API TRAINING EXERCISE

Spring Ver. 1.0

Apr 2020

generali.com

1 Version History

Version	Date	Release Note
V0.1	10-Mar-2020	Initial Draft
V1.0	2-Apr-2020	1st Release

INDEX

2 Exercise 1 – Form Submission

The exercise will demostrate how to create your first Spring Application

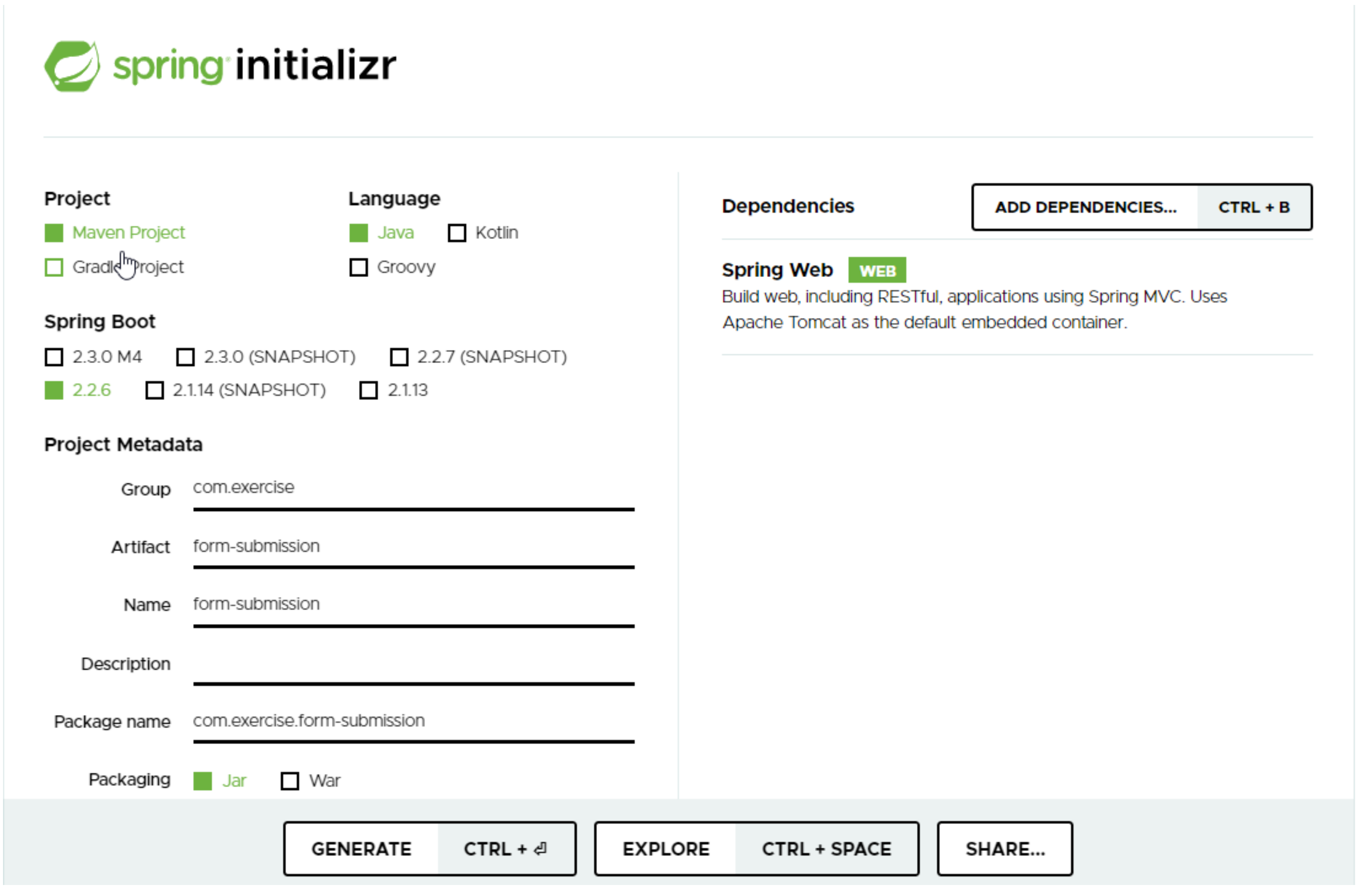
Prerequisite

- Java installed (e.g. JDK 11)
- Maven installed (e.g. version 3.6.3)
- Spring Tool Suite (STS) (e.g. 4.5.1)

2.1 FORM SUBMISSION EXERCISE

2.1.1 Generate sample pom.xml

```
Group:           com.exercise
Artifact:        form-submission
Dependencies:    Spring Web
```



spring initializr

Project
[✓ Maven Project] Gradle Project

Language
[✓ Java] Kotlin Groovy

Spring Boot
2.3.0 M4 2.3.0 (SNAPSHOT) 2.2.7 (SNAPSHOT) [✓ 2.2.6] 2.1.14 (SNAPSHOT) 2.1.13

Project Metadata

Group	com.exercise
Artifact	form-submission
Name	form-submission
Description	
Package name	com.exercise.form-submission
Packaging	[✓ Jar] War

Dependencies

Spring Web WEB
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

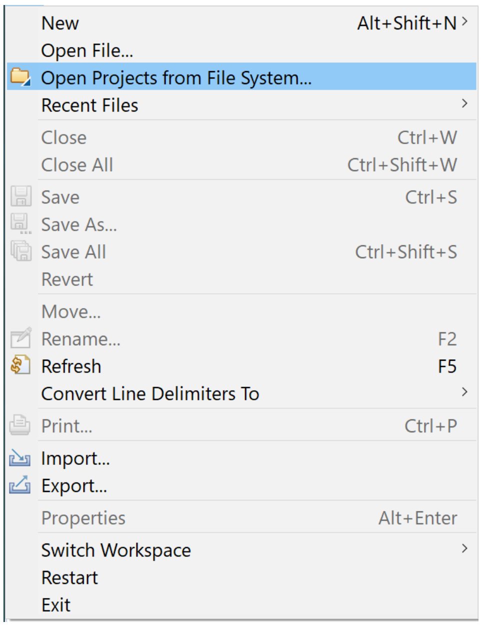
Or you may use this one:

form-submission-em
pty.zip

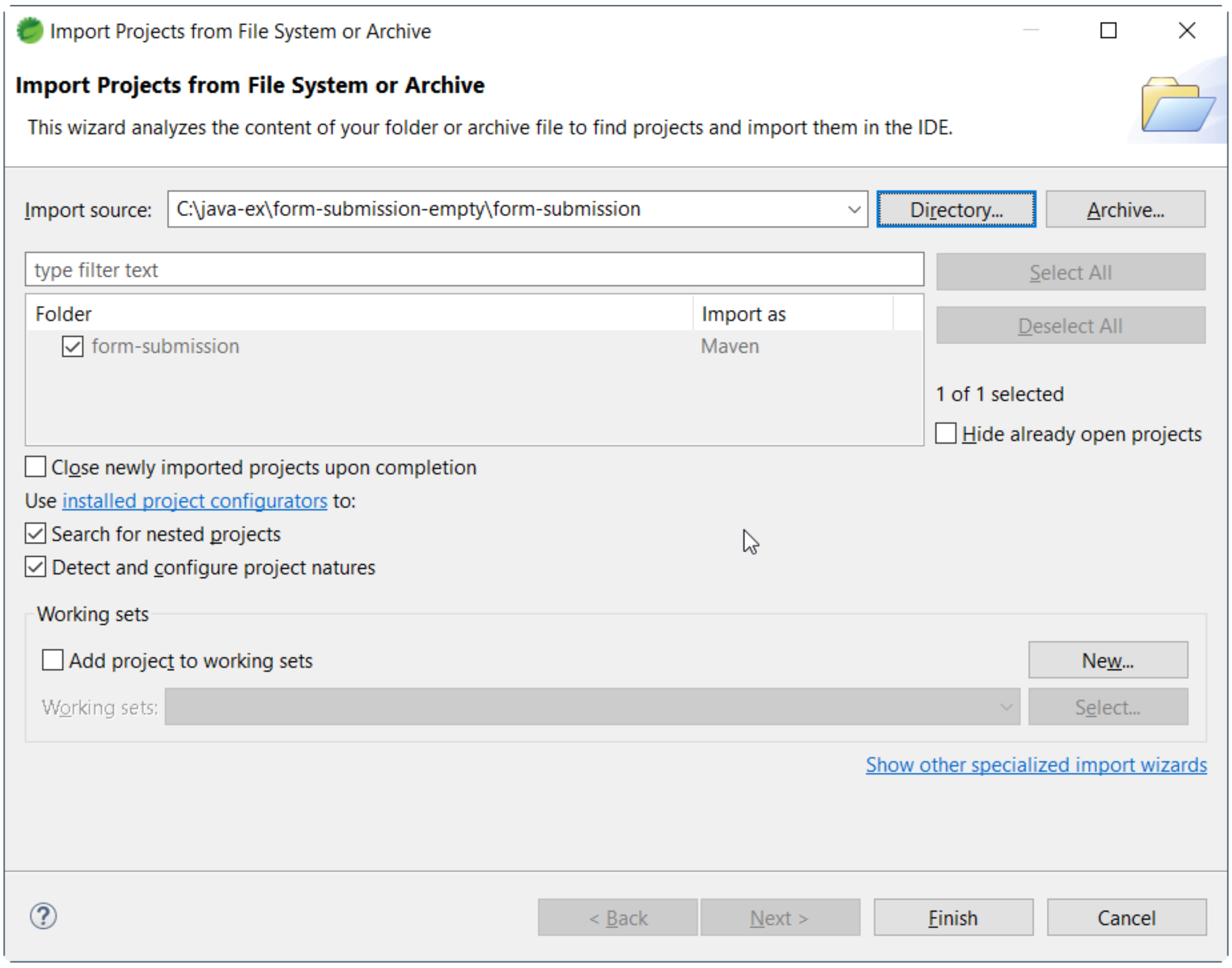
2. Click “Generate” and extract the zipped file to your local directory, e.g. C:\java-ex:

```
C:.
└───form-submission
    ├───.mvn
    │   └───wrapper
    └───src
        ├───main
        │   ├───java
        │   │   └───com
        │   │       └───exercise
        │   │           └───formsubmission
        │   └───resources
        │       ├───static
        │       └───templates
        └───test
            └───java
                └───com
                    └───exercise
                        └───formsubmission
```

3. Start up STS, click “File” and select “Open Projects from File System…”:



4. Type or search your project folder “form-submission”:



Import Projects from File System or Archive

Import Projects from File System or Archive

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

Import source: C:\java-ex\form-submission-empty\form-submission

Folder	Import as
☑ form-submission	Maven

1 of 1 selected

☐ Hide already open projects

☐ Close newly imported projects upon completion

Use installed project configurators to:

☑ Search for nested projects

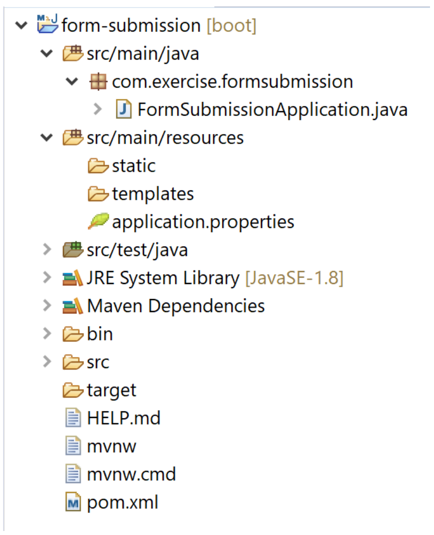
☑ Detect and configure project natures

Working sets

☐ Add project to working sets

Working sets:

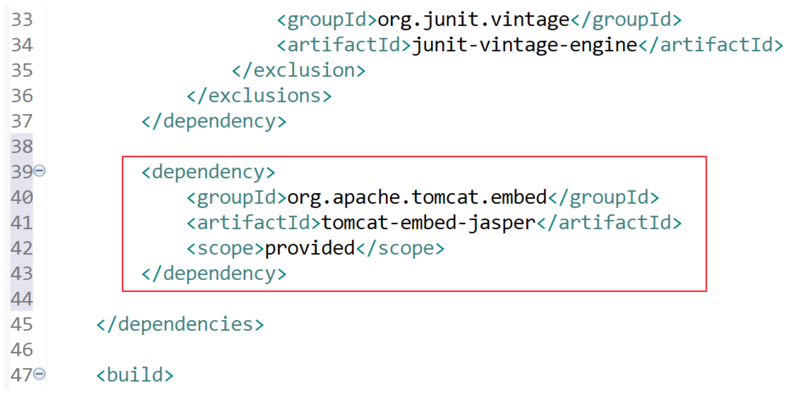
5. The project structure should be shown:



```
form-submission [boot]
  src/main/java
    com.exercise.formsubmission
      FormSubmissionApplication.java
  src/main/resources
    static
    templates
    application.properties
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  bin
  src
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

6. In order to support JSP, we need to add dependency to the project. Now open the file “pom.xml”, and insert the following code to the <dependencies> tag.

```
              <dependency>
                     <groupId>org.apache.tomcat.embed</groupId>
                     <artifactId>tomcat-embed-jasper</artifactId>
                     <scope>provided</scope>
              </dependency>
```



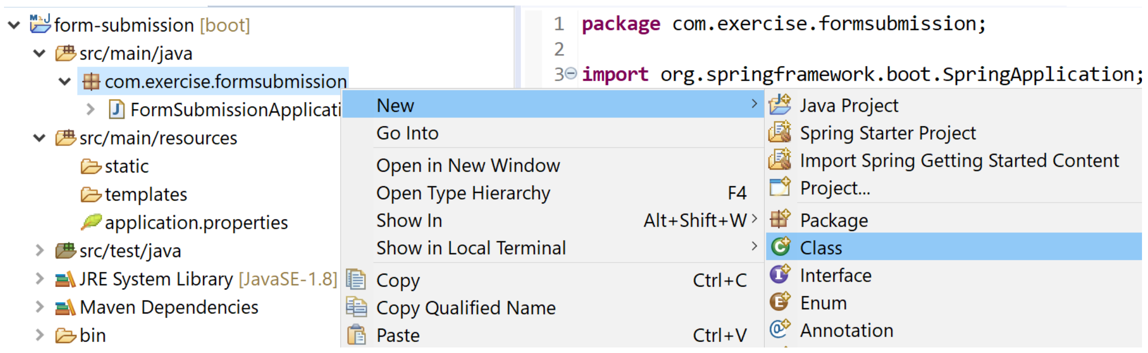
```
                        <groupId>org.junit.vintage</groupId>
                        <artifactId>junit-vintage-engine</artifactId>
                    </exclusion>
                </exclusions>
            </dependency>

            <dependency>
                <groupId>org.apache.tomcat.embed</groupId>
                <artifactId>tomcat-embed-jasper</artifactId>
                <scope>provided</scope>
            </dependency>

        </dependencies>

        <build>
```

7. Open application.properties (.\src\main\resources\application.properties) to configure the MVC’s view. Add the following code to the file:

```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
```

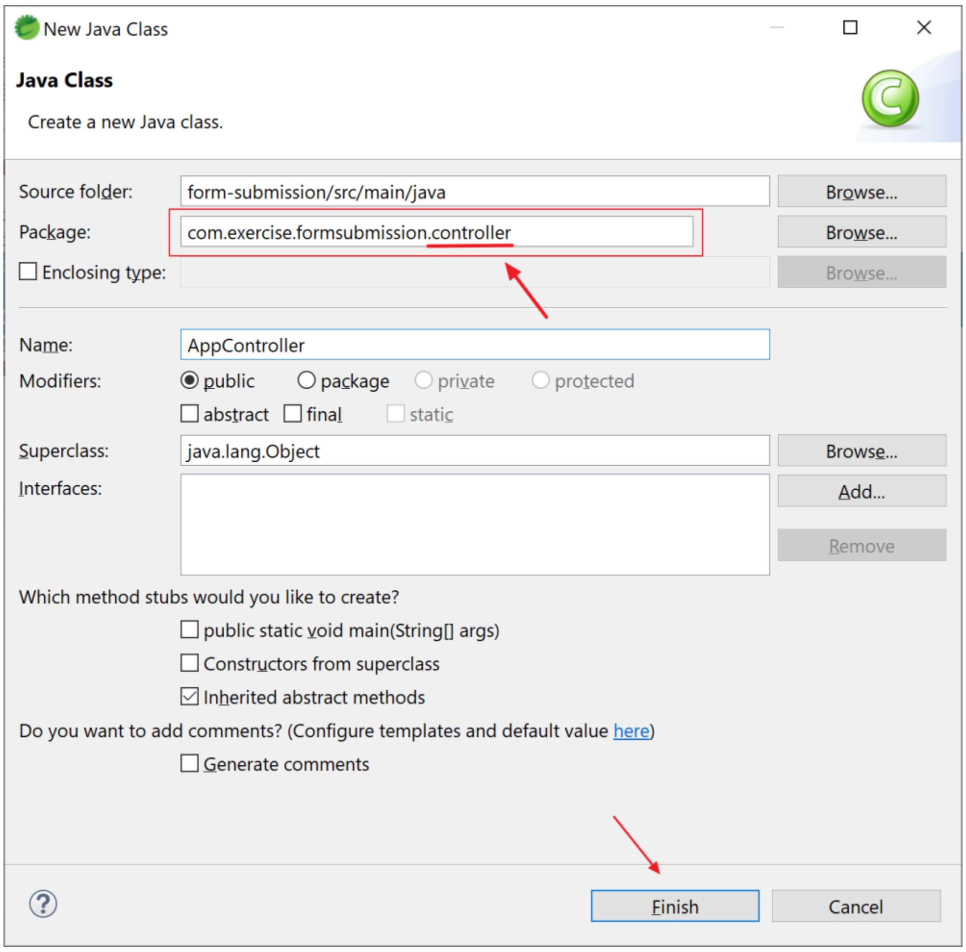
Note: if the view name is “showDetails”, then the view’s path will be “/WEB-INF/jsp/showDetails.jsp”.

8. Create the “Controller” class, right the package “com.exercise.formsubmission”, select “New” and then click “Class”:



```
package com.exercise.formsubmission;

import org.springframework.boot.SpringApplication;
```

9. Add a class “AppController” under the package “com.exercise.formsubmission.controller”:



New Java Class

Java Class

Create a new Java class.

Source folder: form-submission/src/main/java

Package: com.exercise.formsubmission.controller

☐ Enclosing type:

Name: AppController

Modifiers: [✓ public] package private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

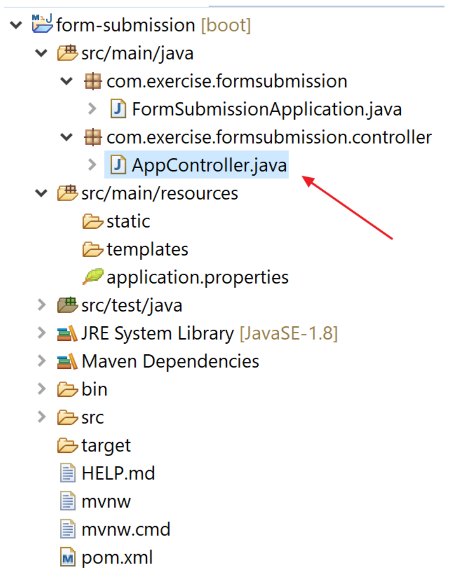
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments

Note: adding “.controller” to the package “com.exercise.formsubmission”.

10. Open the file "AppController.java" and insert the following code:



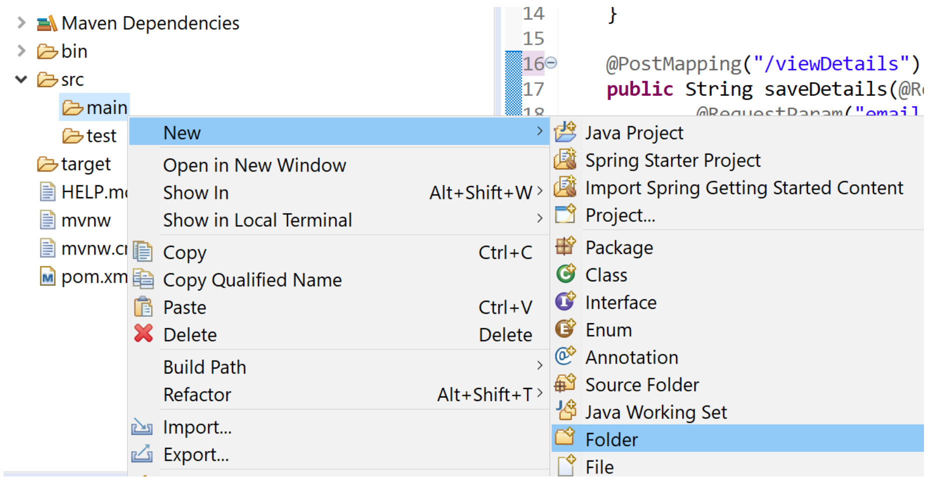
Here's the list of the code:

```
package com.exercise.formsubmission.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class AppController {
       @GetMapping("/getAppForm")
       public String getAppForm() {
              return "appForm";
       }

       @PostMapping("/viewAppDetails")
       public String viewAppDetails (@RequestParam("name") String name,
                     @RequestParam("email") String email, ModelMap modelMap) {

              modelMap.put("name", name);
              modelMap.put("email", email);
              return "appDetails";
       }
}
```

Note:

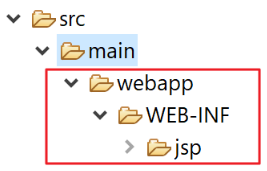
/getAppForm is the URL to the application form
/viewAppDetails is the URL to show the input details of the form

11. Implementing the views. Create a new folder “webapp” under “./src/main/”:

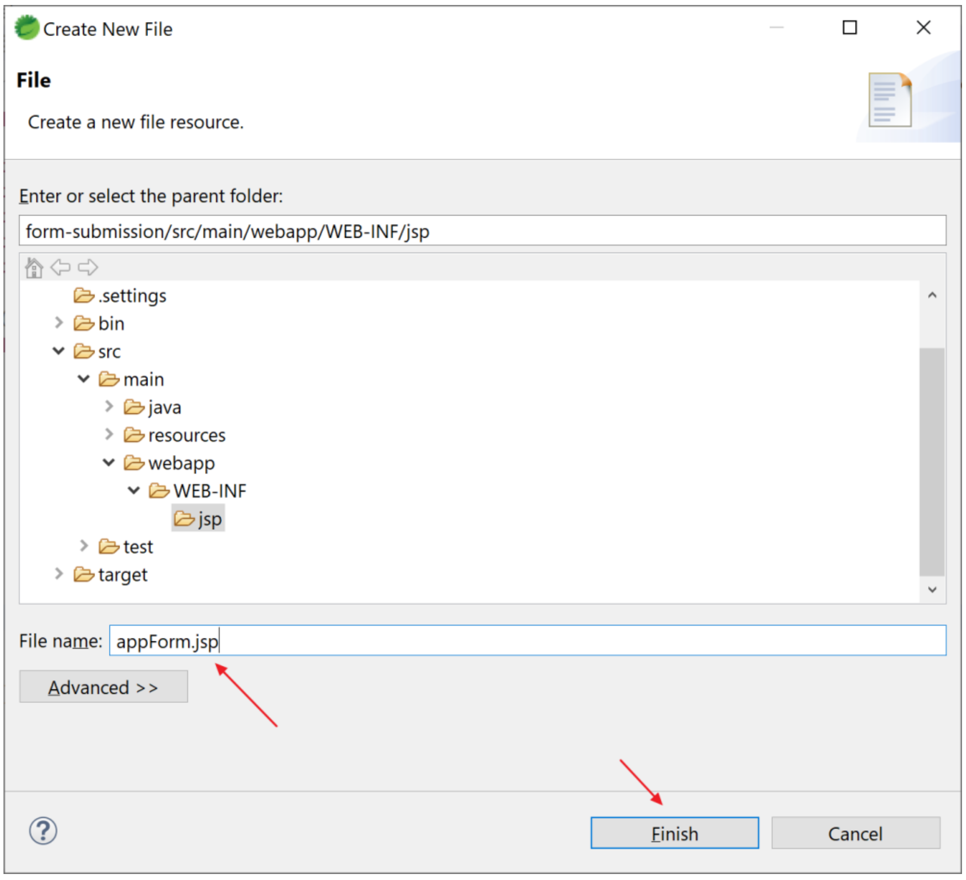


```
    }

    @PostMapping("/viewDetails")
    public String saveDetails(@R
```

Repeat the steps to create the following structure (src/main/webapp/WEB-INF/jsp/):



11a. Create the file appForm.jsp under “src/main/webapp/WEB-INF/jsp/”:



Enter or select the parent folder: form-submission/src/main/webapp/WEB-INF/jsp

File name: appForm.jsp

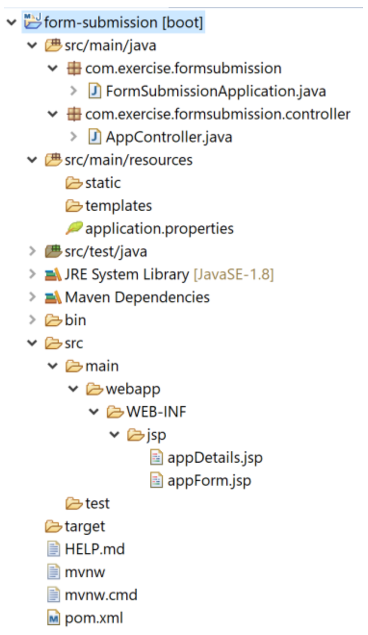
Open the JSP and insert the following code:

```
<html>
<head>
<title>form submit exercise</title>
</head>
<body>
       <h2>Application Details</h2>
       <form method="post" action="viewAppDetails">
              Enter Name : <input type="text" name="name" /> <br>
              Enter Email Address: <input type="email" name="email"> <br>
              <input type="submit" value="Submit">
       </form>
</body>
</html>
```

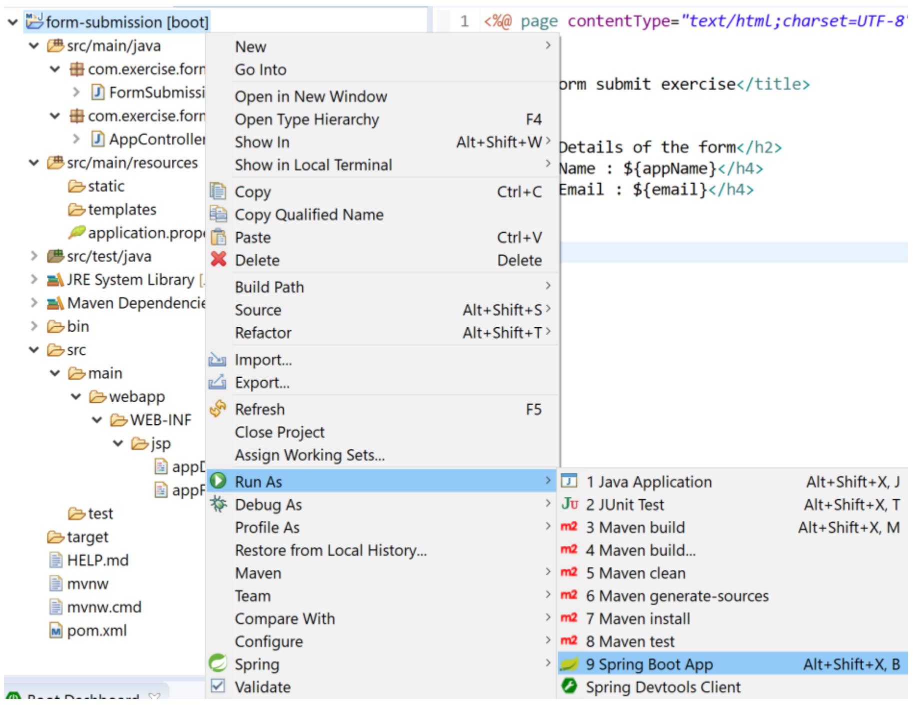
11b. Repeat the step to create the file appDetails.jsp under “src/main/webapp/WEB-INF/jsp/”:

```
<%@ page contentType="text/html;charset=UTF-8" language="java"%>
<html>
<head>
<title>form submit exercise</title>
</head>
<body>
       <h2>Details of the form</h2>
       <h4>Name : ${name}</h4>
       <h4>Email : ${email}</h4>
</body>
</html>
```

12. The final project look like:

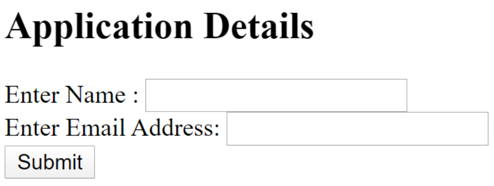


13. Run the application - Right click “form-submission”, click “Run As” and select “9 Spring Boot App”:


```
1  <%@ page contentType="text/html;charset=UTF-8
orm submit exercise</title>
Details of the form</h2>
Name : ${appName}</h4>
Email : ${email}</h4>
```

14. Verify the page – Open a browser and visit the following page:

http://localhost:8080/getAppForm

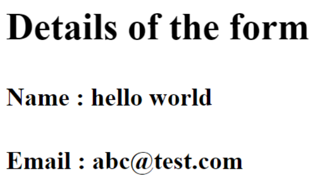


Application Details

Enter Name :

Enter Email Address:

After inputting the name and email, click “Submit”:

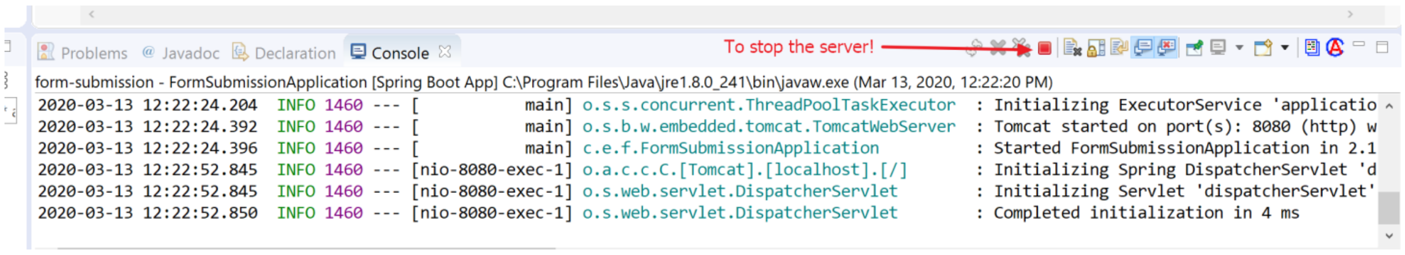


Details of the form

Name : hello world

Email : abc@test.com

15. Stop the server:



```
form-submission - FormSubmissionApplication [Spring Boot App] C:\Program Files\Java\jre1.8.0_241\bin\javaw.exe (Mar 13, 2020, 12:22:20 PM)
2020-03-13 12:22:24.204  INFO 1460 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicatio
2020-03-13 12:22:24.392  INFO 1460 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8080 (http) w
2020-03-13 12:22:24.396  INFO 1460 --- [           main] c.e.f.FormSubmissionApplication          : Started FormSubmissionApplication in 2.1
2020-03-13 12:22:52.845  INFO 1460 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'd
2020-03-13 12:22:52.845  INFO 1460 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2020-03-13 12:22:52.850  INFO 1460 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 4 ms
```

16. In case you want to use command line to run the application, start up the “Command Prompt” and change directory to your project, e.g. C:\java-ex\form-submission. Type the following command:

```
mvn spring-boot:run
```

Note: press Ctrl-C to stop the server.

17. Congratulations, if you can see the page! Try this one if you have problem with this exercise:

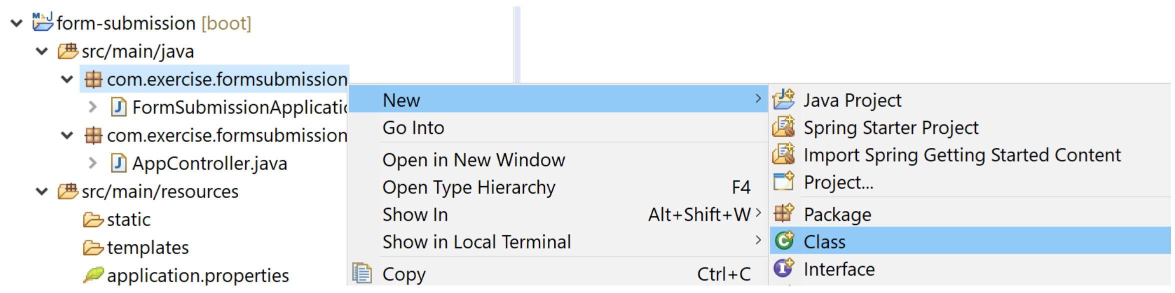
form-submission_complete.zip

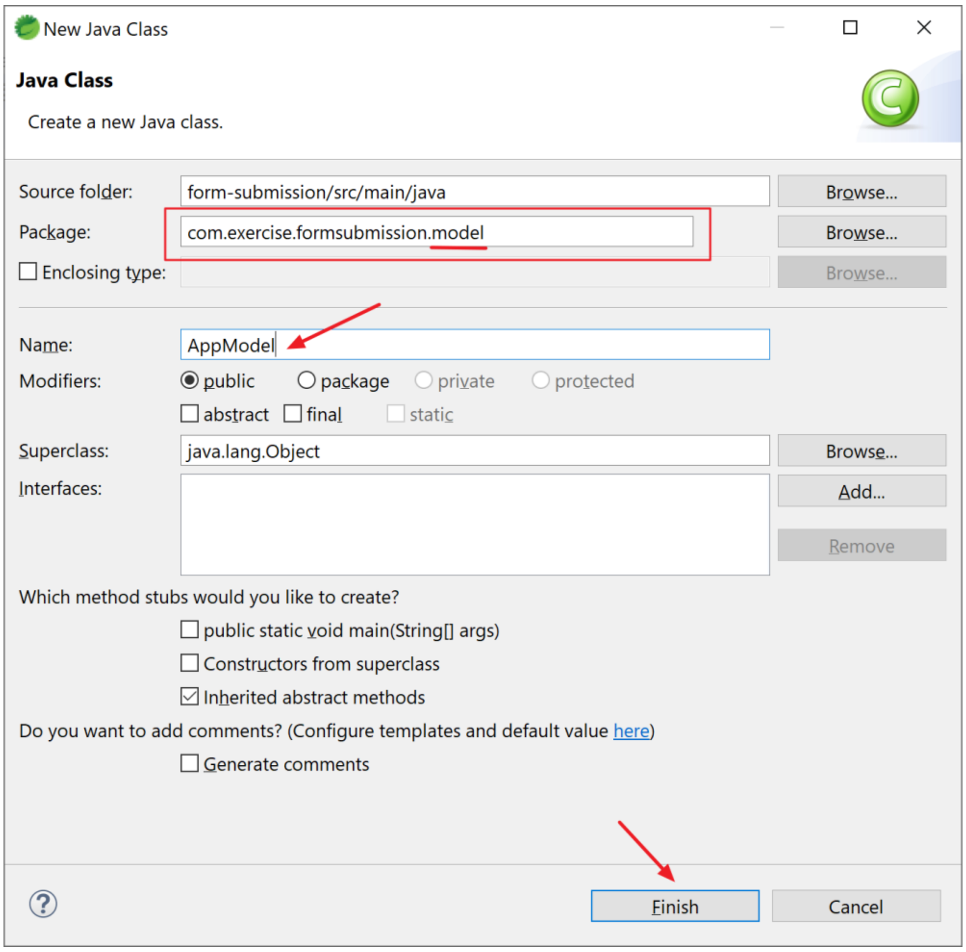
3 Exercise 2 - bonus (to inject a bean)

Prerequisites:

Java installed (e.g. JDK 1.8)
Maven installed (e.g. version 3.6.3)
Spring Tool Suite (STS) (e.g. 4.5.1)
Exercise 1 is finished

1. Create a model class “AppModel” under “./src/main/java/formsubmission/model/”:




New Java Class

Java Class

Create a new Java class.

Source folder: form-submission/src/main/java
Package: com.exercise.formsubmission.model
☐ Enclosing type:
Name: AppModel
Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces:

Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

2. Open “AppModel.java” and insert the following code:

```
package com.exercise.formsubmission.model;

public class AppModel {
	String appName;

	public String getAppName() {
		return appName;
	}

	public void setAppName(String appName) {
		this.appName = appName;
	}

}
```

3. Modified “AppController.java”:
3a. Create and autowire the bean – insert the following code to “AppController.java”:

```
    @Autowired
    private AppModel appModel;

    @Bean
    public AppModel getAppName() {
       AppModel appModel = new AppModel();
       appModel.setAppName("form-submission");
       return appModel;
    }
```

3b. Pass the value from the bean to the JSP – insert the following code to function viewAppDetails():

```
		modelMap.put("appName", appModel.getAppName());
```

```
}
```

Note:
i. we set the appName to “form-submission” (hard coded).
ii. for your reference:

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import com.exercise.formsubmission.model.AppModel;
```

4. Display the value in JSP – edit “appDetails.jsp”

change from

```
	<h2>Details of the form</h2>
```

to

```
	<h2>Details of the form - ${appName}</h2>
```

```
</html>
```

5. Re-start the application, and you should see the appName (form-submission):

6. Get value from property file – edit “application.properties”, insert the following code:

```
app.name=form-submission
```

```
app.name=form-submission
```

7. Modify the bean to read value from property file – edit “AppController.java”

7a. Define the appName:

```
        @Value( "${app.name}" )
        private String appName;
```

7b. Set the bean:

```
        appModel.setAppName(appName);
```

```
    }
```

8. Re-start the application, and you should see the appName (read from property file):

Note: you may modify the value in “application.properties” to see it really works.

9. Congratulations, if you can see the page! Try this one if you have problem with this exercise:

form-submission_bonus.zip

4 Exercise 3 – to access database

Prerequisites:

Java installed (e.g. JDK 1.8)
Maven installed (e.g. version 3.6.3)
Spring Tool Suite (STS) (e.g. 4.5.1)
exercise 2 finished

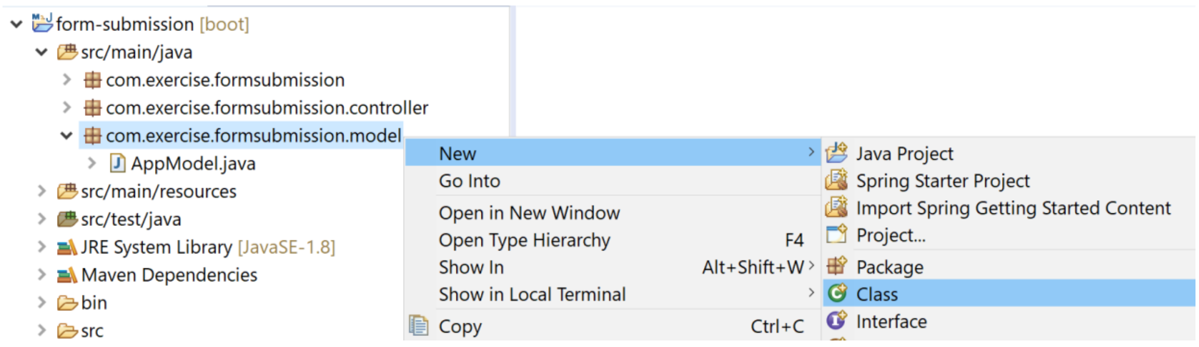
1. Modify “pom.xml” to add dependencies for database persistence. Open the file and insert the following code:

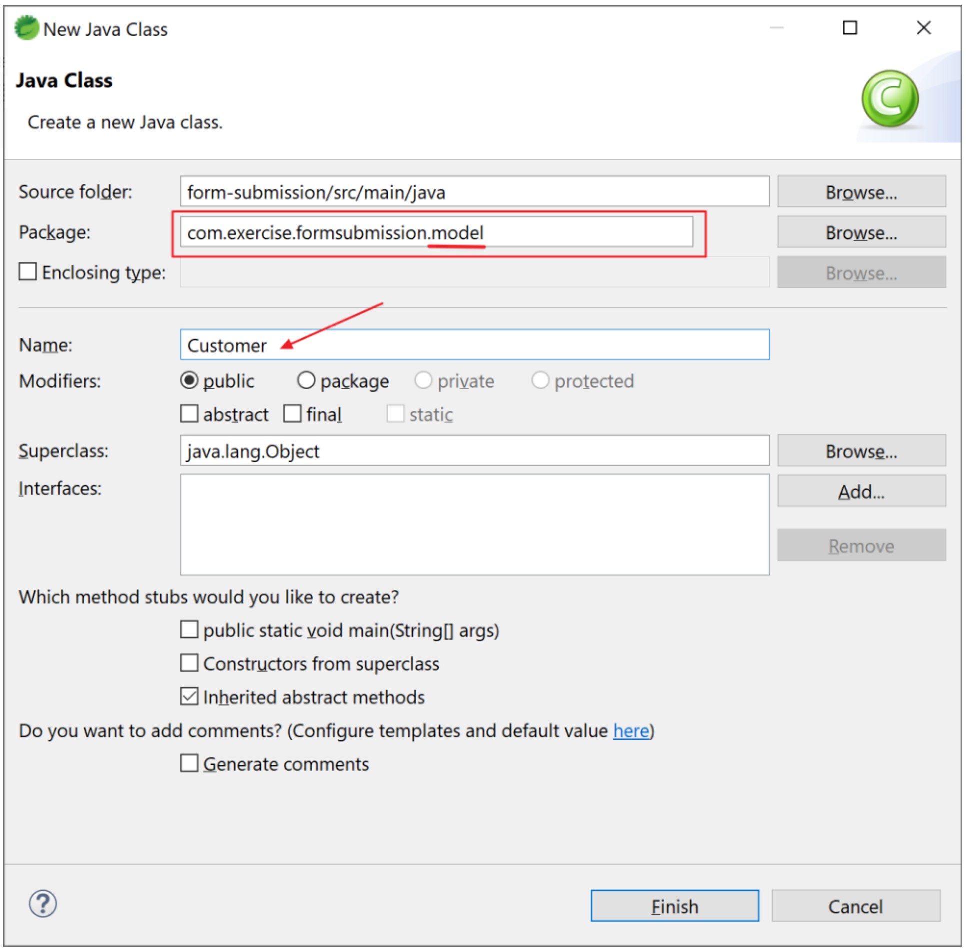
```
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
                <groupId>org.hsqldb</groupId>
                <artifactId>hsqldb</artifactId>
                <scope>runtime</scope>
        </dependency>
        <dependency>
                <groupId>javax.servlet</groupId>
                <artifactId>jstl</artifactId>
        </dependency>
```

```
        </plugins>
```

Note: “spring-boot-starter-data-jpa” is for Spring Data JPA
“hsqldb” is for Hyper SQL Database
“jstl” is for JSTL in JSP

2. Create the model class “Customer” under package “com.exercise.formsubmission.model”:




New Java Class

Java Class

Create a new Java class.

Source folder: form-submission/src/main/java

Package: com.exercise.formsubmission.model

☐ Enclosing type:

Name: Customer

Modifiers: [✓ public] package private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

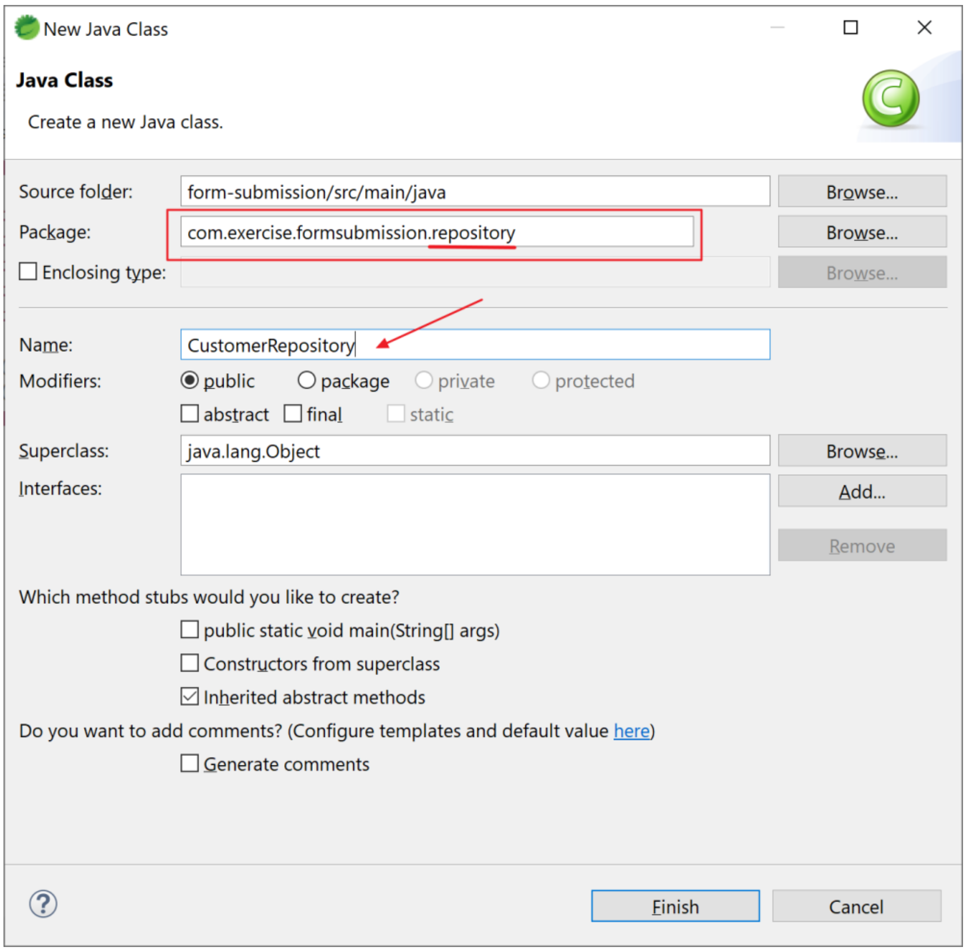
☐ Generate comments

3. Open the newly created file and insert the following code:

```
package com.exercise.formsubmission.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "customers")
public class Customer {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private long id;

	private String name;

	private String email;

	public long getId() {
		return id;
	}
	public void setId(long id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}

	public String getEmail() {
		return email;
	}
	public void setEmail(String email) {
		this.email = email;
	}
}
```

Note:	@Entity is to annotate this is a model class for persistence
		@Table is to annotate the table name in DB
		@Id is to annotate the primary key of the table

4. Create the repository class "CustomerRepository" under package "com.exercise.formsubmission.repository" for accessing the database:



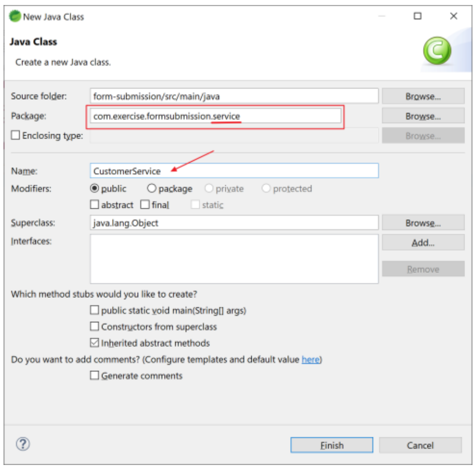
Source folder: form-submission/src/main/java
Package: com.exercise.formsubmission.repository
☐ Enclosing type:
Name: CustomerRepository
Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

5. Open the newly created file and insert the following code:

```
package com.exercise.formsubmission.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.exercise.formsubmission.model.Customer;

public interface CustomerRepository extends JpaRepository<Customer, String> {
}
```

Note: we don't need to implement the functions for basic operations, e.g. save(), findAll(), etc.

6. Create a service for controller to call. Create the service class "CustomerService" under package "com.exercise.formsubmission.service":



New Java Class

Java Class

Create a new Java class.

Source folder: form-submission/src/main/java
Package: com.exercise.formsubmission.service
☐ Enclosing type:
Name: CustomerService
Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

7. Open the newly created file and insert the following code:

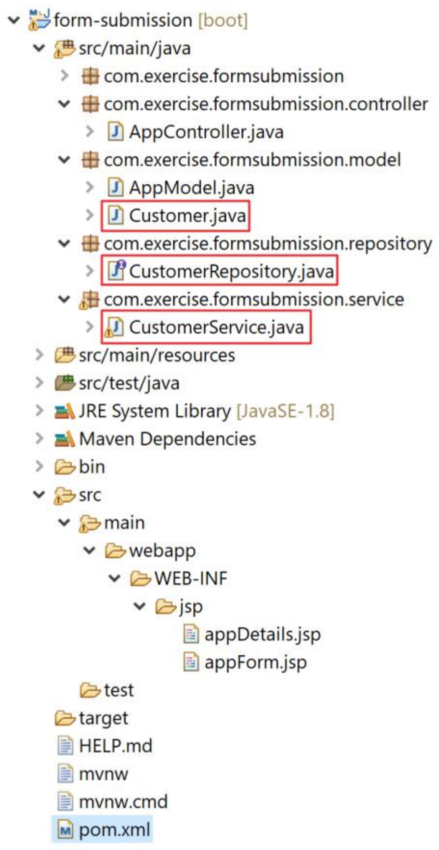
```
package com.exercise.formsubmission.service;

import java.util.ArrayList;
import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.exercise.formsubmission.model.Customer;
import com.exercise.formsubmission.repository.CustomerRepository;

@Service
public class CustomerService {

	@Autowired
	CustomerRepository customerRepository;

	public List<Customer> getAllCustomers() {
		List<Customer> customers = customerRepository.findAll();
		return customers;
	}

	public void addCustomer(Customer customer) {
		Customer c = customerRepository.save(customer);
	}

	public void updateCustomer(String id, Customer customer) {
		customerRepository.save(customer);
```

```
        }

        public void deleteCustomer(String id) {
                customerRepository.deleteById(id);
        }
}
```

Note:

i. use @Autowired to inject CustomerRepository
ii. we don’t need to implement the funcitons findAll, save and deleteById yourself, Spring Data will implement the functions for us.

8. the structure of the project now should look like this:



```
form-submission [boot]
  src/main/java
    com.exercise.formsubmission
    com.exercise.formsubmission.controller
      AppController.java
    com.exercise.formsubmission.model
      AppModel.java
      Customer.java
    com.exercise.formsubmission.repository
      CustomerRepository.java
    com.exercise.formsubmission.service
      CustomerService.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  bin
  src
    main
      webapp
        WEB-INF
          jsp
            appDetails.jsp
            appForm.jsp
    test
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

9. Use the service in controller – open “AppController.java” and edit the functions accordingly.

9a. getAppForm():

```
            @GetMapping("/getAppForm")
            public String getAppForm(ModelMap modelMap) {
                logger.info("get customers");
                List<Customer> customers = customerService.getAllCustomers();
                for(Customer c : customers) {
                        logger.info("###### {}-{}-{}", c.getId(), c.getName(), c.getEmail());
                }
                modelMap.addAttribute("customers", customers);
                return "appForm";
            }
```

Note: calling customerService.getAllCustomers() to get all customers from database.

9b. viewAppDetails():

```
        @PostMapping("/viewAppDetails")
        public String viewAppDetails (@RequestParam("name") String name,
                        @RequestParam("email") String email, ModelMap modelMap) {

                modelMap.put("name", name);
                modelMap.put("email", email);
                modelMap.put("appName", appModel.getAppName());

                Customer customer = new Customer();
                customer.setName(name);
                customer.setEmail(email);
                customerService.addCustomer(customer);
                logger.info("add customer");

                return "appDetails";
        }
```

Note: saving the form to database using customerService.save().

Here’s the code of AppController.java:

```
package com.exercise.formsubmission.controller;

import java.util.List;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.stereotype.Controller;
import org.springframework.ui.ModelMap;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestParam;
import com.exercise.formsubmission.model.AppModel;
import com.exercise.formsubmission.model.Customer;
import com.exercise.formsubmission.service.CustomerService;

@Controller
public class AppController {
        final Logger logger = LoggerFactory.getLogger(AppController.class);

        @Autowired
        CustomerService customerService;

        @GetMapping("/getAppForm")
        public String getAppForm(ModelMap modelMap) {
                logger.info("get customers");
                List<Customer> customers = customerService.getAllCustomers();
                for(Customer c : customers) {
                        logger.info("###### {}-{}-{}", c.getId(), c.getName(), c.getEmail());
                }
```

```
              modelMap.addAttribute("customers", customers);
              return "appForm";
       }

       @PostMapping("/viewAppDetails")
       public String viewAppDetails (@RequestParam("name") String name,
                     @RequestParam("email") String email, ModelMap modelMap) {

              modelMap.put("name", name);
              modelMap.put("email", email);
              modelMap.put("appName", appModel.getAppName());

              Customer customer = new Customer();
              customer.setName(name);
              customer.setEmail(email);
              customerService.addCustomer(customer);
              logger.info("add customer");

              return "appDetails";
       }

       @Value( "${app.name}" )
       private String appName;

    @Autowired
    private AppModel appModel;

    @Bean
    public AppModel getAppName() {
       AppModel appModel = new AppModel();
       appModel.setAppName(appName);
       return appModel;
    }
}
```

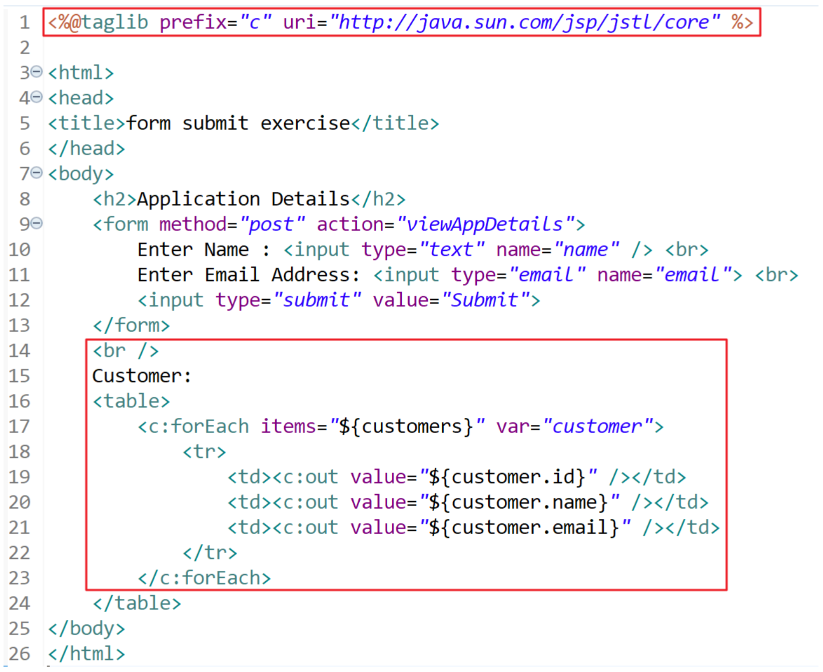
10. Let's show the retrieved data in the JSP. Open "appForm.jsp" and add the following code below the form.

10a. use JSTL in JSP, add the following at the first line:

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
```

10b. add the following code in body:

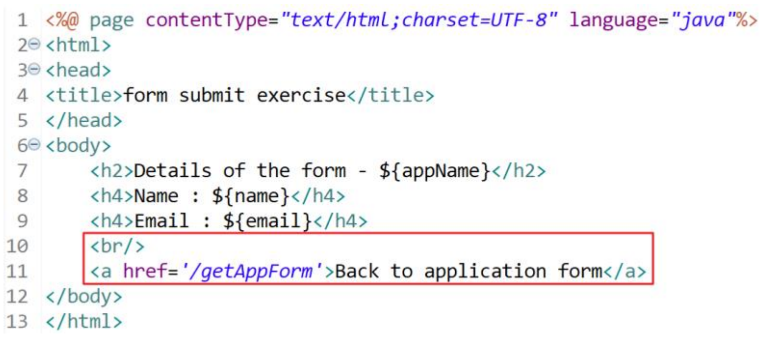
```
       <br />
       Customer:
       <table>
              <c:forEach items="${customers}" var="customer">
                     <tr>
                            <td><c:out value="${customer.id}" /></td>
                            <td><c:out value="${customer.name}" /></td>
                            <td><c:out value="${customer.email}" /></td>
                     </tr>
              </c:forEach>
       </table>
```



```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>
<head>
<title>form submit exercise</title>
</head>
<body>
    <h2>Application Details</h2>
    <form method="post" action="viewAppDetails">
        Enter Name : <input type="text" name="name" /> <br>
        Enter Email Address: <input type="email" name="email"> <br>
        <input type="submit" value="Submit">
    </form>
    <br />
    Customer:
    <table>
        <c:forEach items="${customers}" var="customer">
            <tr>
                <td><c:out value="${customer.id}" /></td>
                <td><c:out value="${customer.name}" /></td>
                <td><c:out value="${customer.email}" /></td>
            </tr>
        </c:forEach>
    </table>
</body>
</html>
```

11. Add a link in “appDetails.jsp” to redirect the page back to “appForm.jsp”:

```
        <br/>
        <a href='/getAppForm'>Back to application form</a>
```



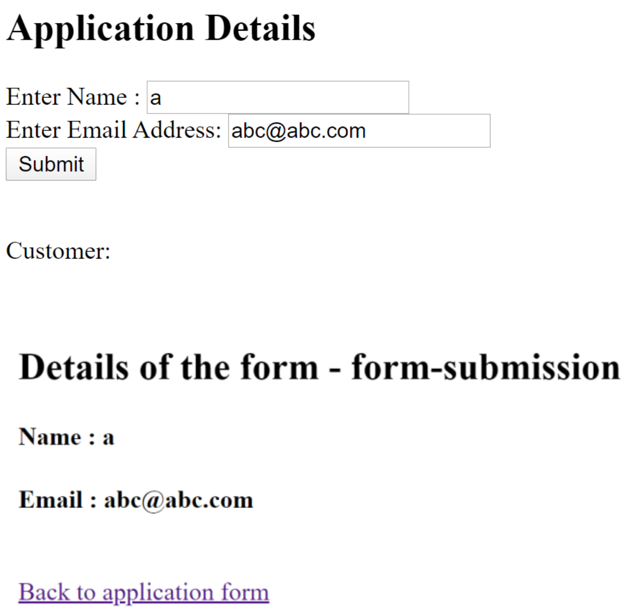
```
<%@ page contentType="text/html;charset=UTF-8" language="java"%>
<html>
<head>
<title>form submit exercise</title>
</head>
<body>
    <h2>Details of the form - ${appName}</h2>
    <h4>Name : ${name}</h4>
    <h4>Email : ${email}</h4>
    <br/>
    <a href='/getAppForm'>Back to application form</a>
</body>
</html>
```

12. Run the application and examine the results. If you do not remember how to start the application, please revisit Exercise 1 step 13.

Open a browser and visit:
http://localhost:8080/getAppForm

Note: the list of “Customer” is empty.

After submitting the form, this details and a link “Back to application form” show:



Application Details

Enter Name : a
Enter Email Address: abc@abc.com
Submit

Customer:

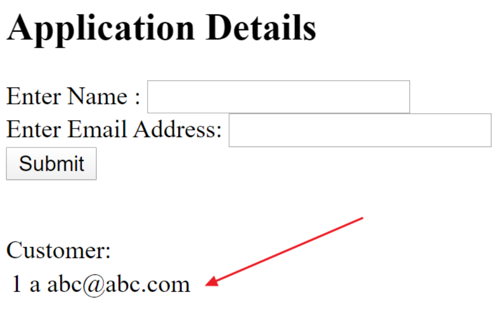
Details of the form - form-submission

Name : a

Email : abc@abc.com

Back to application form

After clicking “Back to application form”, the customer list shows (retrieve from database):



Application Details

Enter Name :
Enter Email Address:
Submit

Customer:
1 a abc@abc.com

13. Congratulations, if you can see the page! Try this one if you have problem with this exercise:

form-submission_2a_db_complete.zip

5 Exercise 4 - bonus (to connect external database)

Prerequisites:

Java installed (e.g. JDK 1.8)
Maven installed (e.g. version 3.6.3)
Spring Tool Suite (STS) (e.g. 4.5.1)
Exercise 3 is finished

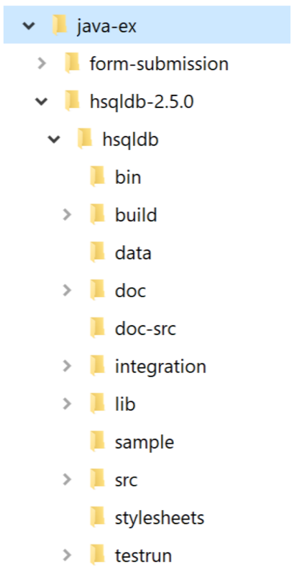
1. Question: In above exercise, we don’t have any configurations about connecting the database, why does it work?
   Answer: You need only include a build dependency to the **embedded** database that you want to use. All the work will be done by Spring Data.

   Question: How can I configure to connect to an **external** database?
   Answer: Follow this exercise to find the answer.

2. Before any changes to the code, we need an external database. In this exercise, we are going to use HSQLDB again.

2a. Download the HSQLDB (e.g. hsqldb-2.5.0.zip) from
http://hsqldb.org/
or directly from
https://sourceforge.net/projects/hsqldb/files/latest/download.

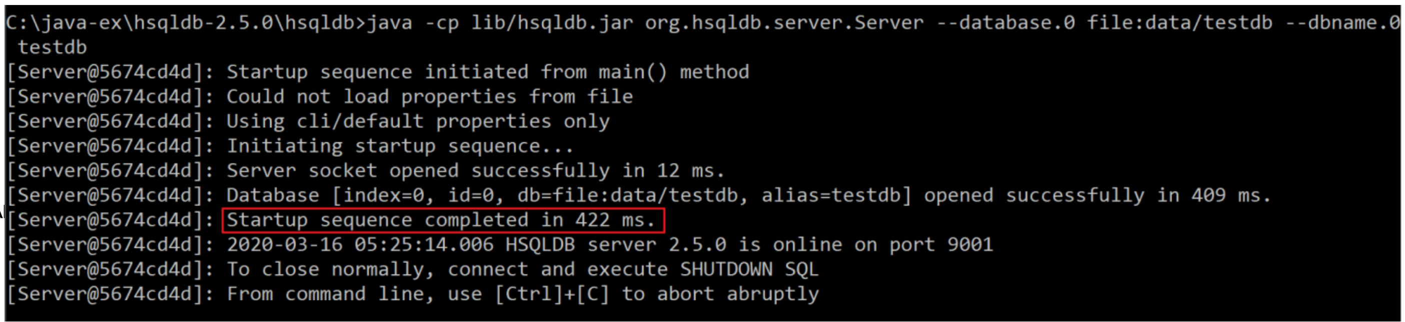
2b. Extract your downloaded file to a folder, e.g. C:\java-ex\hsqldb-2.5.0\.



2c. Start the server. Start a “Command Prompt” and go to the home directory of HSQLDB server, e.g. C:\java-ex\hsqldb-2.5.0\hsqldb and run the following command:

```
java -cp lib/hsqldb.jar org.hsqldb.server.Server --database.0 file:data/testdb --dbname.0
```

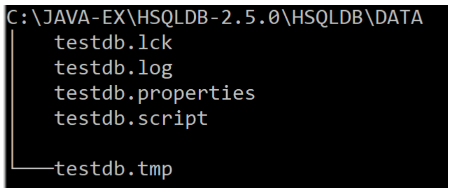
A



```
C:\java-ex\hsqldb-2.5.0\hsqldb>java -cp lib/hsqldb.jar org.hsqldb.server.Server --database.0 file:data/testdb --dbname.0
 testdb
[Server@5674cd4d]: Startup sequence initiated from main() method
[Server@5674cd4d]: Could not load properties from file
[Server@5674cd4d]: Using cli/default properties only
[Server@5674cd4d]: Initiating startup sequence...
[Server@5674cd4d]: Server socket opened successfully in 12 ms.
[Server@5674cd4d]: Database [index=0, id=0, db=file:data/testdb, alias=testdb] opened successfully in 409 ms.
[Server@5674cd4d]: Startup sequence completed in 422 ms.
[Server@5674cd4d]: 2020-03-16 05:25:14.006 HSQLDB server 2.5.0 is online on port 9001
[Server@5674cd4d]: To close normally, connect and execute SHUTDOWN SQL
[Server@5674cd4d]: From command line, use [Ctrl]+[C] to abort abruptly
```

```
testdb
```

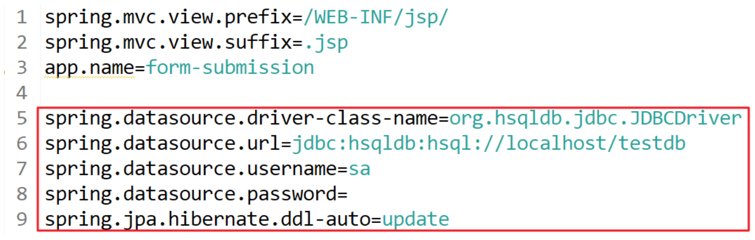
Note: data files will be stored in “data” directory with prefix “testdb”:



```
C:\JAVA-EX\HSQLDB-2.5.0\HSQLDB\DATA
│   testdb.lck
│   testdb.log
│   testdb.properties
│   testdb.script
│
└───testdb.tmp
```

3. Add configuration for the database connection in “application.properties”:

```
spring.datasource.driver-class-name=org.hsqldb.jdbc.JDBCDriver
spring.datasource.url=jdbc:hsqldb:hsql://localhost/testdb
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
```

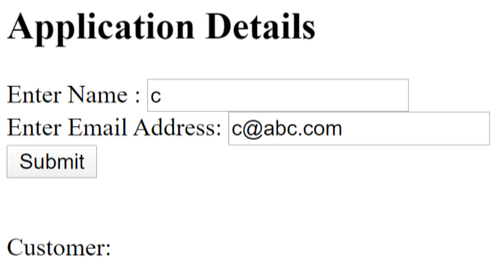


```
spring.mvc.view.prefix=/WEB-INF/jsp/
spring.mvc.view.suffix=.jsp
app.name=form-submission

spring.datasource.driver-class-name=org.hsqldb.jdbc.JDBCDriver
spring.datasource.url=jdbc:hsqldb:hsql://localhost/testdb
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=update
```

Note: Do not contain any white space in the above configurations.

4. Restart the application and visit the following page:

http://localhost:8080/getAppForm

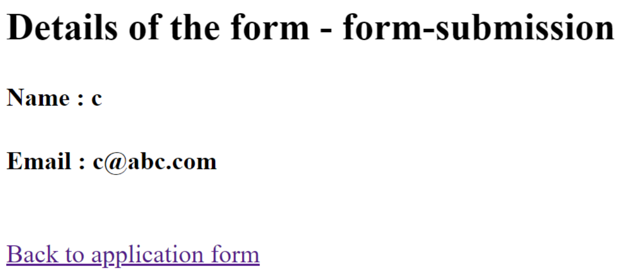


Application Details

Enter Name : c
Enter Email Address: c@abc.com

Customer:

4a. Fill in the form and submit:



Details of the form - form-submission

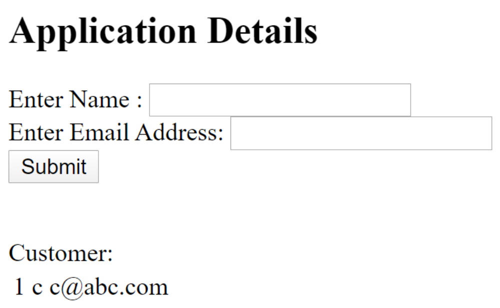
Name : c

Email : c@abc.com

Back to application form

4b. Click “Back to application from”:

4c. The Customer list should show:

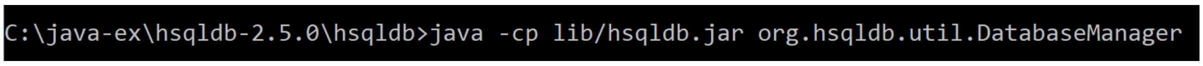

Application Details

Enter Name :
Enter Email Address:

Customer:
1 c c@abc.com

5. Verify if the data is stored in the external database. Start a “command prompt” and go to the home directory of HSQLDB server, e.g. C:\java-ex\hsqldb-2.5.0\hsqldb and run the following command:

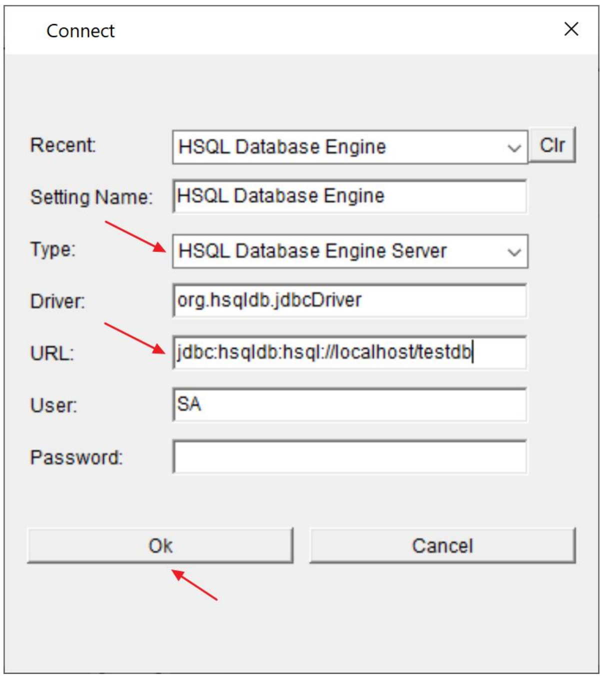
```
java -cp lib/hsqldb.jar org.hsqldb.util.DatabaseManager
```



```
C:\java-ex\hsqldb-2.5.0\hsqldb>java -cp lib/hsqldb.jar org.hsqldb.util.DatabaseManager
```

5a. database manager should show with a connection input form:

```
Type: HSQL Database Engine Server
URL: jdbc:hsqldb:hsql://localhost/testdb
```

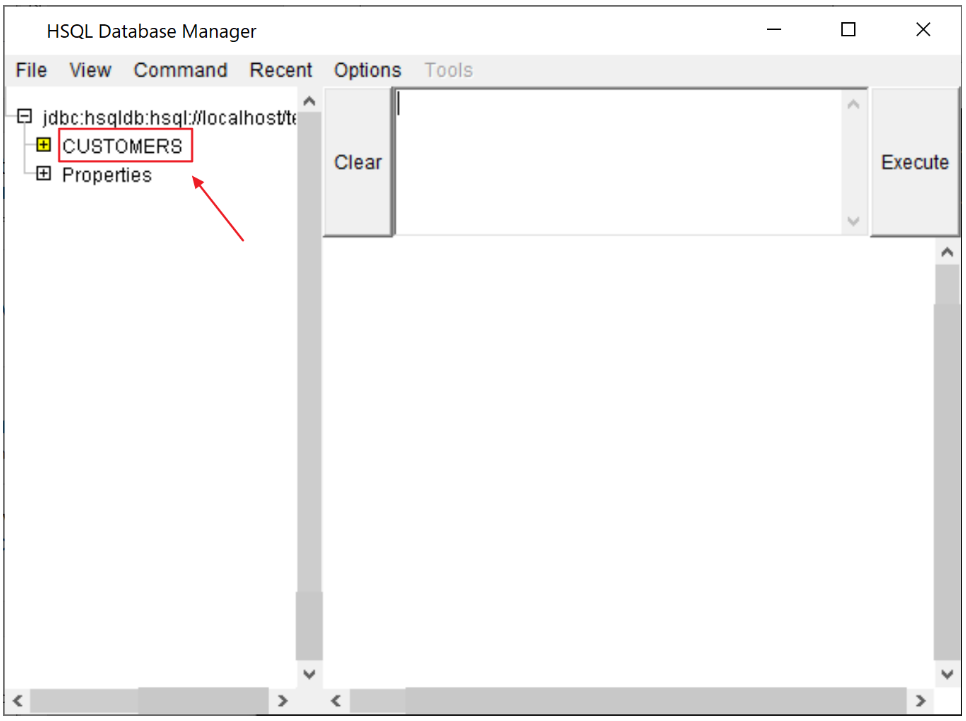


Recent: HSQL Database Engine

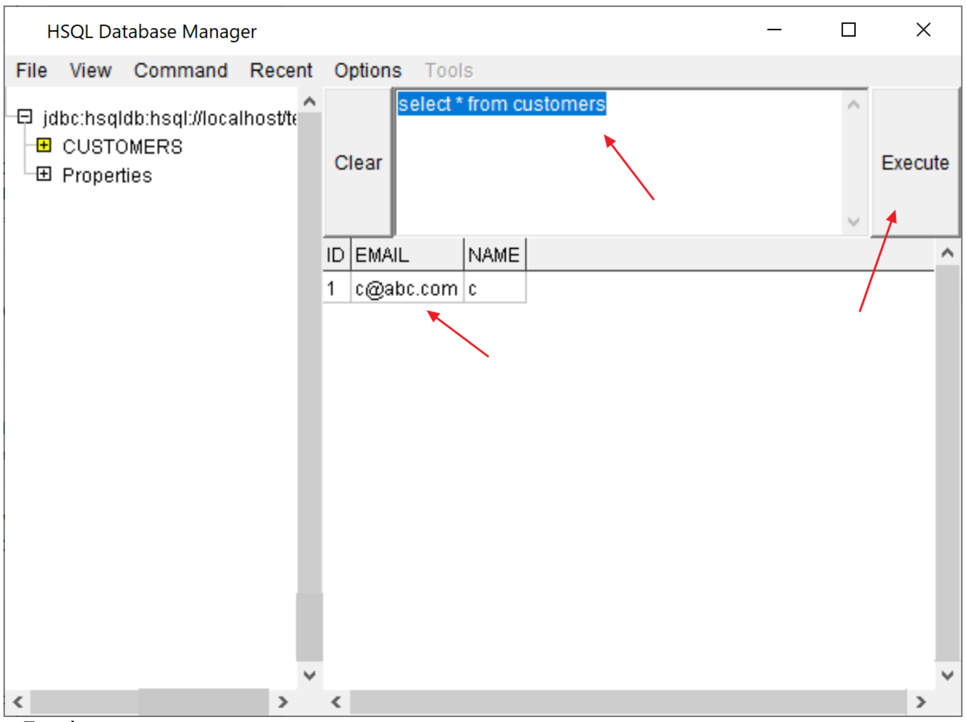
Setting Name: HSQL Database Engine
Type: HSQL Database Engine Server
Driver: org.hsqldb.jdbcDriver
URL: jdbc:hsqldb:hsql://localhost/testdb
User: SA
Password:

Note: leave Password empty (this is the default password)

5b. the following panel should show:


HSQL Database Manager
File View Command Recent Options Tools
jdbc:hsqldb:hsql://localhost/te
CUSTOMERS
Properties
Clear
Execute

5c. verify the table, execute the following SQL:


HSQL Database Manager
File View Command Recent Options Tools
jdbc:hsqldb:hsql://localhost/te
CUSTOMERS
Properties
Clear
Execute

```
select * from customers
```

ID	EMAIL	NAME
1	c@abc.com	c

```
select * from customers
```

You should see the data that you inputted in the form previously.

6. Congratulations! That’s it, we only need to configure the settings in “application.properties”. In case you have problem in this exercise. Try this one if you have problem with this exercise:

form-submission_2b_db_bonus.zip

~ End of the exercises ~