

# Lab 13. Spring Boot

Author: Yida Tao, Yao Zhao

This tutorial is aligned with our Sprint Boot lecture notes.

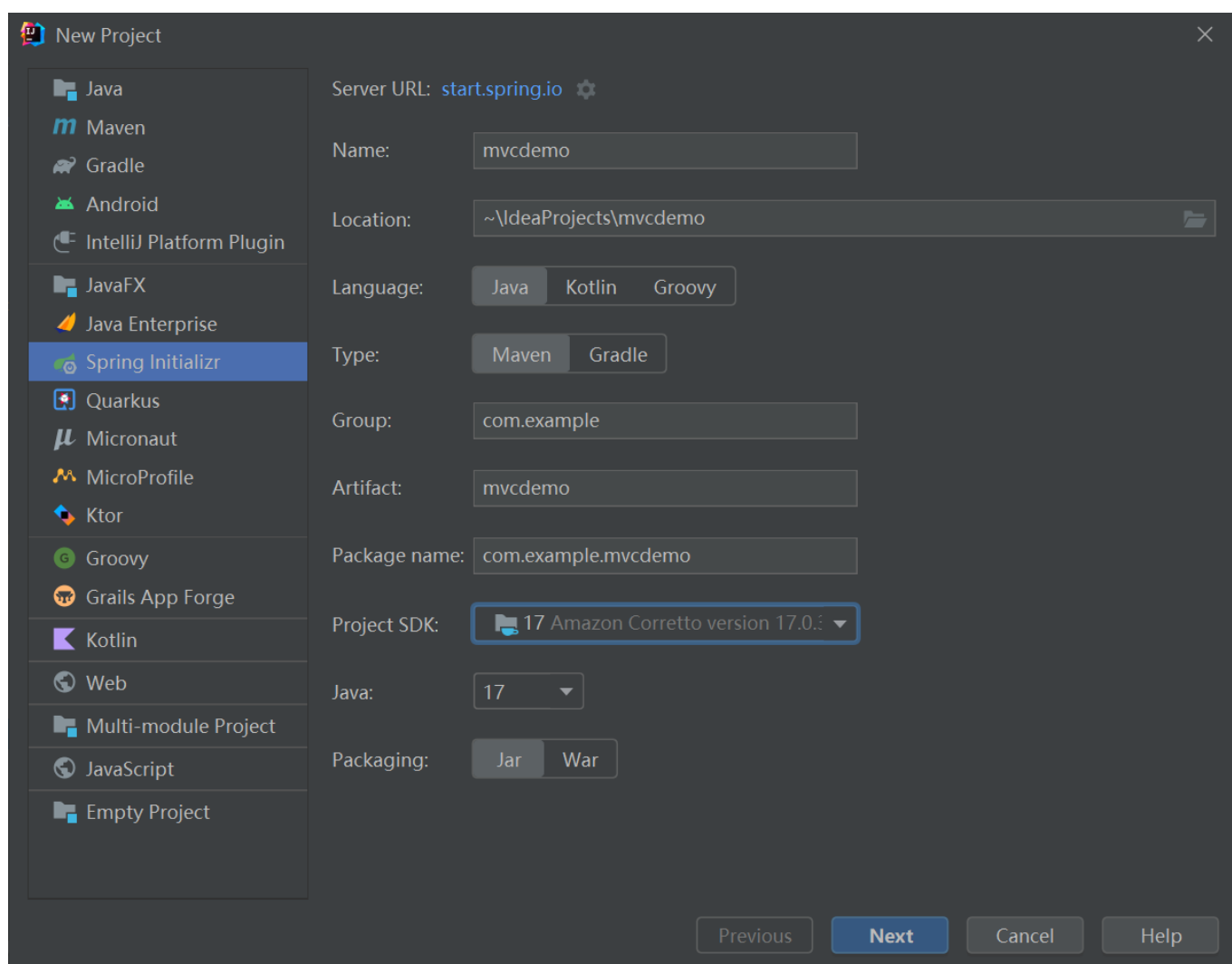
In this tutorial, we'll create a simple Spring Boot application, which could either be accessed by a browser or as a RESTful API.

## Preparation: Downloading IntelliJ Ultimate

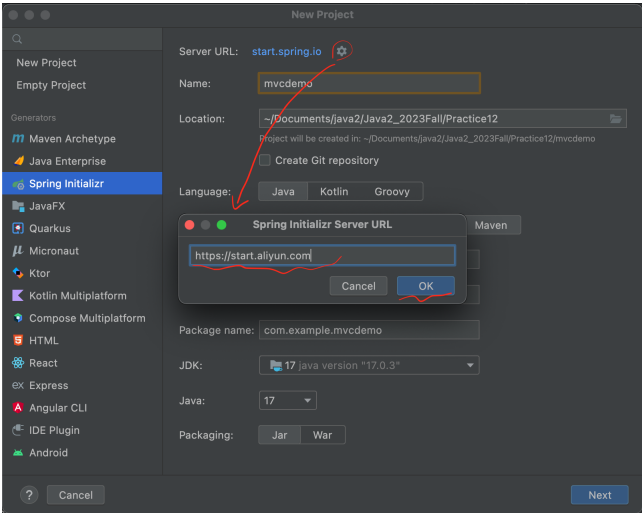
IntelliJ Ultimate makes developing Spring Boot applications easier. You may register a free educational license using your SUSTech email. See [here](#) for registration instructions.

## Part I: Creating a New Spring Boot Project

File -> New -> Project -> Spring Initializr, then specify project information like name and Java version and click "Next".

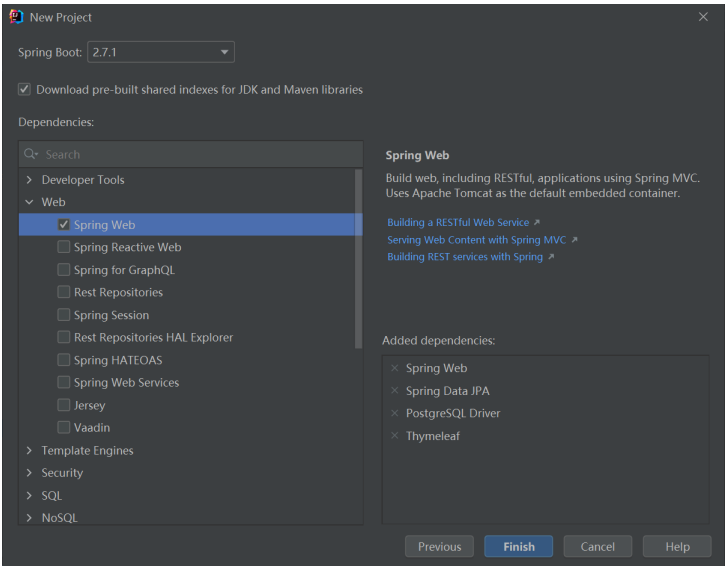


The default **Server URL** is `start.spring.io`. If you can't connect to this server, or the download speed is slow, or you are looking for spring boot version `2.7.x`, you can change the **Server URL** to `start.aliyun.com`.

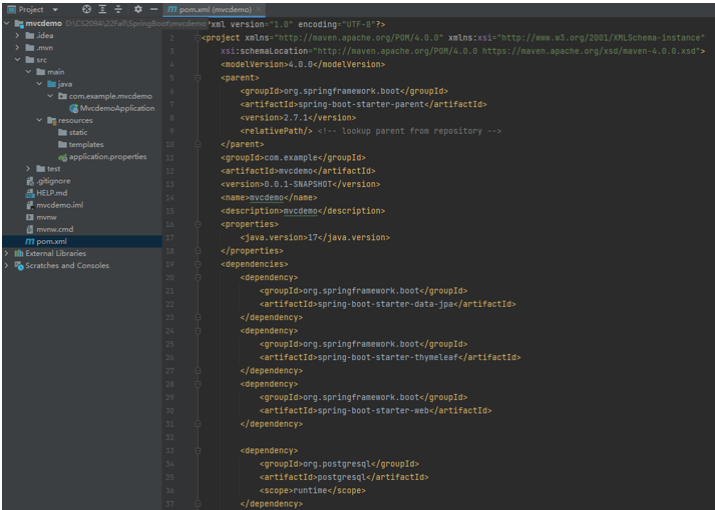


Then, we'll select necessary dependencies for this project.

In this tutorial, we'll use Spring Web (for MVC and REST service), Spring Data JPA (for data persistence), PostgreSQL Driver (for connecting to a PostgreSQL database), and Thymeleaf (for working with HTML).



Click "Finish", and you'll see a project structure as follows. Maven will automatically download all the dependencies, which may take a while.



Since we're going to use the PostgreSQL database, we need to install this database as instructed [here](#). During installation, you could set the username and password.

After installing PostgreSQL, you could open **SQL Shell (psql)** (windows). Enter all the necessary information such as the server, database, port, username, and password. To accept the default, you can simply press **Enter**. Note that you should provide the password that you entered during installing the PostgreSQL.

#### SQL Shell (psql)

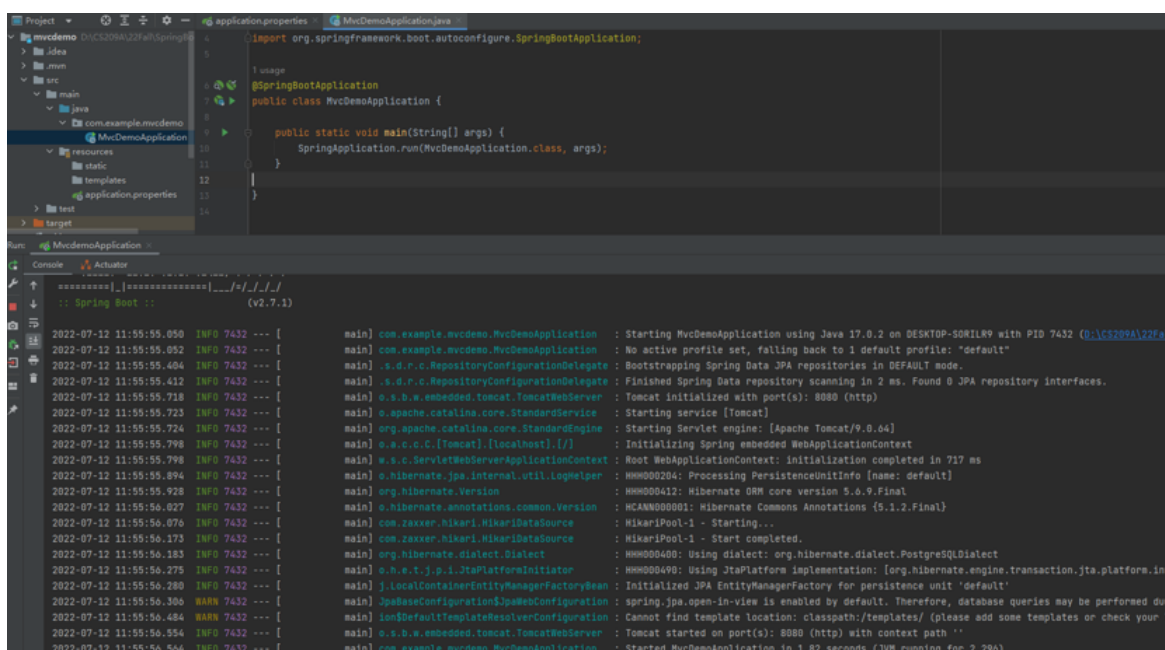
```
Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]: postgres
用户 postgres 的口令:
psql (14.4)
输入 "help" 来获取帮助信息.
```

In SQL shell, type **CREATE DATABASE cs209a;** to create a database called "cs209a". Finally, add the following properties to **src/main/resources/application.properties** to configure the database to our Spring Boot project.

```
spring.datasource.url=jdbc:postgresql://localhost:5432/cs209a
spring.datasource.username=postgres
spring.datasource.password=123456
spring.jpa.hibernate.ddl-auto=create-drop
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.properties.hibernate.format_sql=true

server.error.include-message=always
```

Now, run **MvcDemoApplication.java**, and if you see the following content in your console, congratulations! You've successfully executed your first Spring Boot application.

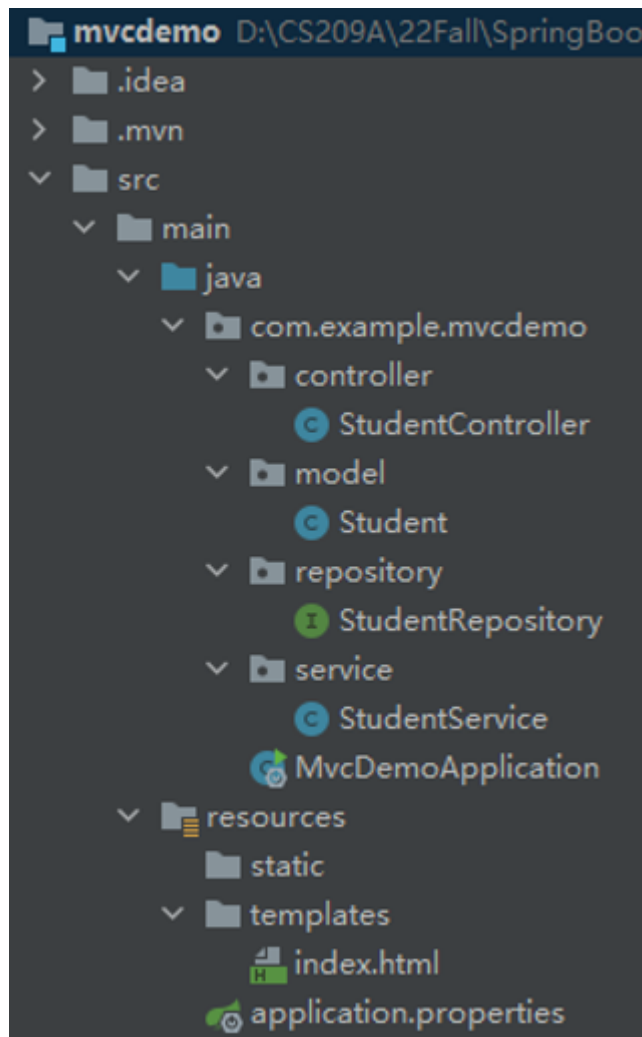


```
Project: mvcdemo
src
  resources
    application.properties
  test
  target
  mvcdemoApplication.java

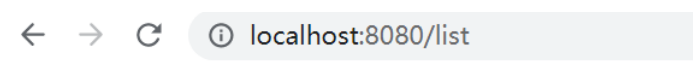
Console:
2022-07-12 11:55:55.050 INFO 7432 --- [main] com.example.mvcdemo.MvcDemoApplication : Starting MvcDemoApplication using Java 17.0.2 on DESKTOP-SORIL89 with PID 7432 (0\cs209a\22f4)
2022-07-12 11:55:55.052 INFO 7432 --- [main] com.example.mvcdemo.MvcDemoApplication : No active profile set, falling back to 1 default profile: "default"
2022-07-12 11:55:55.404 INFO 7432 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-07-12 11:55:55.412 INFO 7432 --- [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 2 ms. Found 0 JPA repository interfaces.
2022-07-12 11:55:55.718 INFO 7432 --- [main] s.s.b.e.embedded.TomcatTomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2022-07-12 11:55:55.723 INFO 7432 --- [main] s.apache.catalina.core.StandardService : Starting service [Tomcat]
2022-07-12 11:55:55.724 INFO 7432 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.64]
2022-07-12 11:55:55.798 INFO 7432 --- [main] s.s.c.o.C [Tomcat].[localhost]. [/] : Initializing Spring embedded WebApplicationContext
2022-07-12 11:55:55.798 INFO 7432 --- [main] s.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 717 ms
2022-07-12 11:55:55.894 INFO 7432 --- [main] s.hibernate.jpa.internal.util.LogHelper : HHN000204: Processing PersistenceUnitInfo [name: default]
2022-07-12 11:55:55.928 INFO 7432 --- [main] org.hibernate.Version : HHN0000412: Hibernate ORM core version 5.6.9.Final
2022-07-12 11:55:56.027 INFO 7432 --- [main] s.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-07-12 11:55:56.076 INFO 7432 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2022-07-12 11:55:56.173 INFO 7432 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2022-07-12 11:55:56.183 INFO 7432 --- [main] org.hibernate.dialect.Dialect : HHN000490: Using dialect: org.hibernate.dialect.PostgreSQLDialect
2022-07-12 11:55:56.275 INFO 7432 --- [main] s.h.e.t.j.p.i.JtaPlatformInitiator : HHN000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.int
2022-07-12 11:55:56.280 INFO 7432 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-07-12 11:55:56.306 WARN 7432 --- [main] jpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed dur
2022-07-12 11:55:56.484 WARN 7432 --- [main] lonDefaultTemplateResolverConfiguration : Cannot find template location: classpath:/templates/ (please add some templates or check your T
2022-07-12 11:55:56.554 INFO 7432 --- [main] s.s.b.e.embedded.TomcatTomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2022-07-12 11:55:56.564 INFO 7432 --- [main] com.example.mvcdemo.MvcDemoApplication : Started MvcDemoApplication in 1.82 seconds (JVM running for 2.29s)
```

## Part II: Creating a Simple MVC Application

Let's create a simple MVC application with the following structure, which has also been explained in our lectures. You could download [mvcdemo.zip](#) for Spring Boot 2.7.x or [mvcdemo2.zip](#) for Spring Boot 3 from Blackboard to get the source code.



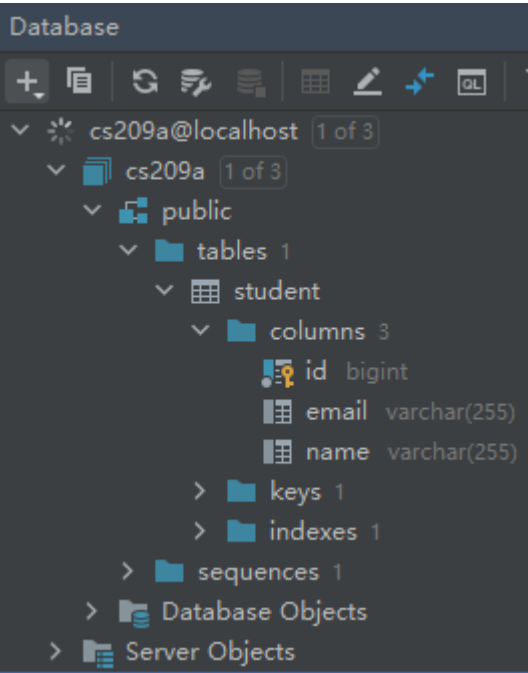
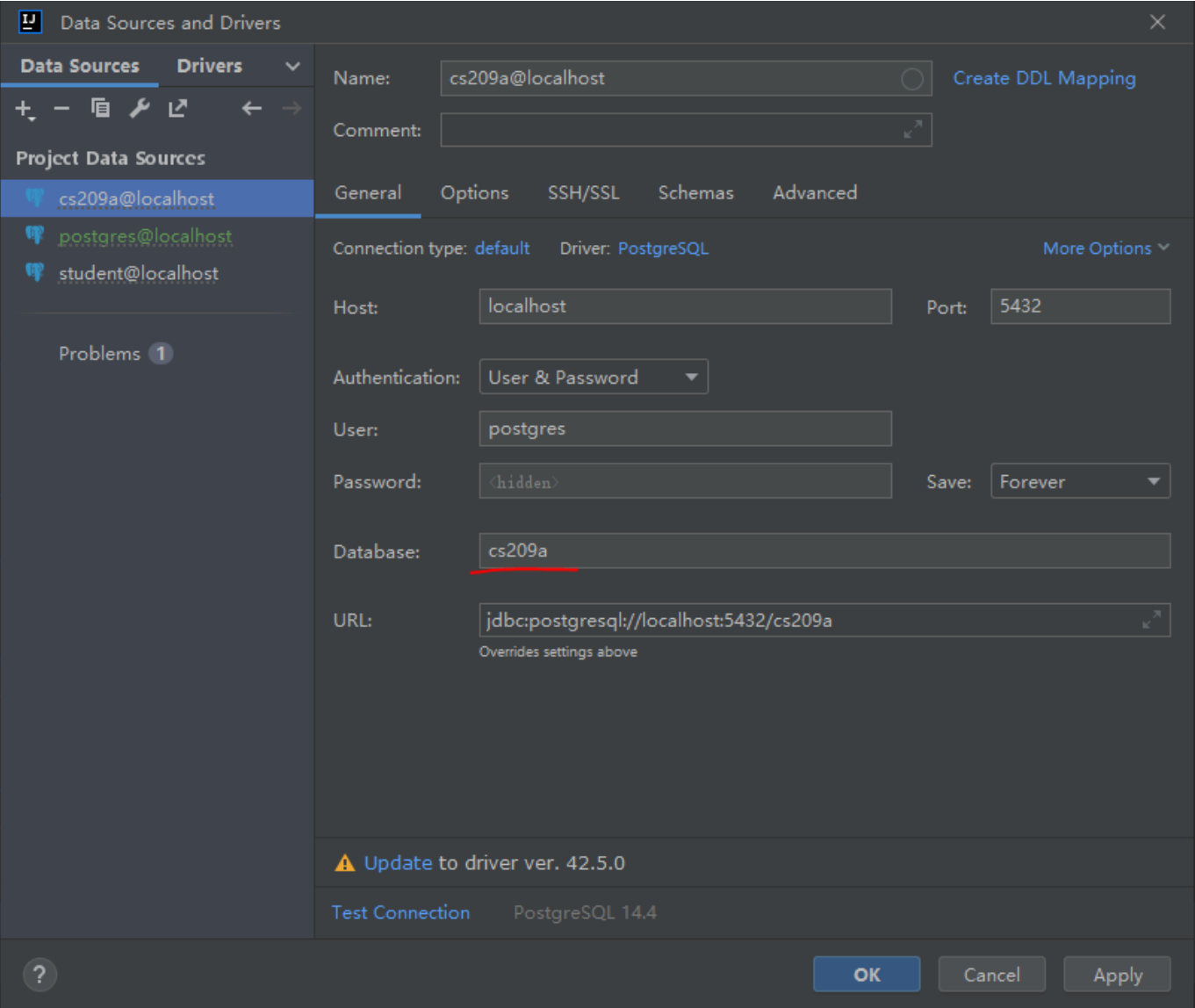
Then, start Spring Boot by executing `MvcDemoApplication.java` in `mvcdemo.zip` or `Mvcdemo2Application.java` in `mvcdemo2.zip`. Now open your browser and type `localhost:8080/list`. If you've done everything correctly, you should see the following table:



# Student List

- 1 Mary mary@gmail.com
- 2 Alex alex@gmail.com
- 3 Dean dean@yahoo.com

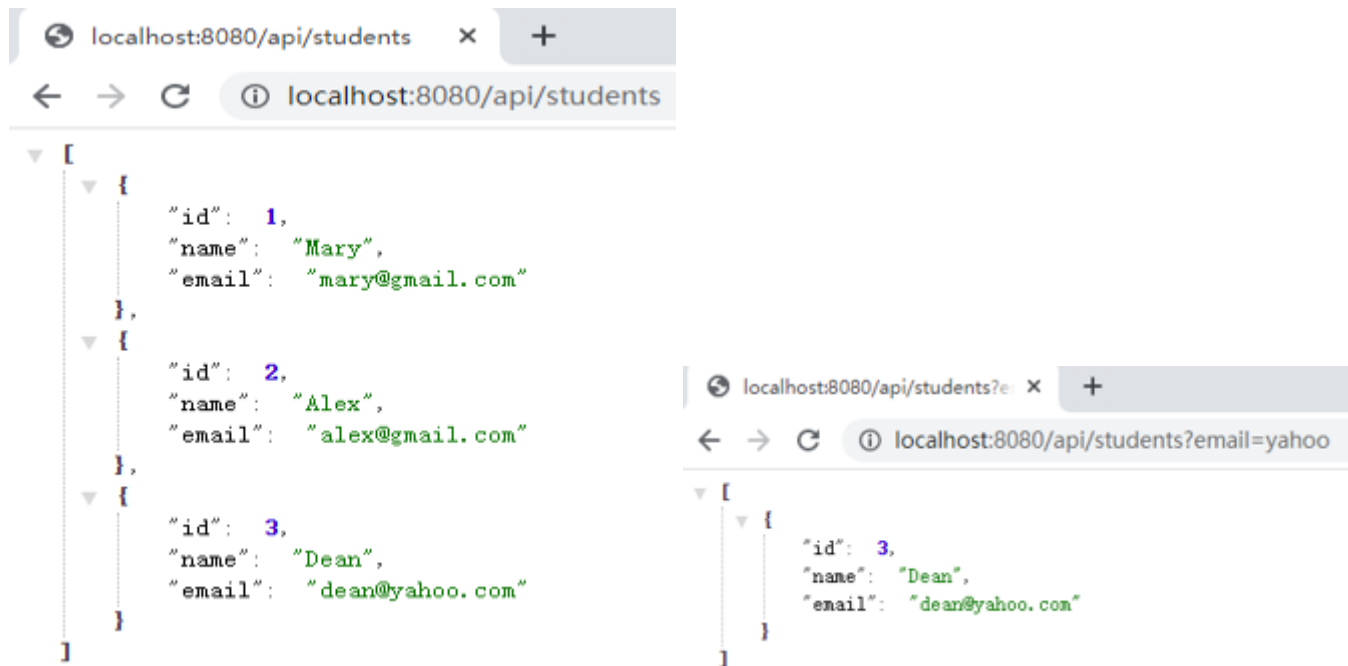
Using JPA features as well as the proper annotations such as `@Entity` and `@Table`, Spring Boot automatically creates and updates a `Student` table in `cs209a` database. You might also view the table inside of IntelliJ IDEA by clicking `Database -> + -> Data source -> PostgreSQL` and type in `cs209a` to connect to the database.



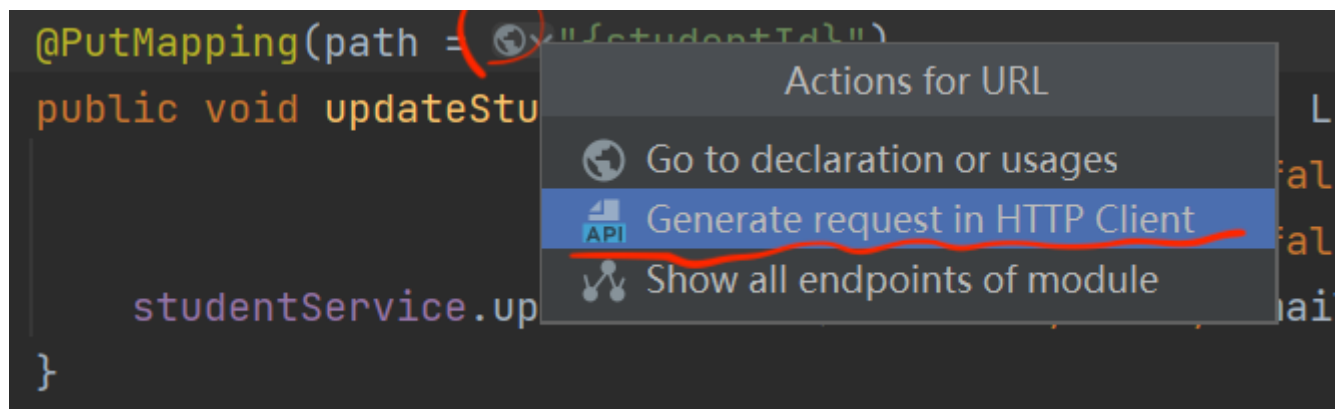
| WHERE |    |                | ORDER BY |  |
|-------|----|----------------|----------|--|
|       | id | email          | name     |  |
| 1     | 1  | mary@gmail.com | Mary     |  |
| 2     | 2  | alex@gmail.com | Alex     |  |
| 3     | 3  | dean@yahoo.com | Dean     |  |

## Part III: Creating a Simple RESTful Service

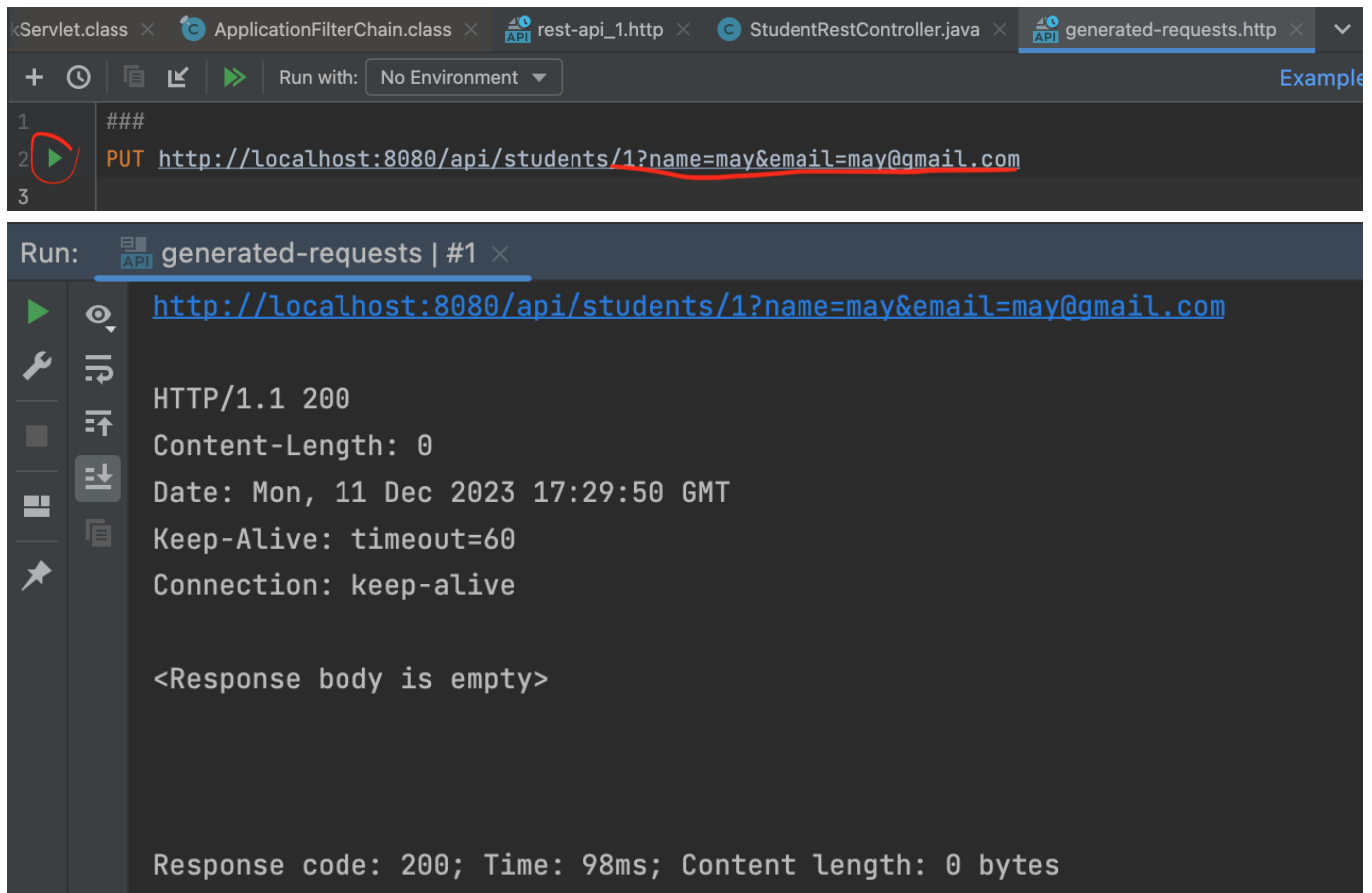
Let's also create a `StudentRestController.java` based on our lecture notes and put it in the `controller` package. Specifically, by implementing the `getStudentsByEmail` method, which maps to `/api/students`, you'll get the following `json` responses back.



By implementing the `updateStudent` method, you'll be able to execute a `PUT` request like `PUT http://localhost:8080/api/students/1?name=may&email=may@gmail.com` to update the information of a certain student. You could open the HTTP client (as follows) to execute the REST request.



Click the green button to execute the `PUT` method.



The screenshot shows an IDE with several tabs: `<Servlet.class`, `ApplicationFilterChain.class`, `rest-api_1.http`, `StudentRestController.java`, and `generated-requests.http`. The `generated-requests.http` tab is active, showing a REST client request:

```
1 ###
2 PUT http://localhost:8080/api/students/1?name=may&email=may@gmail.com
3
```

Below the code editor, the 'Run' window shows the response for the first request:

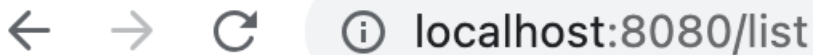
```
Run: generated-requests | #1 x
http://localhost:8080/api/students/1?name=may&email=may@gmail.com

HTTP/1.1 200
Content-Length: 0
Date: Mon, 11 Dec 2023 17:29:50 GMT
Keep-Alive: timeout=60
Connection: keep-alive

<Response body is empty>

Response code: 200; Time: 98ms; Content length: 0 bytes
```

Refresh the page in your browser to check the updated table.



← → ↻ ⓘ localhost:8080/list

# Student List

2 Alex alex@gmail.com

3 Dean dean@yahoo.com

1 may may@gmail.com

## References

<https://amigoscode.com/p/spring-boot>