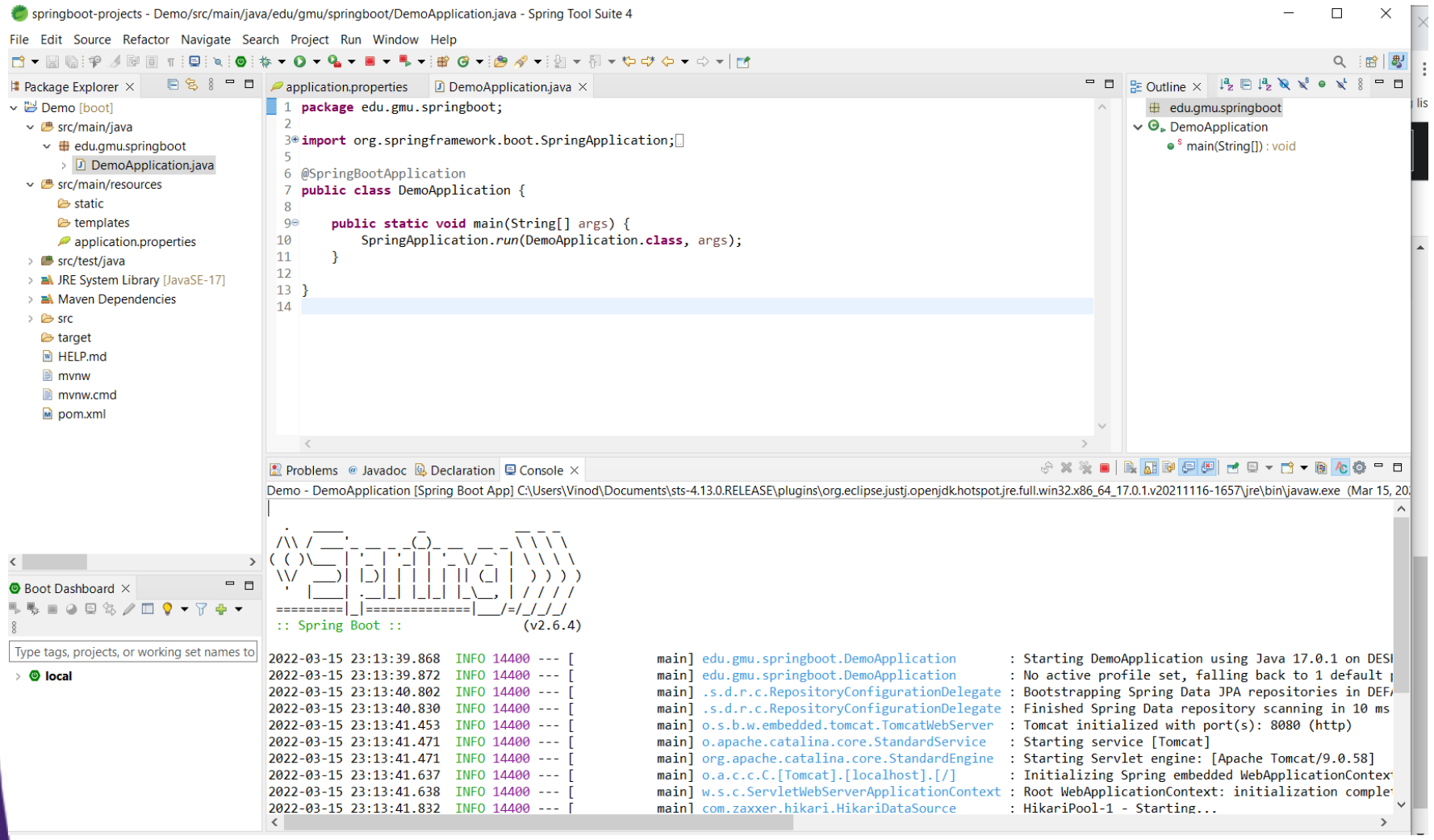
Update application.properties with database/jpa properties



Run the application

DemoApplication->Run As -> Spring Boot Application

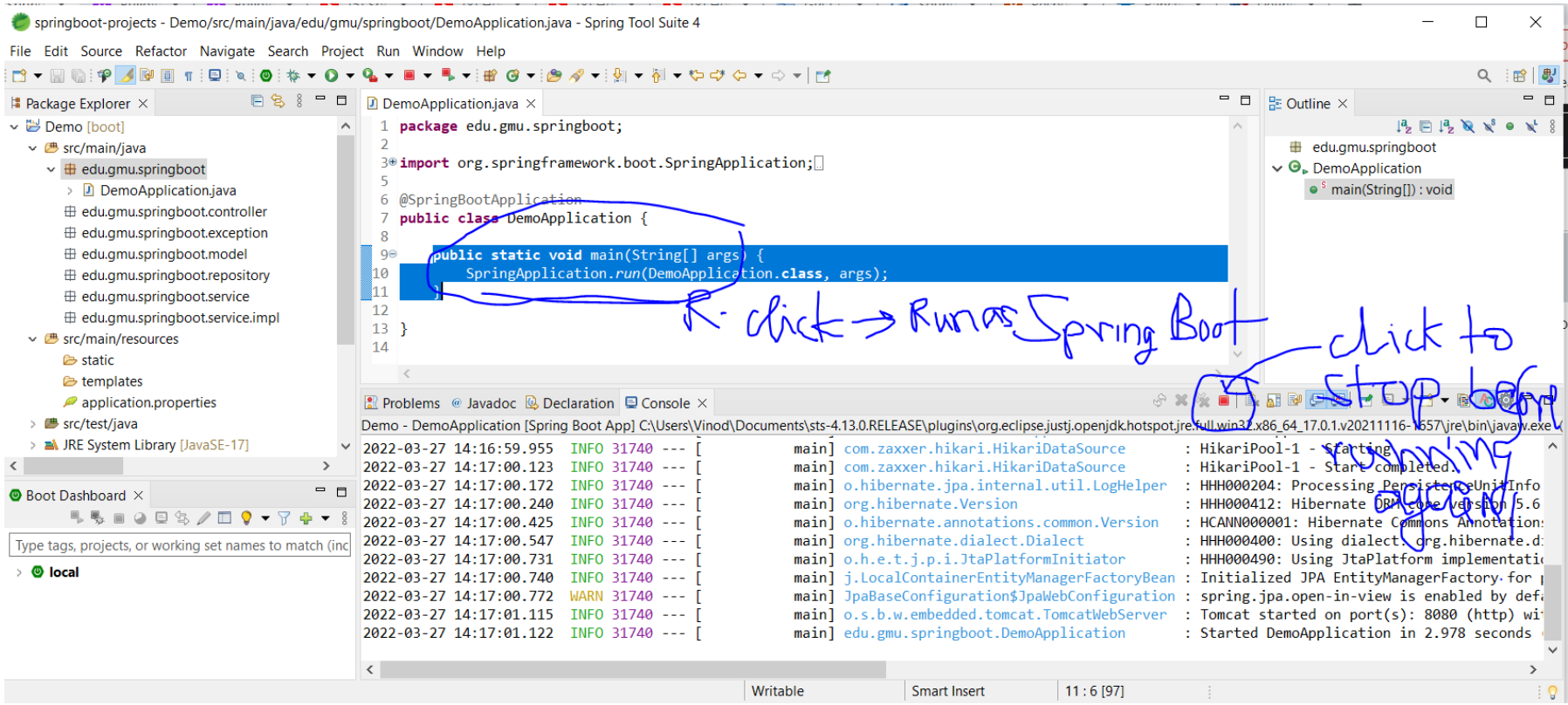
The Console shows the Spring Boot application is running on Tomcat and connected to MySQL database using JPA/Hibernate.



Run the application

DemoApplication->Run As -> Spring Boot Application

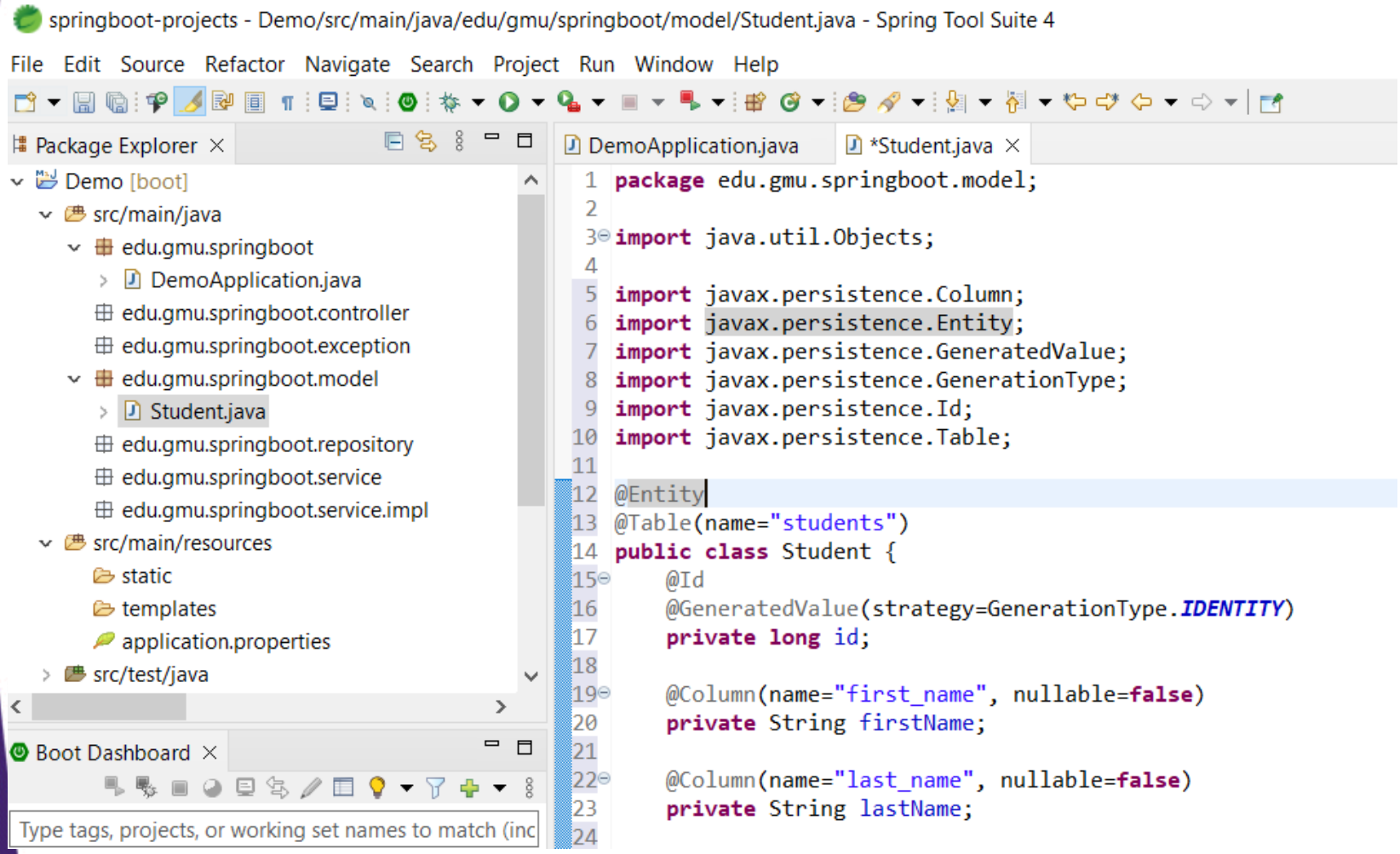
The Console shows the Spring Boot application is running on Tomcat and connected to MySQL database using JPA/Hibernate.



Create JPA Student Entity and Repository

Create JPA Entity called Student *under package* `edu.gmu.springboot.model`

(Right click on package name and Create a java class and add annotations)



Create JPA Student Entity and Repository (contd.)

Create JPA Entity called Student *under package* `edu.gmu.springboot.model`

(Right click on package name and Create a java class and add annotations)

```
25      @Column(name="email", nullable=false)
26      private String email;
27
28      public long getId() {
29          return id;
30      }
31      public void setId(long id) {
32          this.id = id;
33      }
34      public String getFirstName() {
35          return firstName;
36      }
37      public void setFirstName(String firstName) {
38          this.firstName = firstName;
39      }
40      public String getLastName() {
41          return lastName;
42      }
43      public void setLastName(String lastName) {
44          this.lastName = lastName;
45      }
```

Create JPA Student Entity and Repository (contd)

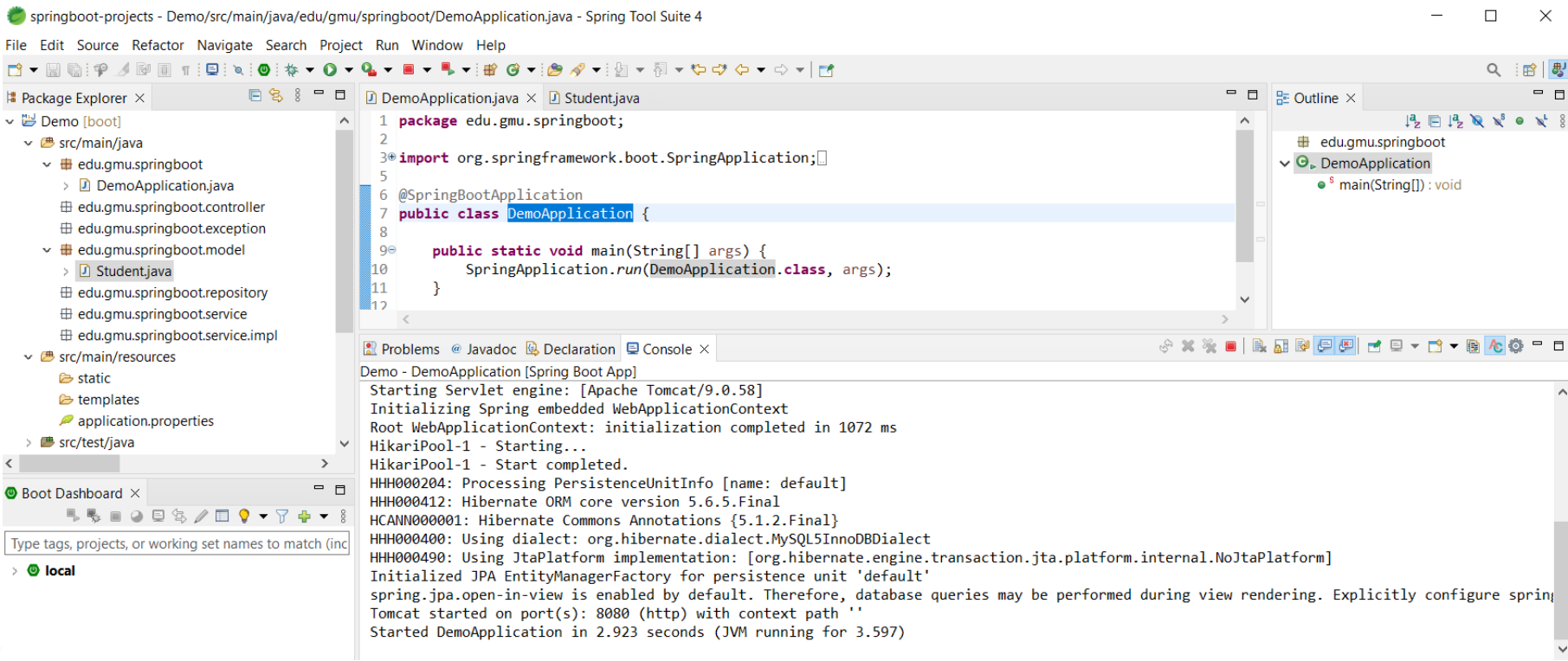
Create JPA Entity called Student *under package* `edu.gmu.springboot.model`

(Right click on package name and Create a java class and add annotations)

```
49      public void setEmail(String email) {
50          this.email = email;
51      }
52
53      @Override
54      public String toString() {
55          return "Student [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + ", emai
56      }
57      @Override
58      public int hashCode() {
59          return Objects.hash(email, firstName, id, lastName);
60      }
61      @Override
62      public boolean equals(Object obj) {
63          if (this == obj)
64              return true;
65          if (obj == null)
66              return false;
67          if (getClass() != obj.getClass())
68              return false;
69          Student other = (Student) obj;
70          return Objects.equals(email, other.email) && Objects.equals(firstName, other.firstName) && i
71                  && Objects.equals(lastName, other.lastName);
72      }
```

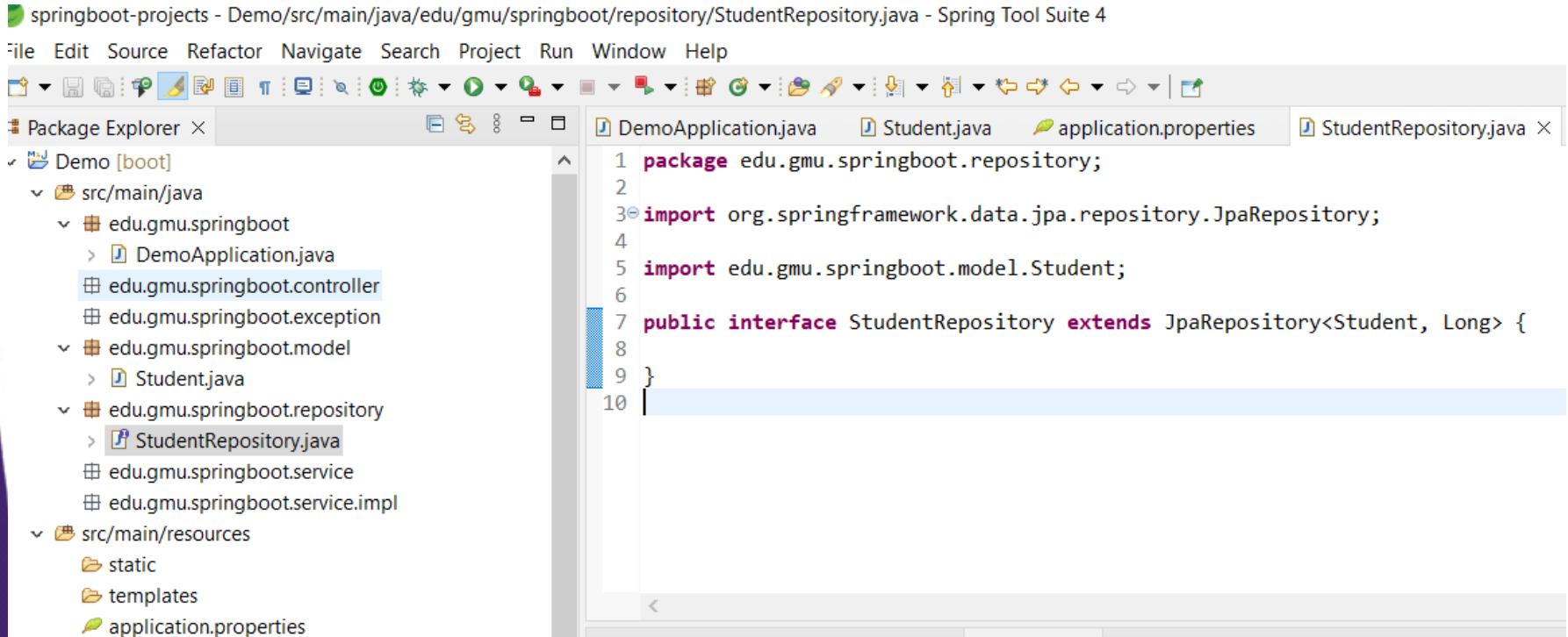
Create JPA Student Entity and Repository

If you run the application as Spring Boot, highlighting and right-clicking on DemoApplication and selecting run as Spring Boot. Should be resulted in creating students database



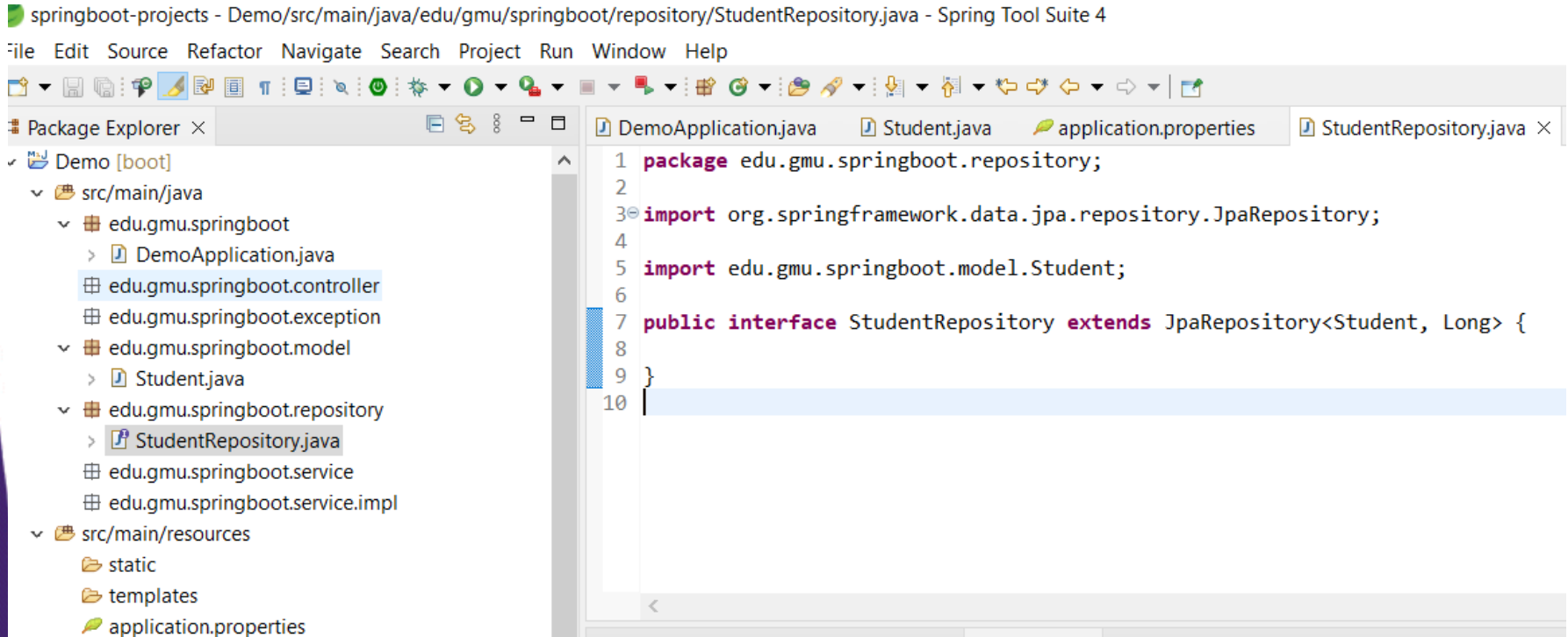
Create Student Repository Interface

- *Create StudentRepository interface under the repository package by extending JpaRepository,*
- *JpaRepository requires two parameters – type of entity, Student, and the second parameter is type of primary key, long in this example.*
- *Now StudentRepository will have CRUD methods for student.*
  - *The Student JPA repository allows to implement and perform CRUD operation on Student jpa entity*



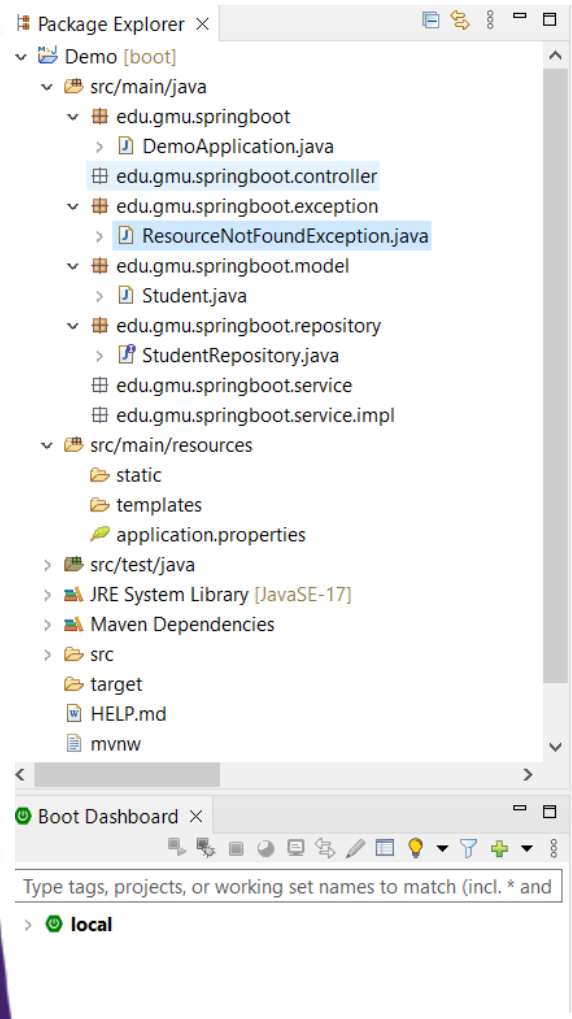
Create Student Repository Interface

- *No need to add @Repository annotation to StudentRepository interface since Spring Data JPA takes care of this automatically.*
- *Also, no need to add @Transaction annotation to StudentServiceImpl class as Spring Data JPA internally provides transaction support to all of StudentServiceImpl class methods*



ResourceNotFoundException

Lets create a custom resource not found exception in exception package



DemoApplication.java | ResourceNotFoundException.java

```java
package edu.gmu.springboot.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private String resourceName;
    private String fieldName;
    private Object fieldValue;

    public ResourceNotFoundException(String resourceName, String fieldName, Object fieldValue) {
        super(String.format("%s not found with %s : '%s'", resourceName, fieldName, fieldValue));
        this.resourceName = resourceName;
        this.fieldName = fieldName;
        this.fieldValue = fieldValue;
    }

    public String getResourceName() {
        return resourceName;
    }

    public void setResourceName(String resourceName) {
        this.resourceName = resourceName;
    }

    public String getFieldName() {
        return fieldName;
    }

    public void setFieldName(String fieldName) {
        this.fieldName = fieldName;
    }
```

ResourceNotFoundException

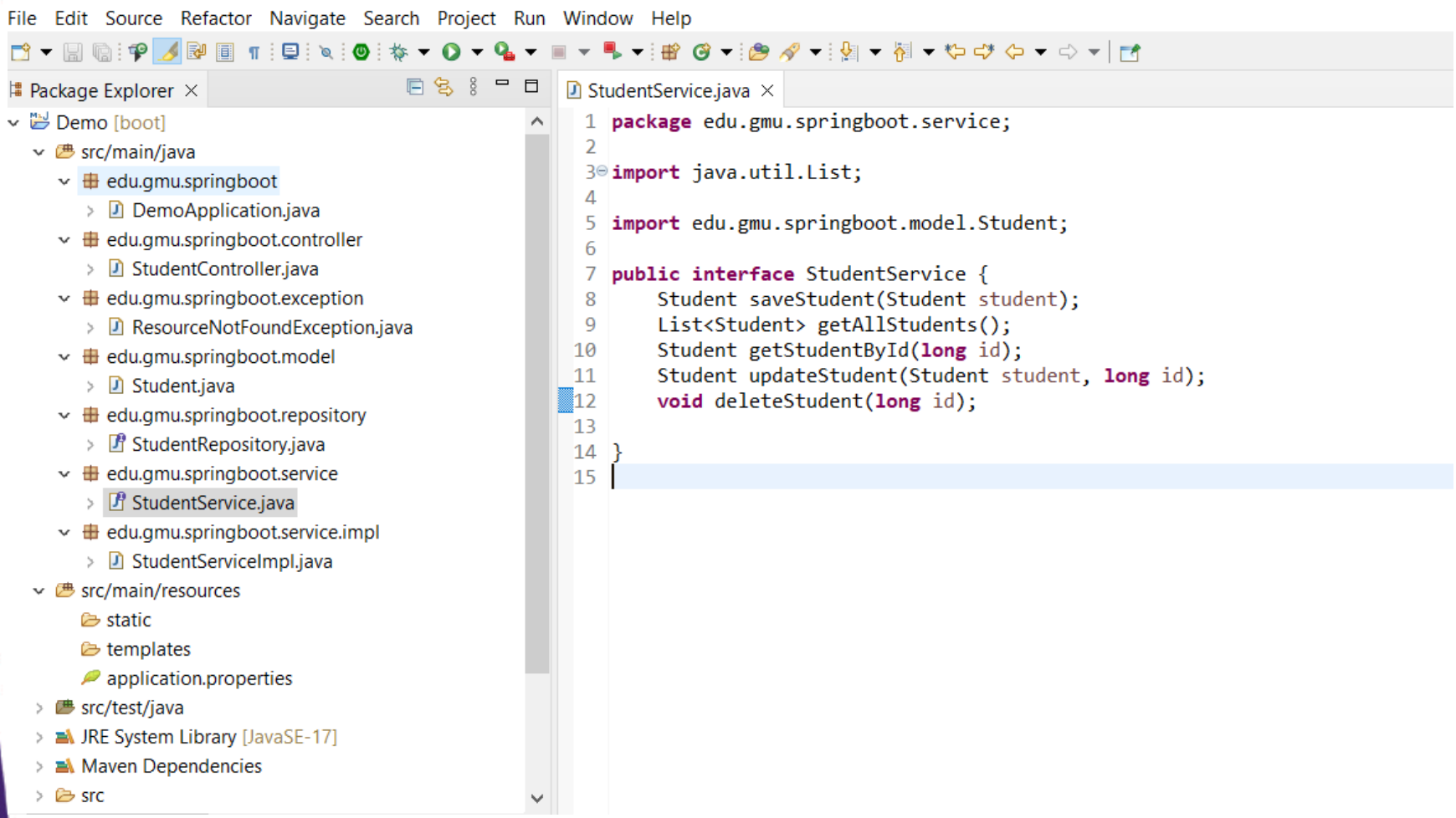
Lets create a custom resource not found exception in exception package

```java
package edu.gmu.springboot.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    private String resourceName;
    private String fieldName;
    private Object fieldValue;

    public ResourceNotFoundException(String resourceName, String fieldName, Object fieldValue) {
        super(String.format("%s not found with %s : '%s'", resourceName, fieldName, fieldValue));
        this.resourceName = resourceName;
        this.fieldName = fieldName;
        this.fieldValue = fieldValue;
    }

    public String getResourceName() {
        return resourceName;
    }

    public void setResourceName(String resourceName) {
        this.resourceName = resourceName;
    }

    public String getFieldName() {
        return fieldName;
    }

    public void setFieldName(String fieldName) {
        this.fieldName = fieldName;
    }

    public Object getFieldValue() {
        return fieldValue;
    }

    public void setFieldValue(Object fieldValue) {
        this.fieldValue = fieldValue;
    }


}

```

Service Layer Implementation

Controller layer depends on Service layer

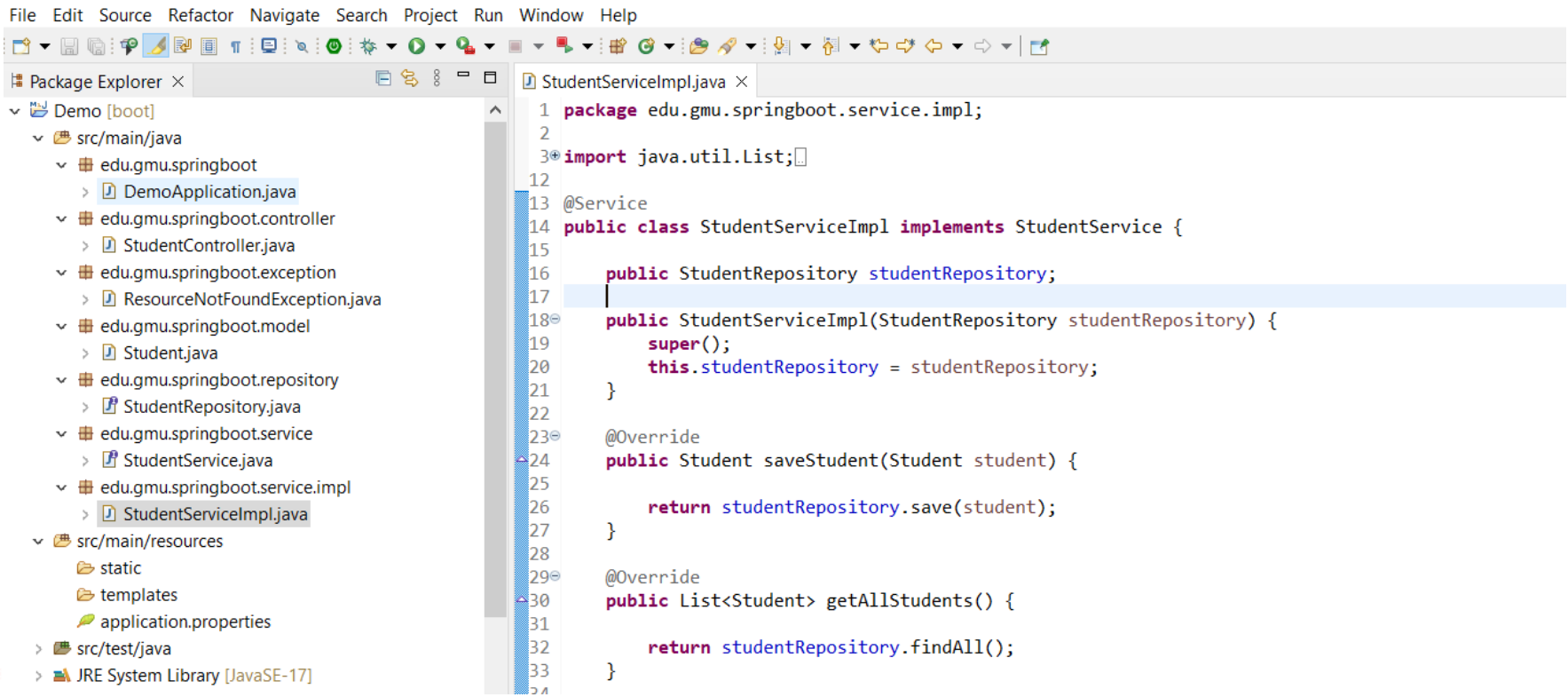
Create StudentService interface inside edu.gmu.sprintboot.service package



Service Layer Implementation

Controller layer depends on Service layer

Create StudentServiceImpl class that implments StudentService



Service Layer Implementation

Controller layer depends on Service layer

Create StudentServiceImpl interface that implments StudentService

StudentServiceImpl.java

```
34
35      @Override
36      public Student getStudentById(long id) {
37          //Please note that repository's findById() method returns a Optional object (of java.util)
38          //which is a generic, hence we need to pass the type Student to the returned Optional object
39          Optional<Student> student = studentRepository.findById(id);
40
41          //Check if the returned Optional object contains the student, if so, return the student object
42          if (student.isPresent()){
43            return student.get();  // here get() method returns the content of the Optional object.
44          }else {
45             //this internally creates a message. Check the ResourceNotFoundException for details.
46             throw new ResourceNotFoundException("Student", "Id", id);
47          }
48
49          //Above code can be replaced by one line code with lambda expresssion
50          // return studentRepository.findById(id).orElseThrow(() -> new ResourceNotFoundException("Employee", "Id", id));
51      }
52
53      @Override
54      public Student updateStudent(Student student, long id) {
55      //check whether student with given id exists in database
56        Student existingStudent = studentRepository.findById(id).orElseThrow(
57              () -> new ResourceNotFoundException("Student", "Id", id));
58
59           existingStudent.setFirstName(student.getFirstName());
60           existingStudent.setLastName(student.getLastName());
61           existingStudent.setEmail(student.getEmail());
62
63           //save existing student to database
64           studentRepository.save(existingStudent);
65
66           return existingStudent;   //return the existing student to the controller layer
67      }
68
69      @Override
70      public void deleteStudent(long id) {
71          //check whether the student exists in the database
72            studentRepository.findById(id).orElseThrow( () -> new ResourceNotFoundException("Student", "Id", id));
73
74            studentRepository.deleteById(id);
75      }
76  }
```

Controller Layer Implmentation

- **Let's Create StudentController under edu.gmu.springboot.controller and anotate it with @RestController,**
  - This is a convenient annotation that combines @Controller and @ResponseBody
  - This eliminates the need to annotate every request handling method of the controller class with the @ResponseBody
- **Also, inject StudentService dependency**
  - Need to inject StudentService using constructor based dependency injection
- **Use @RequestMapping annotation to provide base url for the controller (StudentController)**

StudentController

StudentController.java

```java
package edu.gmu.springboot.controller;

import java.util.List;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import edu.gmu.springboot.model.Student;
import edu.gmu.springboot.service.StudentService;

@RestController
@RequestMapping("/api/students")   //defines the base url for all apis
public class StudentController {
    private StudentService studentService;

    //inject studentService object using constructor based dependency injection
    public StudentController(StudentService studentService) {
        super();
        this.studentService = studentService;
    }

```

StudentController: saveStuedent()

- **Now let's build create student REST API, saveStudent() to save Student data to the database. Key Annotations:**
- **@PostMapping**
  - Anotating saveStudent() with @PostMapping allows this REST API (method) to handle http POST requests.
- **@RequestBody**
  - @RequestBody annotation allows to retrieve the request's body and autommatically convert it to Java object
- **ResponseEntity**
  - Return type ResponseEntity is used to return response for the rest api
  - ResponseEntity allows to provide complete response details in this class, including status, header.
  - This api created the resource, hence we pass HttpStatus.CREATED
- **The url to access above method will be http://localhost:8080/api/students**
  - when invoked with http POST method and also send data for request body, such as json object with key value pairs.
  - You can use Postman Rest client to test the RESTful APIs

StudentController: saveStuedent()

```
19 @RestController
20 @RequestMapping("/api/students")   //defines the base url for all apis
21 public class StudentController {
22     private StudentService studentService;
23
24     //inject studentService object using constructor based dependency injection
25     public StudentController(StudentService studentService) {
26         super();
27         this.studentService = studentService;
28     }
29
30     //Build create student api that handles post requests
31
32     @PostMapping
33     public ResponseEntity<Student> saveStudent(@RequestBody Student student) {
34       return new ResponseEntity<Student>(studentService.saveStudent(student), HttpStatus.CREATED);
35     }
36
```

StudentController: getAllStuedents()

- **Now let's create getAll Students() REST API to get all students from the database table,**
  - getAllStudents() REST API within the StudentController.
  - Since StudentController depends on StudentService, so add getAllStudents() method in StudentService interface, and implement the getAllStudents() method in StudentServiceImpl class.
- **public List<Student> getAllStudents() rest api can be accessed using the url**
  - http://localhost:8080/api/students, which by default uses http GET method.
  - This may return array of JSON objects.

StudentController: getAllStuedents()

```
19 @RestController
20 @RequestMapping("/api/students")   //defines the base url for all apis
21 public class StudentController {
22     private StudentService studentService;
23 
24     //inject studentService object using constructor based dependency injection
25     public StudentController(StudentService studentService) {
26         super();
27         this.studentService = studentService;
28     }
29 
30     //Build create student api that handles post requests
31 
32     @PostMapping
33     public ResponseEntity<Student> saveStudent(@RequestBody Student student) {
34       return new ResponseEntity<Student>(studentService.saveStudent(student), HttpStatus.CREATED);
35     }
36 
37    //Create get all students REST API
38    @GetMapping
39    public List<Student> getAllStudents(){
40       return studentService.getAllStudents();
41    }
```

StudentController: getStuedentById()

- **Now, let's build a REST API that returns a specific Student for a given ID**
  - Since the controller layer depends on Service layer, so need to add the corresponding method in StudentService
- **In the next example {id} is url template variable**
  - a syntax to get path variable which is dynamic and appended to the end of URI, e.g., 1 at the end of the following URI; allows url to be http://localhost:8080/api/employees/1

StudentController: getStuedentById()

StudentController.java

```
19 @RestController
20 @RequestMapping("/api/students")   //defines the base url for all apis
21 public class StudentController {
22     private StudentService studentService;
23
24     //inject studentService object using constructor based dependency injection
25     public StudentController(StudentService studentService) {
26         super();
27         this.studentService = studentService;
28     }
29
30     //Build create student api that handles post requests
31
32     @PostMapping
33     public ResponseEntity<Student> saveStudent(@RequestBody Student student) {
34       return new ResponseEntity<Student>(studentService.saveStudent(student), HttpStatus.CREATED);
35     }
36
37    //Create get all students REST API
38    @GetMapping
39    public List<Student> getAllStudents(){
40       return studentService.getAllStudents();
41    }
42
43    //create get student by id REST API.
44
45    @GetMapping("{id}")
46    public ResponseEntity<Student> getStudentById(@PathVariable("id") long studentId){
47       return new ResponseEntity<Student>(studentService.getStudentById(studentId), HttpStatus.OK);
48
49    }
```

StudentController: updateStuedent()

- **Now to implement update Student REST API**
  - Since the Controller layer depends on Service Layer, need to update service layer first
  - So, add new method updateStudent() in StudentService interface and implment it into StudentServiceImpl
- **@PutMapping annotation makes this method as a RESTful API that can handle http PUT requests.**
  - "{id}" used with PutMapping is a syntax to bind with path variable value
  - client can use URL of sort http://localhost:8080/api/employees/1
- **Return an instance of ResponseEntity class with two parameters- body and status**

StudentController: updateStuedent()

```
50
51      //Now to implement update Student REST API
52
53      @PutMapping("{id}")
54      public ResponseEntity<Student> updateStudent(@PathVariable("id") long id, @RequestBody Student student){
55              return new ResponseEntity<Student>(studentService.updateStudent(student, id), HttpStatus.OK);
56
57      }
58
59      // Create delete student REST API.
60
61      @DeleteMapping("{id}")
62      public ResponseEntity<String> deleteStudent(@PathVariable("id") long id){
63         //delete student from database
64         studentService.deleteStudent(id);
65
66         return new ResponseEntity<String>("Employee deleted successfully.", HttpStatus.OK);
67
68      }
69  }
```

StudentController: deleteStuedent()

- **Finally let's create delete student REST API.**
  - Need to make change in the service layer first since the controller layer depends on Service layer
- **This method becomes REST api by using the annotation @DeleteMapping**
- **API can be accessed using the following URL using http method delete**
  - http://localhost:8080/api/students/1

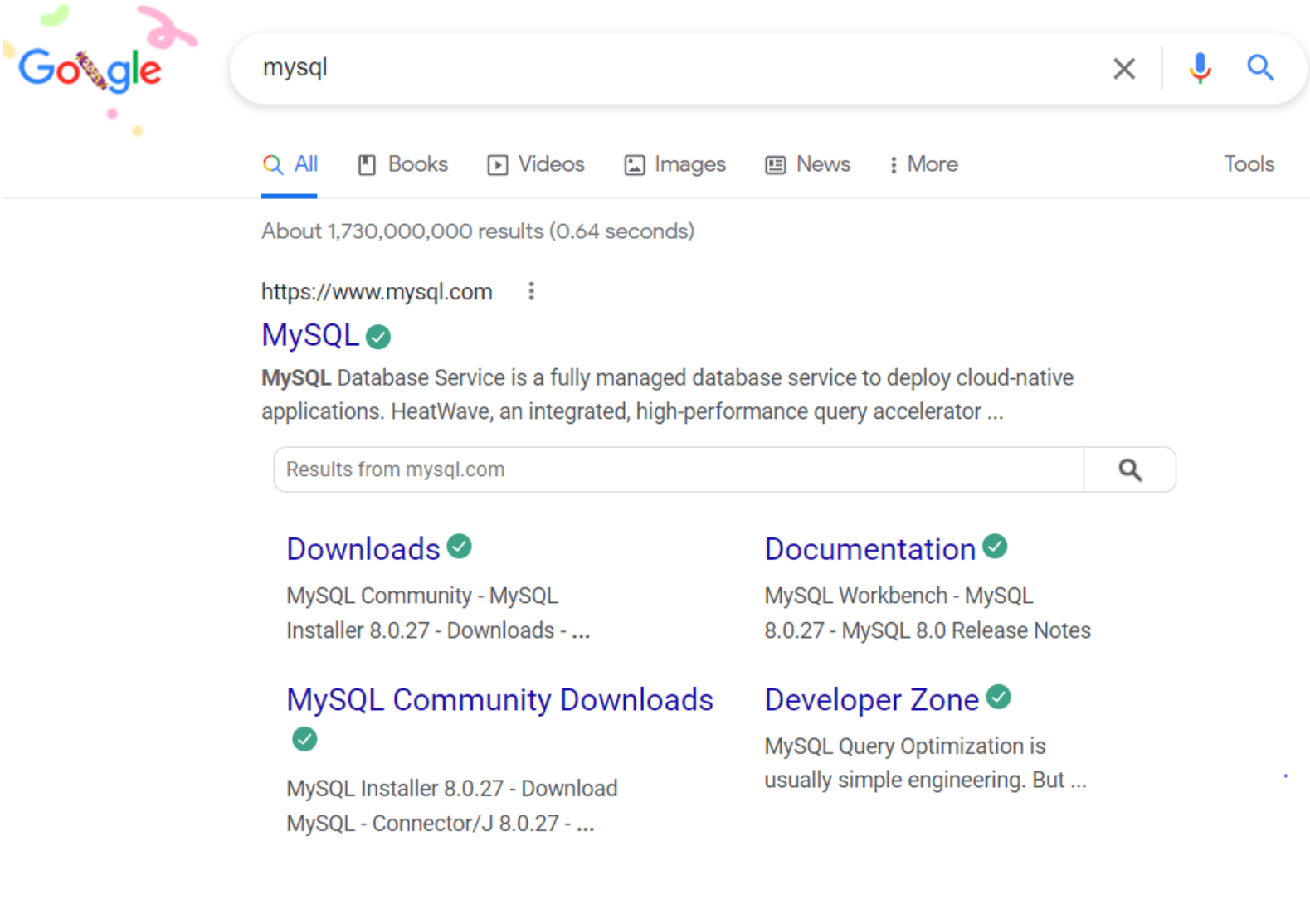
StudentController: deleteStuedent()

```
50
51      //Now to implement update Student REST API
52
53      @PutMapping("{id}")
54      public ResponseEntity<Student> updateStudent(@PathVariable("id") long id, @RequestBody Student student){
55              return new ResponseEntity<Student>(studentService.updateStudent(student, id), HttpStatus.OK);
56
57      }
58
59      // Create delete student REST API.
60
61      @DeleteMapping("{id}")
62      public ResponseEntity<String> deleteStudent(@PathVariable("id") long id){
63         //delete student from database
64         studentService.deleteStudent(id);
65
66         return new ResponseEntity<String>("Employee deleted successfully.", HttpStatus.OK);
67
68      }
69  }
```

Installing MySQL Server and Workbench on Windows 10

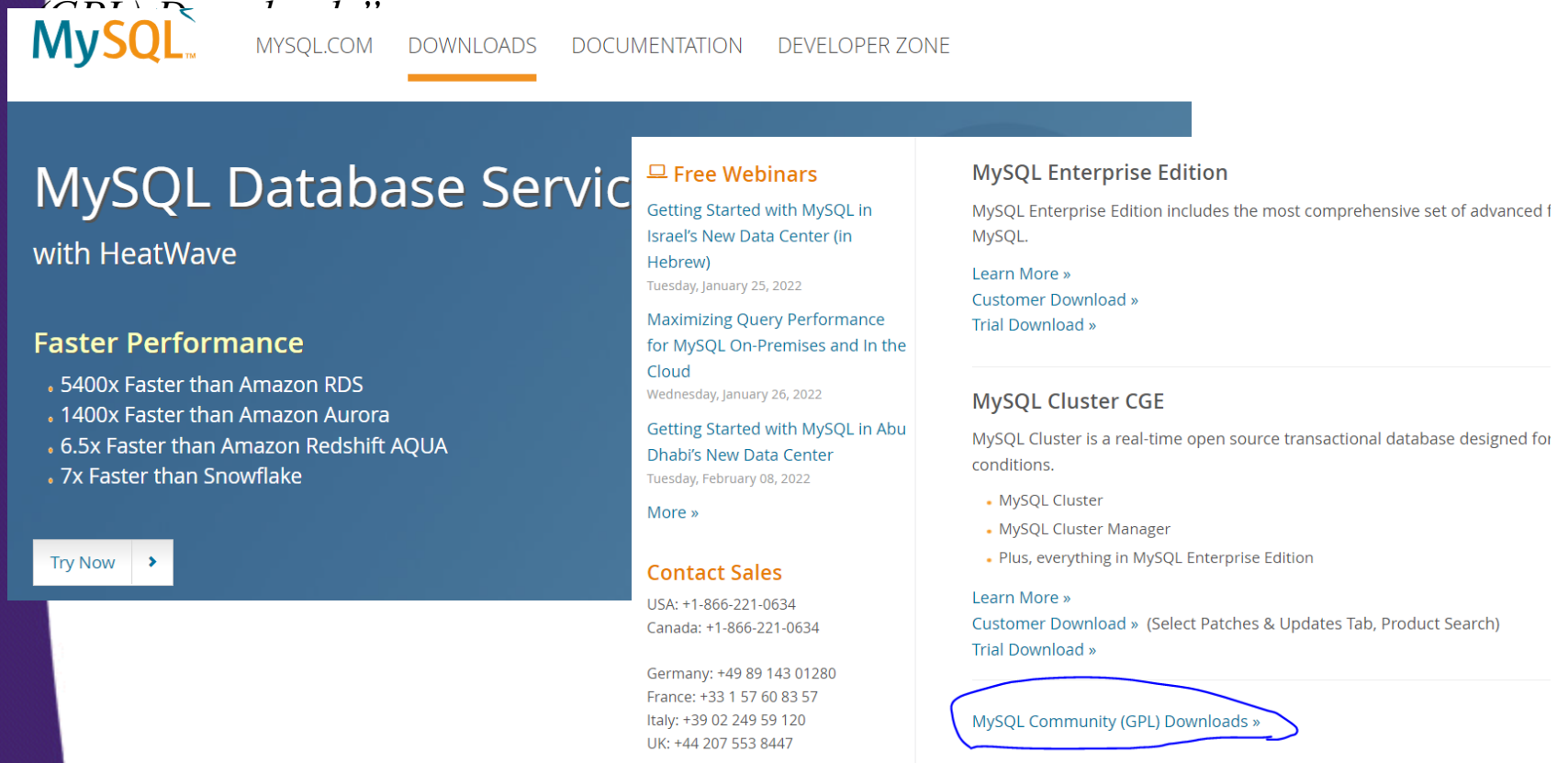
Installing MySQL Server and Workbench on Windows 10

Google search mysql and select the first link.

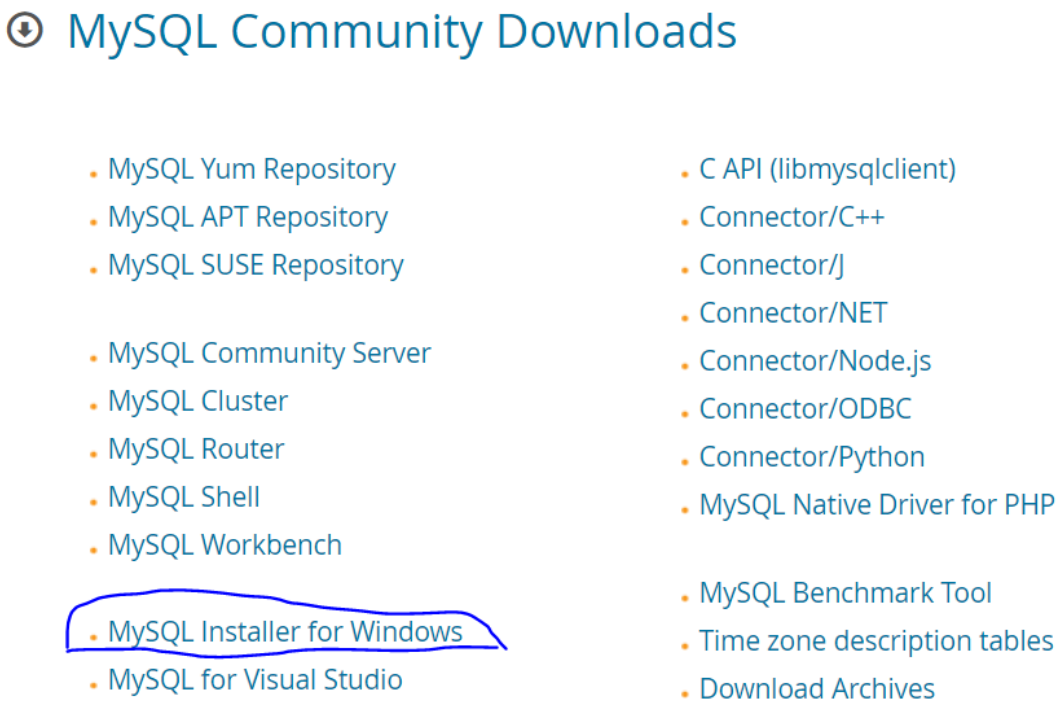


Google search mysql and select the first link.

Click on DOWNLOADS and then click on "MySQL Community (GPL) Downloads"

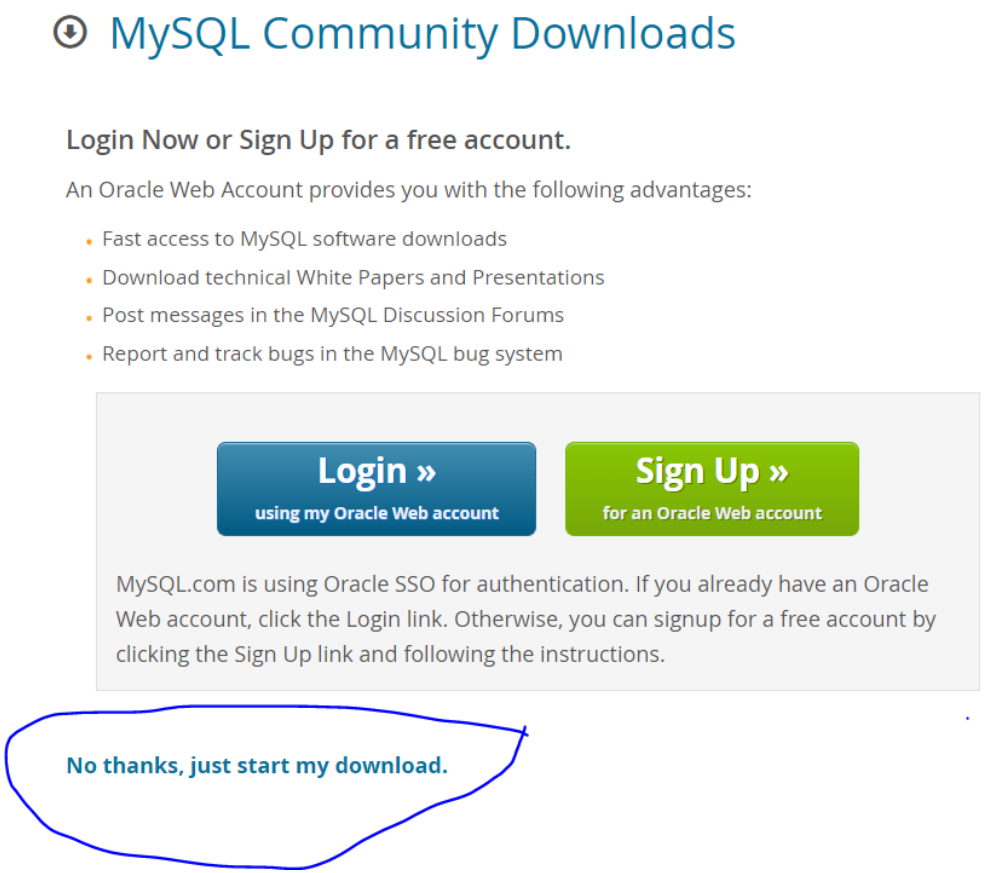


On "MySQL Community Downloads" page, click on "MySQL Installer for Windows"

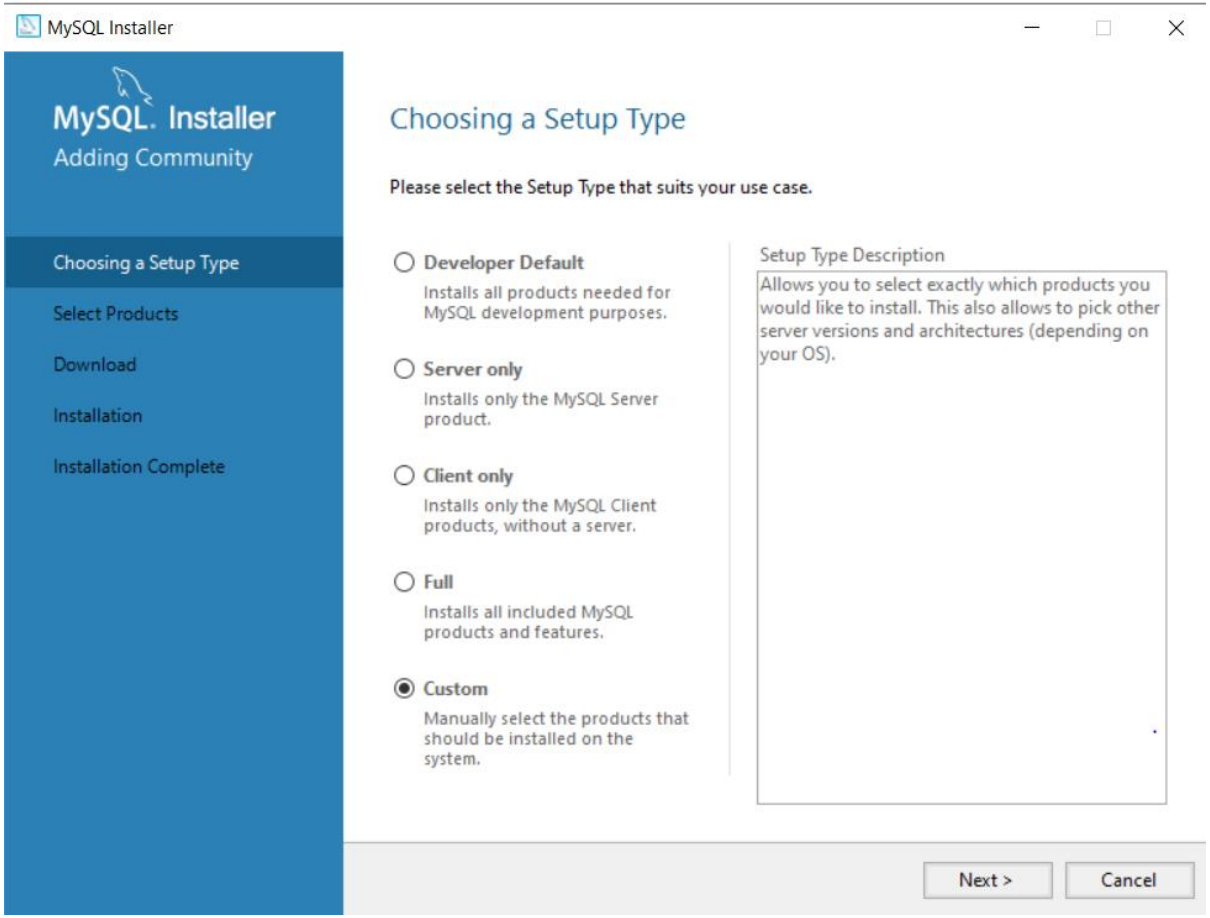


On "MySQL Installer for Windows", download 'Windows (x86, 32-bit, MSI Installer (the 2nd link below) – this works for 62-bit O/S as well.

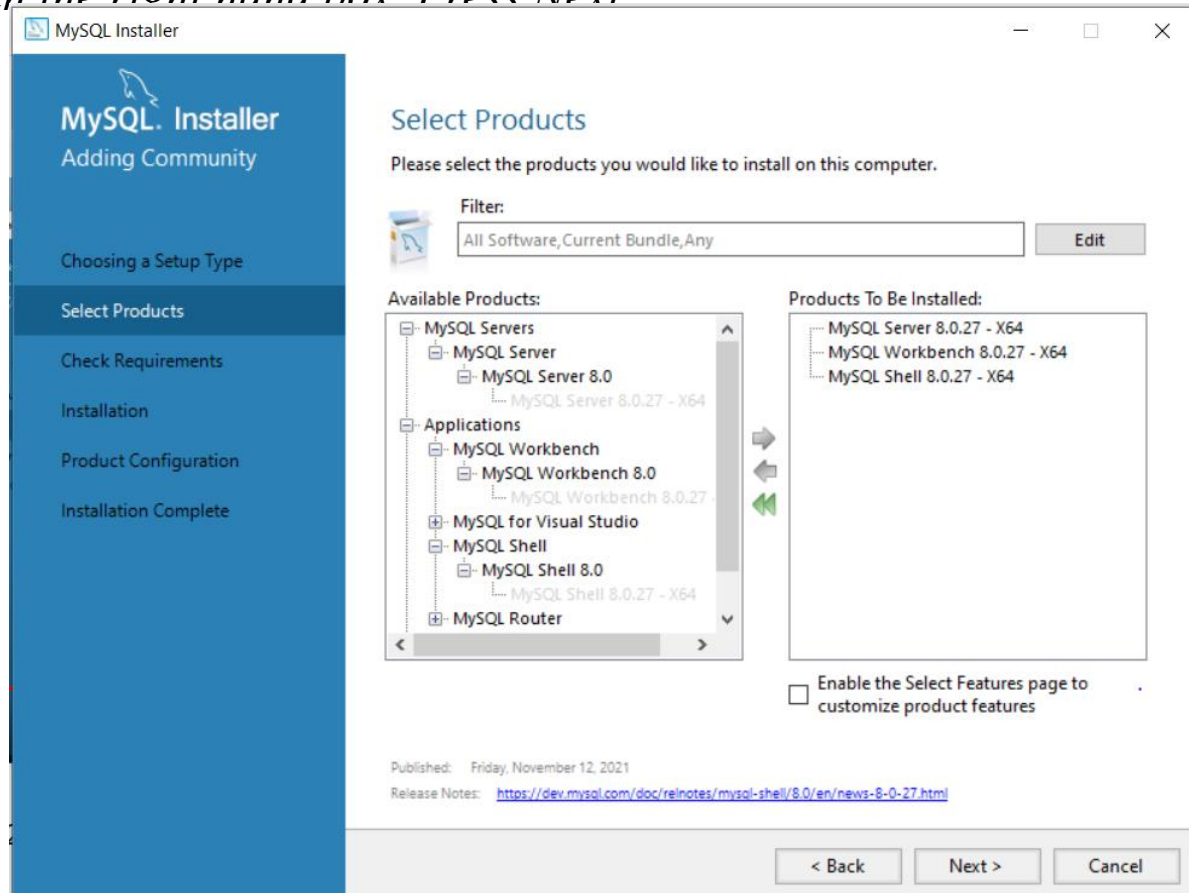
You do not need to Login or Sign up. Just Press "No thanks, just start my download."



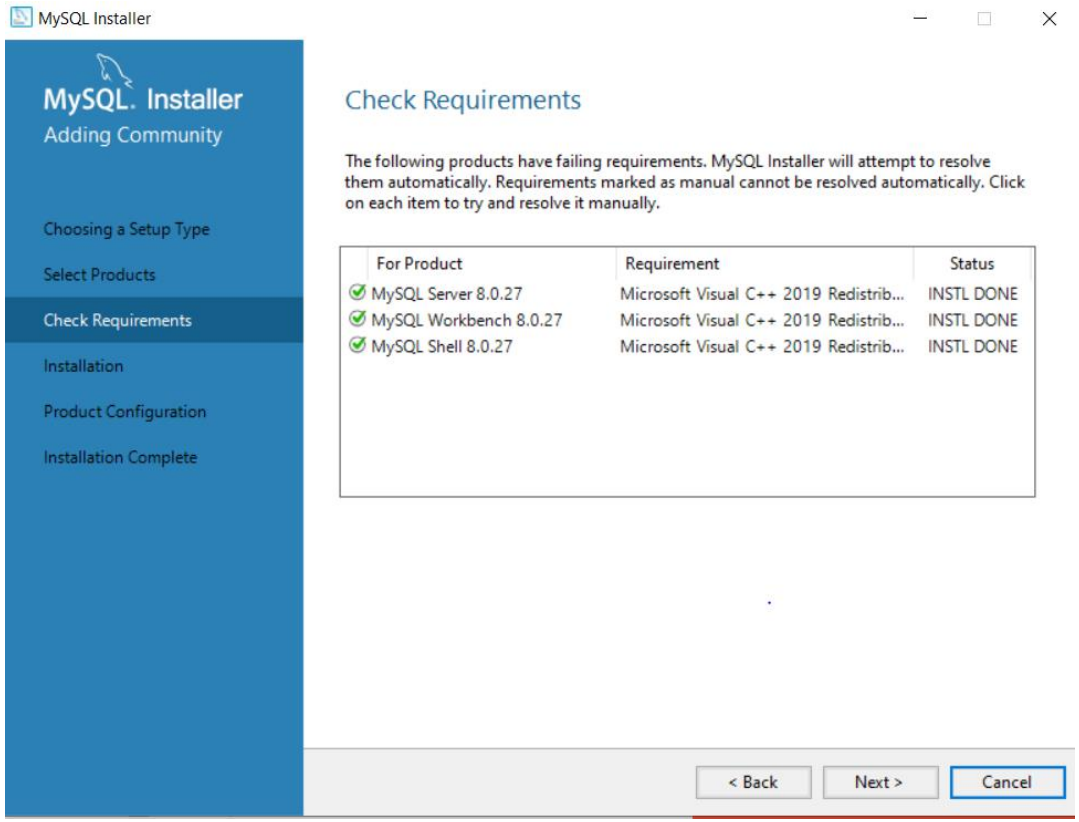
Double click on the downloaded msi file and Select Custom, press Next



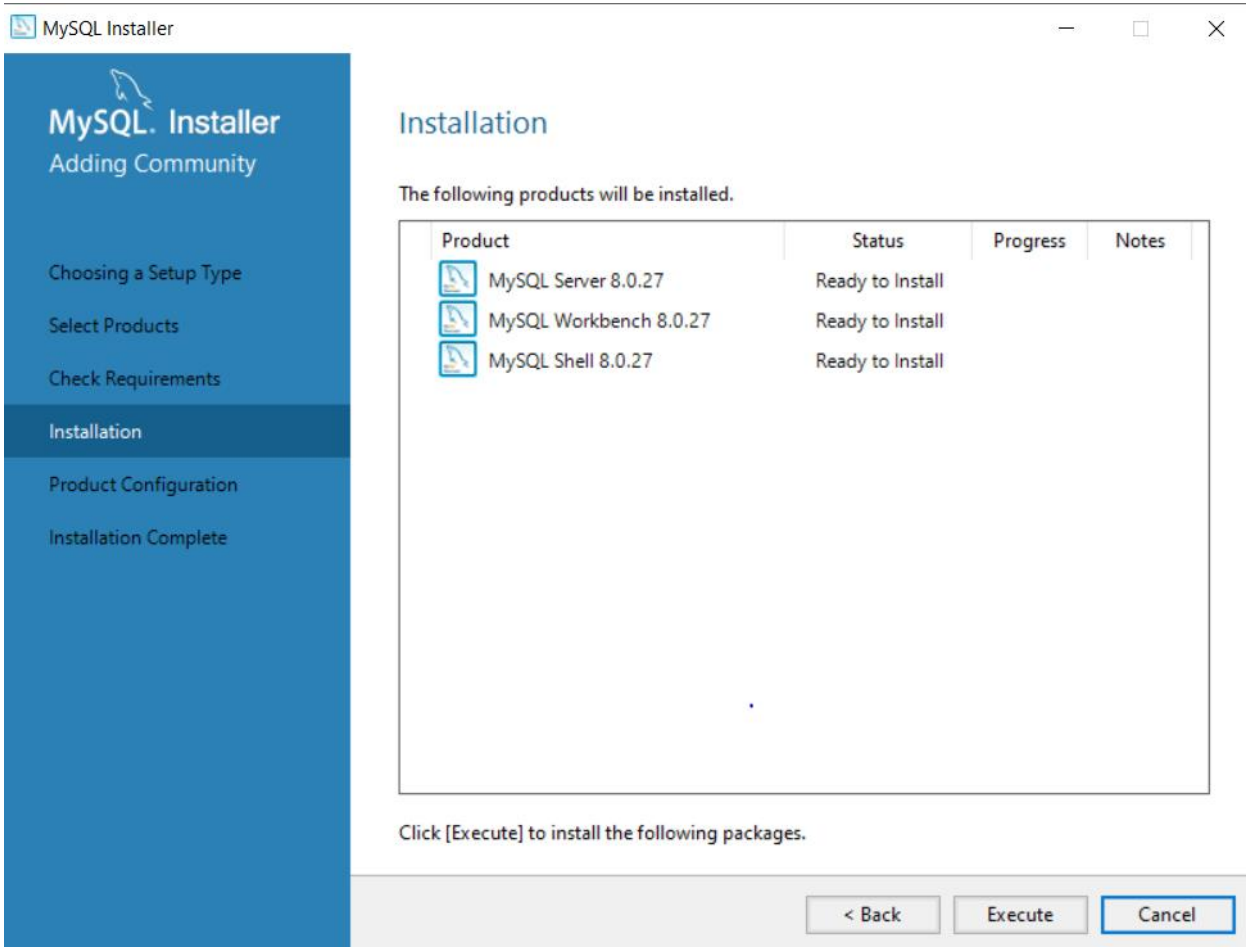
Select MySQL Server, MySQL Workbench, and MySQL Shell so that they appear in the right hand box. Press Next



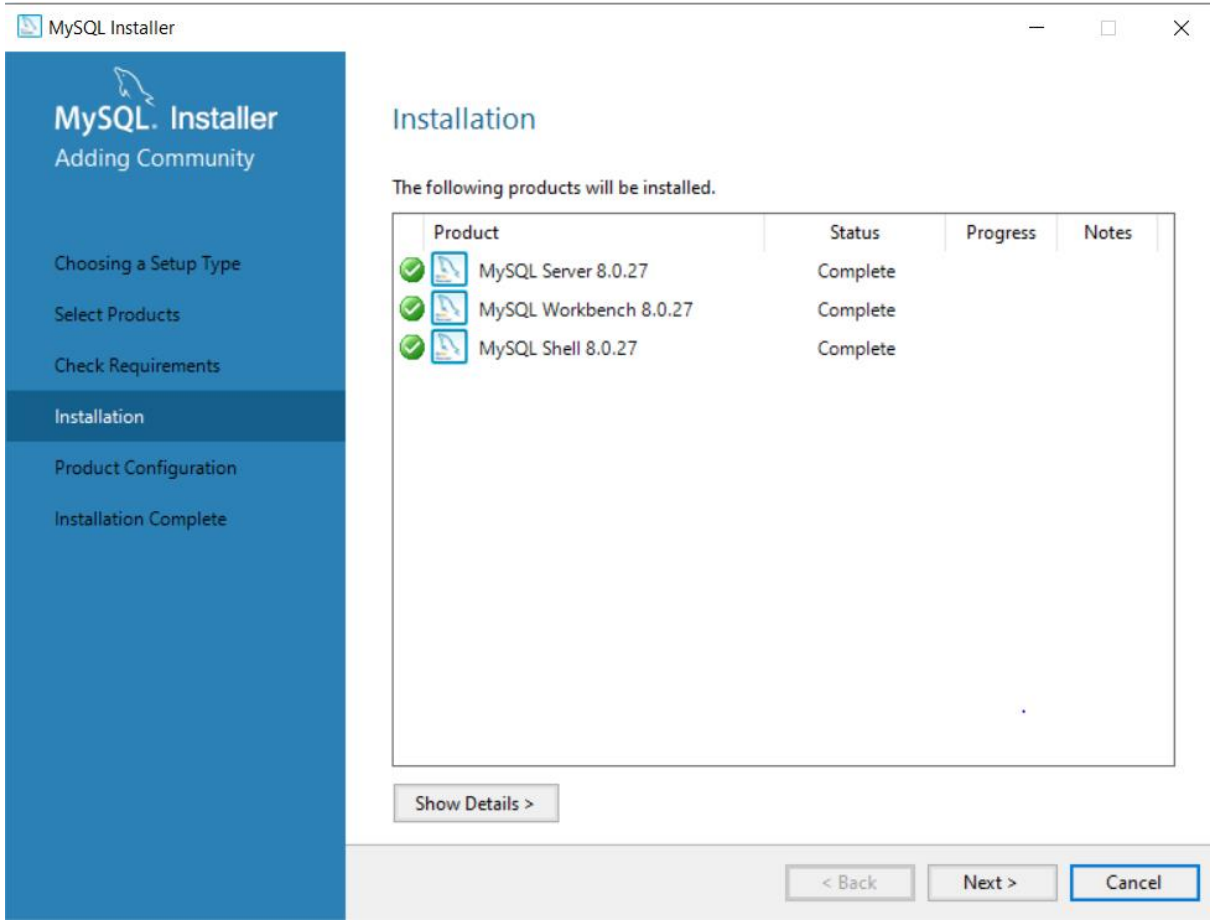
Press Next


MySQL Installer
MySQL. Installer
Adding Community
Choosing a Setup Type
Select Products
Check Requirements
Installation
Product Configuration
Installation Complete
Check Requirements
The following products have failing requirements. MySQL Installer will attempt to resolve them automatically. Requirements marked as manual cannot be resolved automatically. Click on each item to try and resolve it manually.
For Product
Requirement
Status
MySQL Server 8.0.27
Microsoft Visual C++ 2019 Redistrib...
INSTL DONE
MySQL Workbench 8.0.27
Microsoft Visual C++ 2019 Redistrib...
INSTL DONE
MySQL Shell 8.0.27
Microsoft Visual C++ 2019 Redistrib...
INSTL DONE
< Back
Next >
Cancel

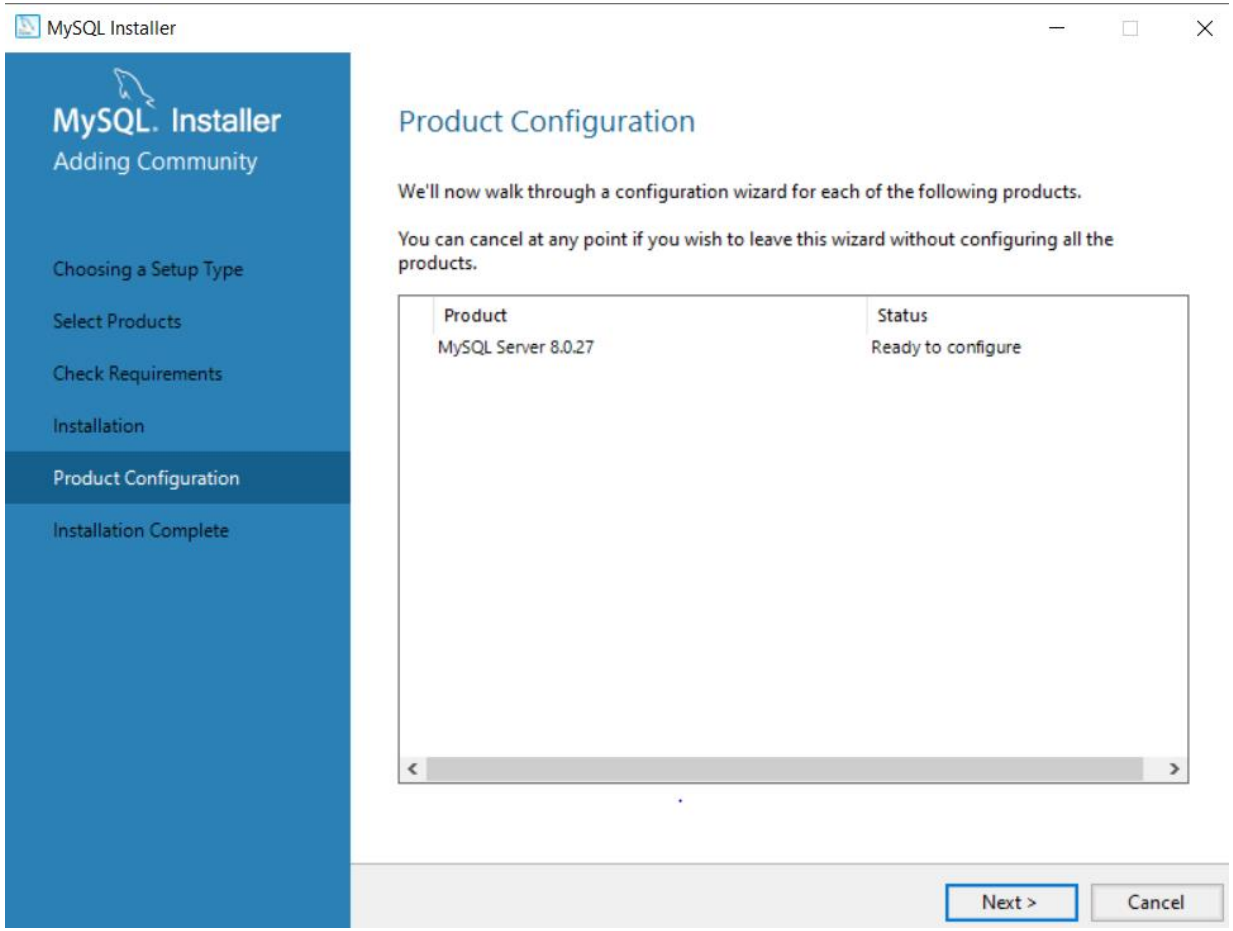
Press Execute



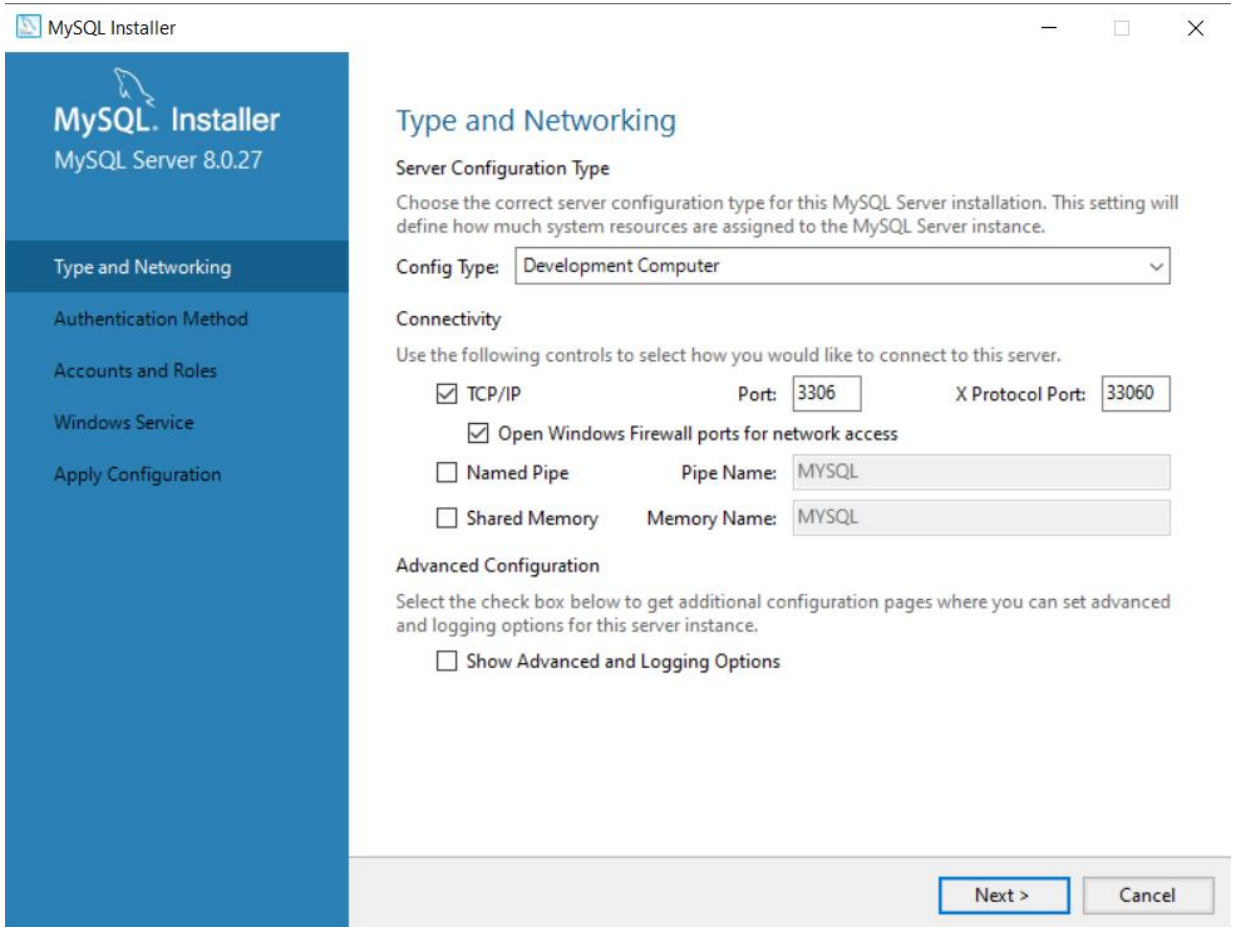
Here is how it looks when installed. Press Next



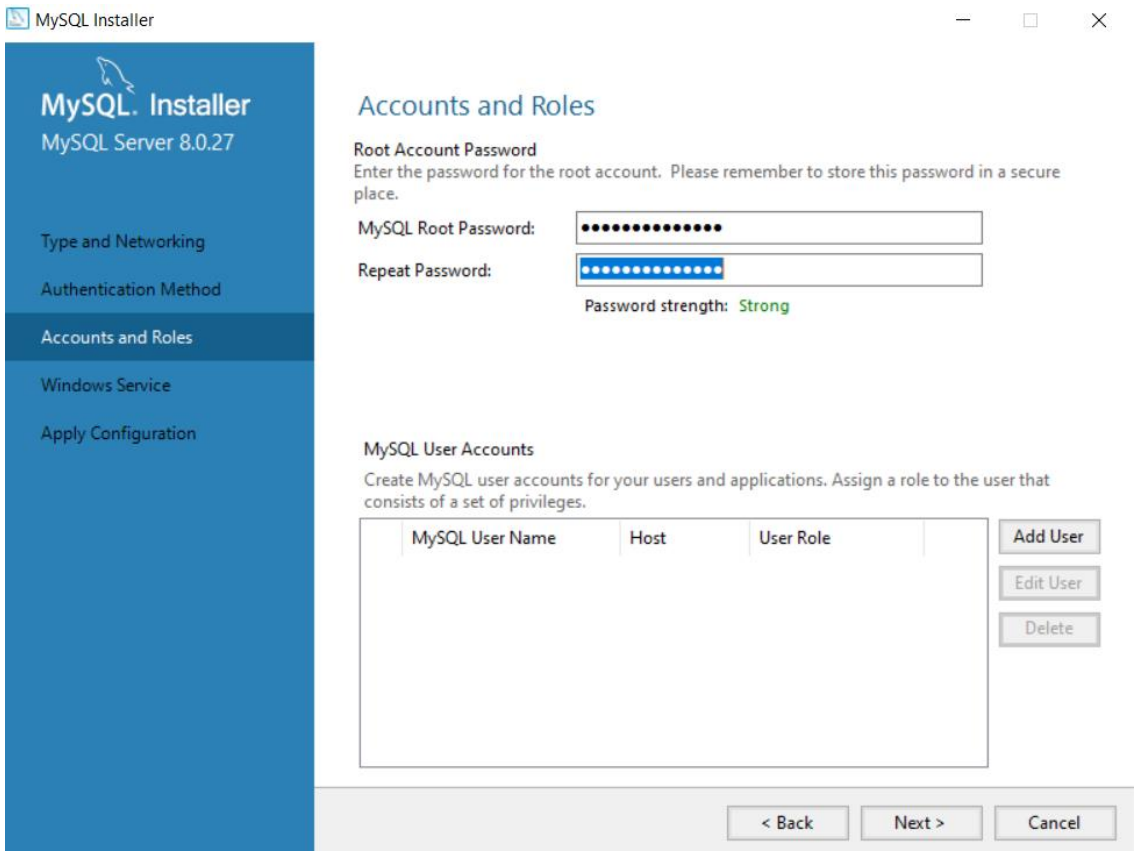
On the Product Configuration page, Press Next



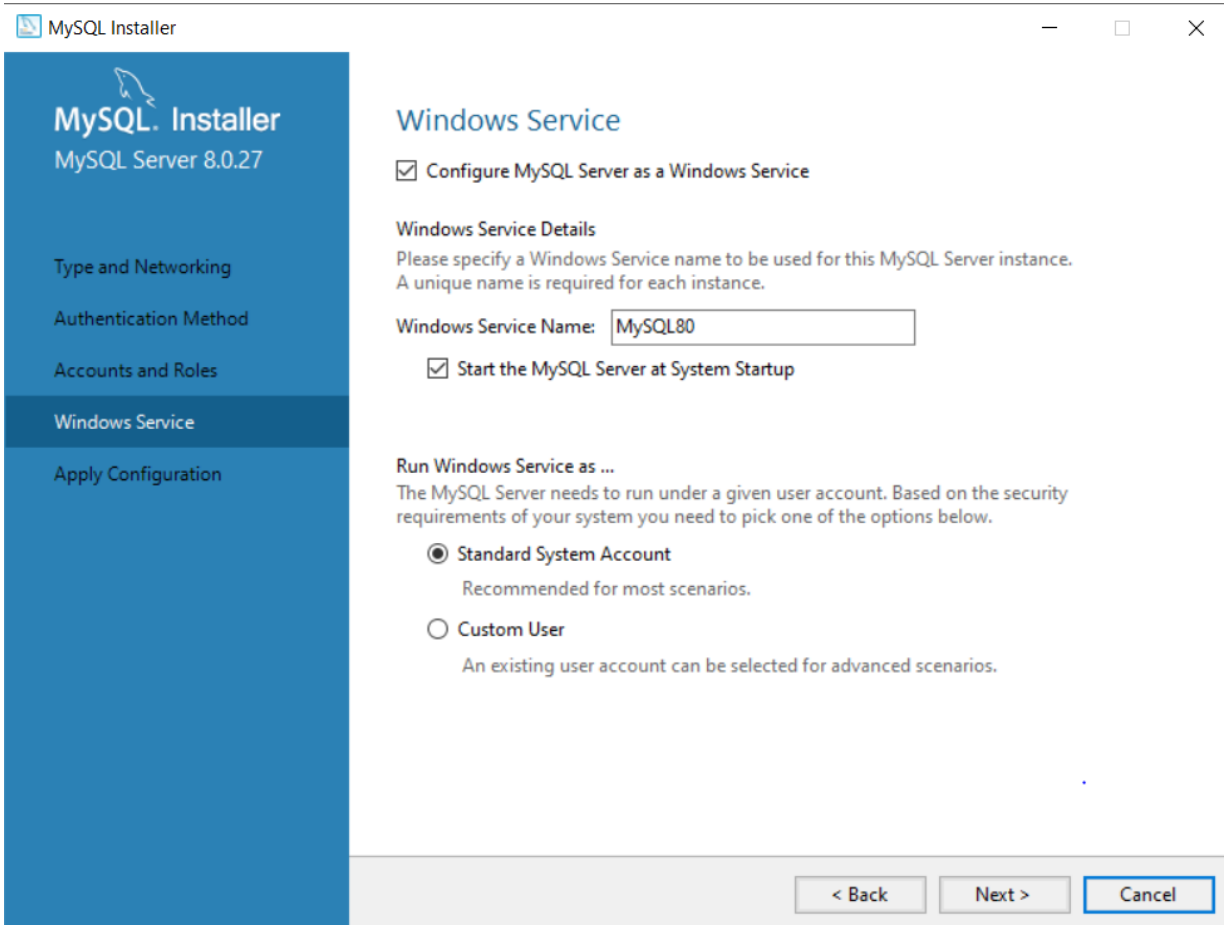
On the Type and Networking page, keep default, Press Next



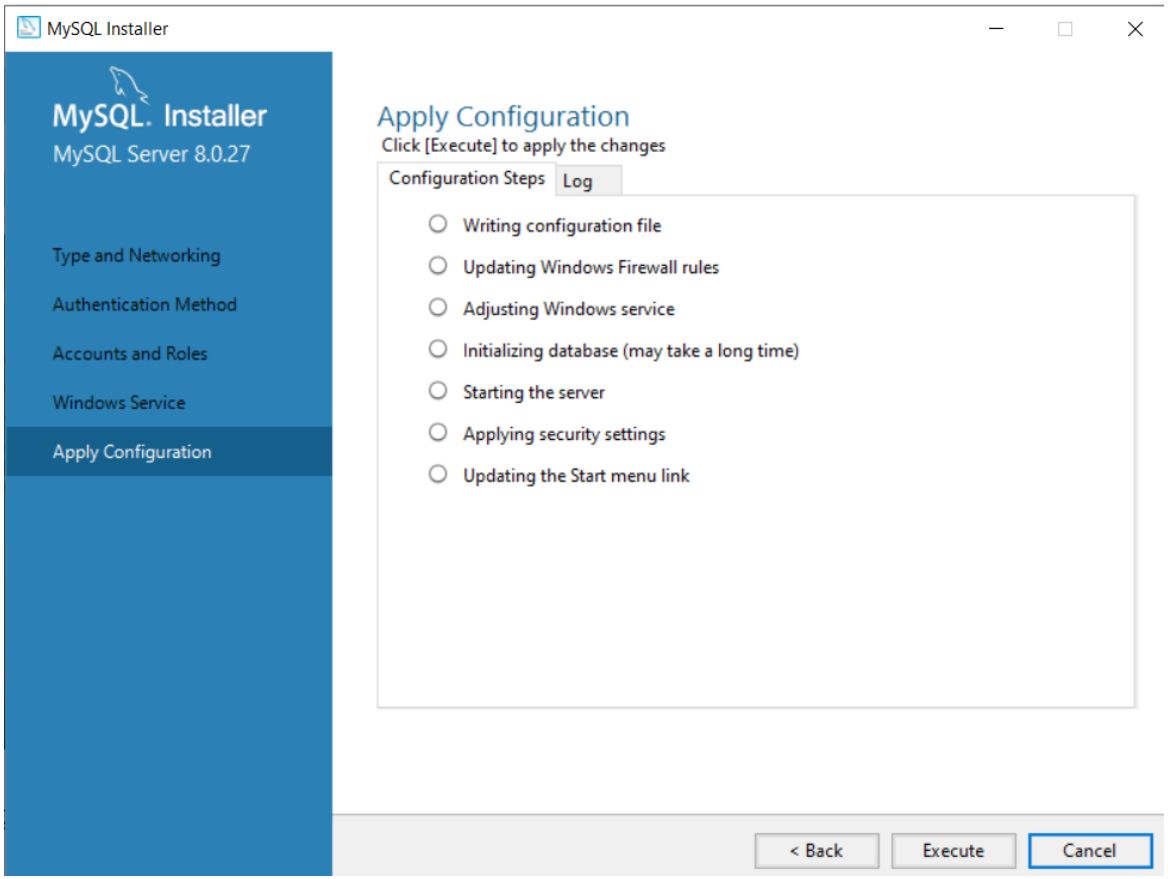
Select Strong Password for root user, Press Next



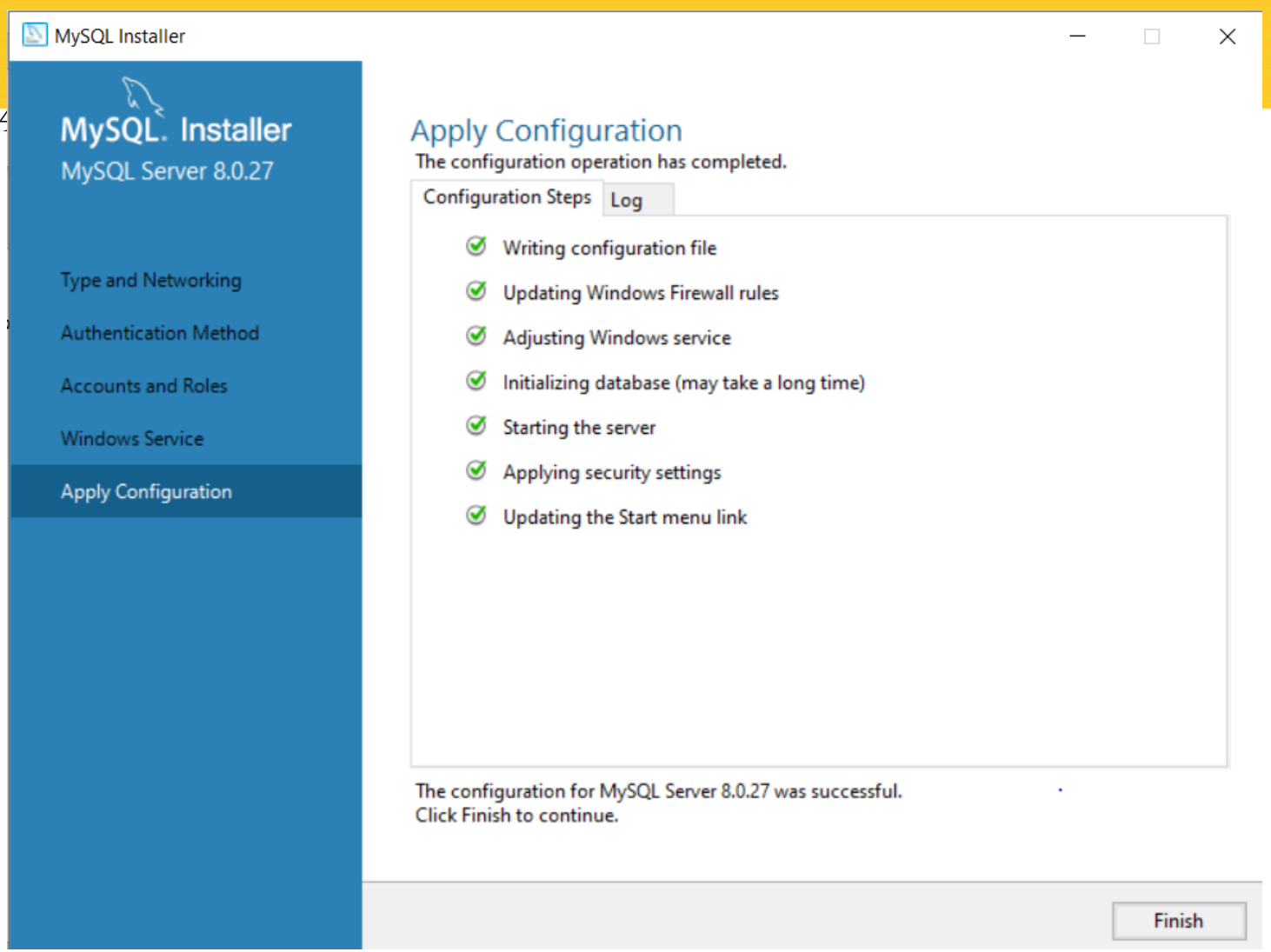
On Windows Service page, keep default, Press Next



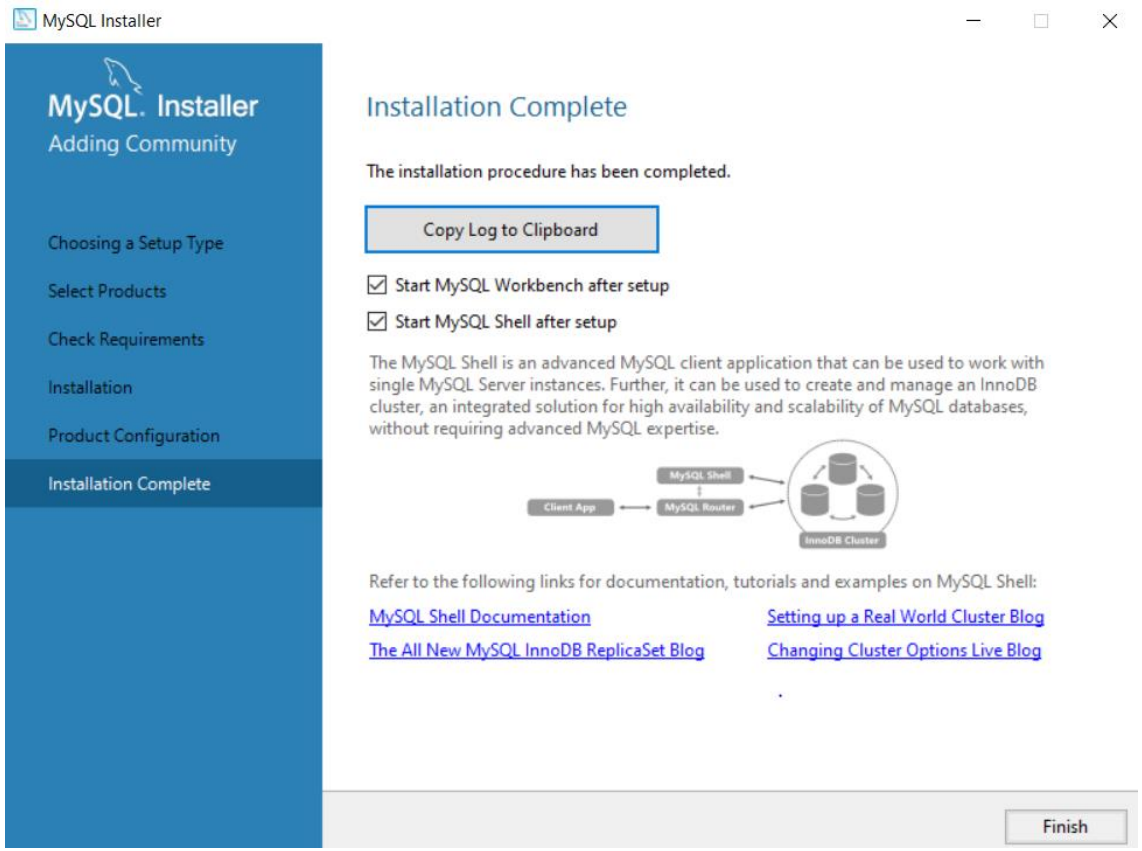
On Apply Configuration page, keep default, Press Execute



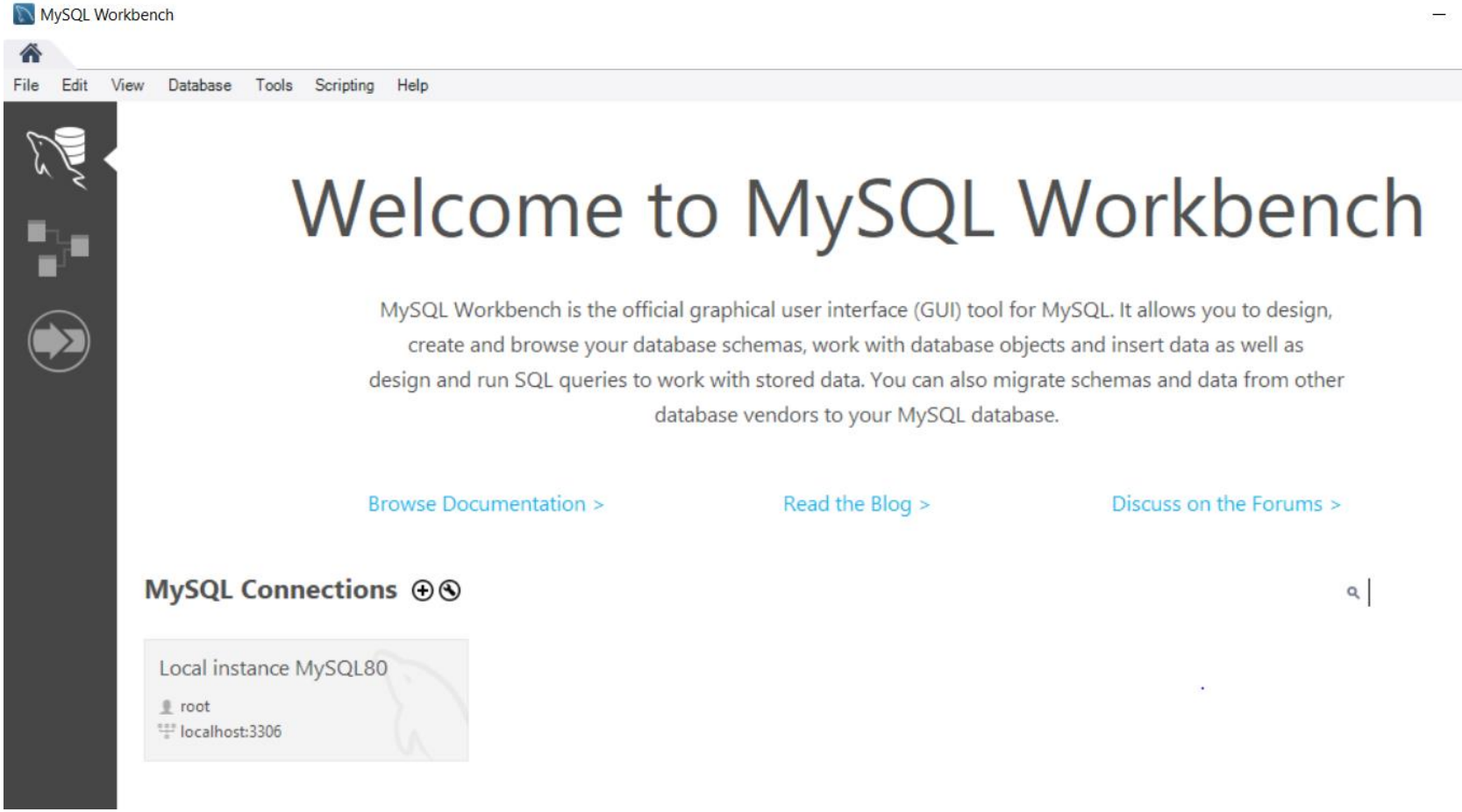
On A


MySQL Installer
MySQL. Installer
MySQL Server 8.0.27
Type and Networking
Authentication Method
Accounts and Roles
Windows Service
Apply Configuration
Apply Configuration
The configuration operation has completed.
Configuration Steps
Log
Writing configuration file
Updating Windows Firewall rules
Adjusting Windows service
Initializing database (may take a long time)
Starting the server
Applying security settings
Updating the Start menu link
The configuration for MySQL Server 8.0.27 was successful.
Click Finish to continue.
Finish

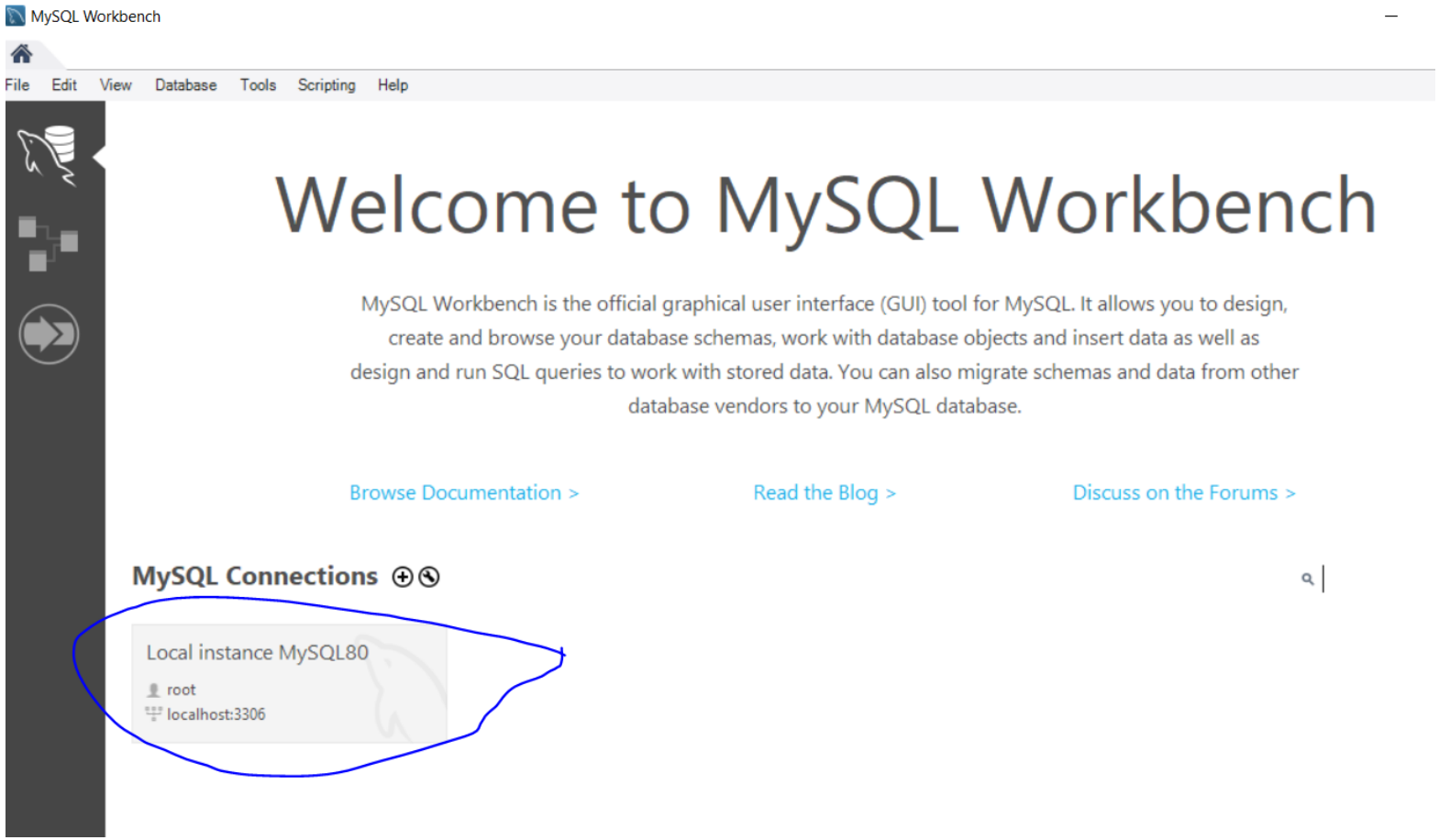
Press Next and then press Finish



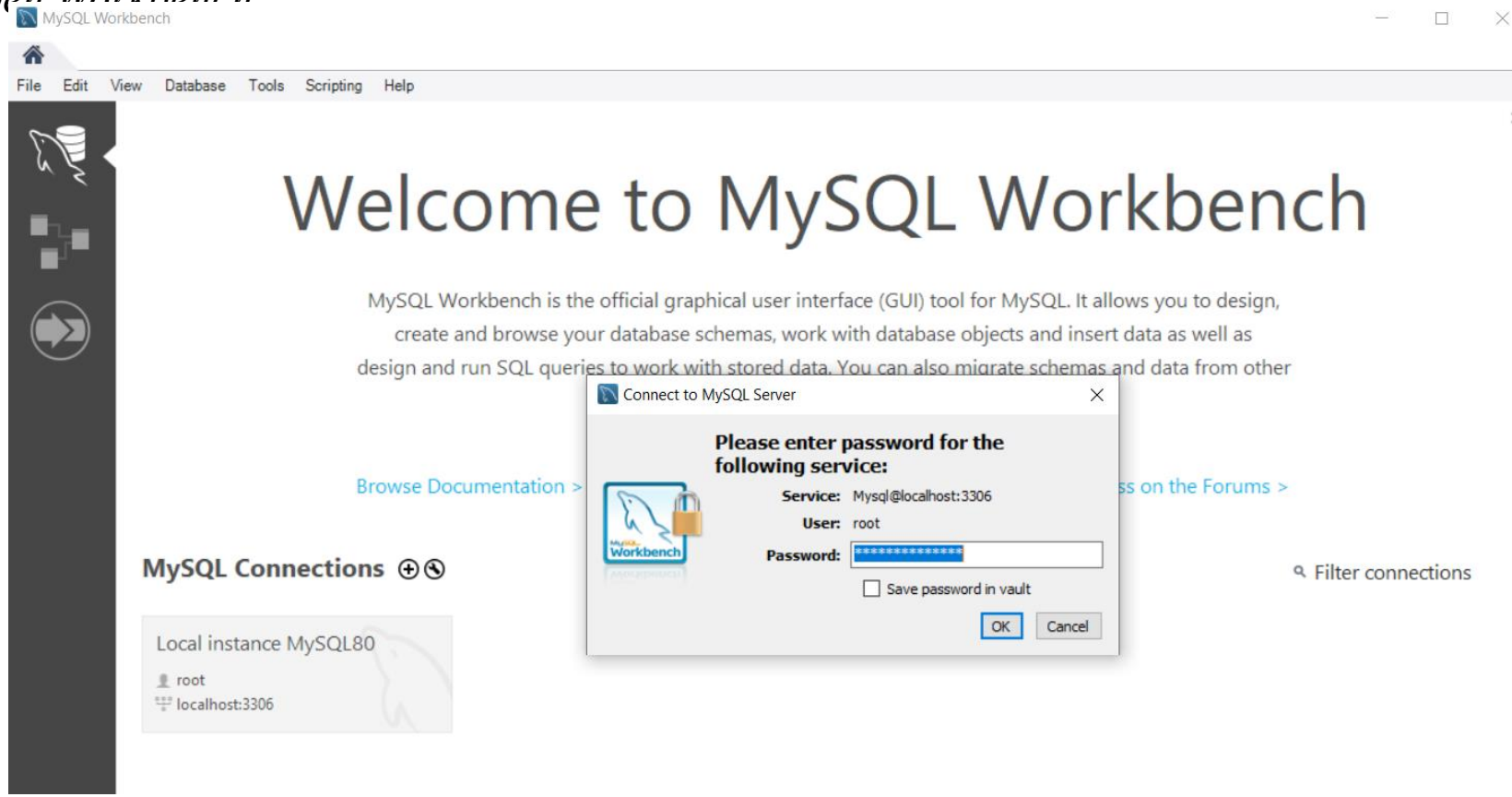
You will see console for MySQL Workbench opens



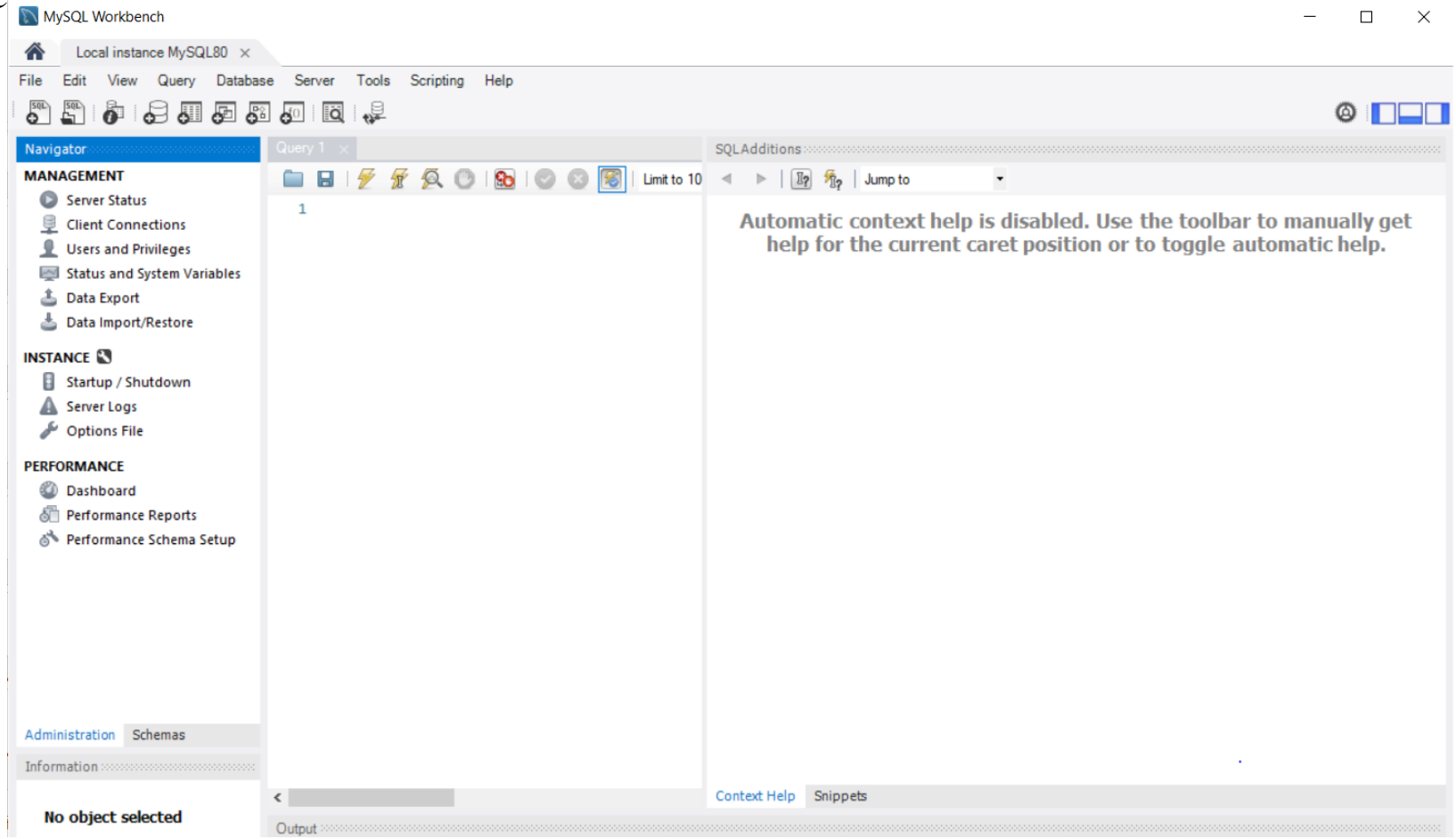
Click on Local instance MySQL8.0 to open Workbench.



Click on Local instance MySQL8.0 to login with root credentials and open Workbench



MySQL8.0 Workbench opens.



MySQL8.0 Workbench opens.