Table of Contents

Revision History

Date	Version	Description	Author
January 23, 2018	1.0	Created	Batjargal Bayarsaikhan (Alex)

Build a CRUD App using Spring Boot, Spring MVC, JPA, Hibernate, Thymeleaf and Bootstrap with MySQL database

This guide walks you through the process of building a simple CRUD application that uses Spring Boot, Spring MVC, JPA, Hibernate, Thymeleaf and Bootstrap with MySQL database.

1. What you'll build

You will build a simple "Student Info" web application with full CRUD (Create, Read, Update, Delete) functionality and save data in MySQL database using Hibernate, all using annotation configuration.

2. What you'll need

- About 40 minutes
- JDK 1.8 or later
- Maven 3.0+
- MySQL database
- You can also import the code straight into your IDE:
  - Spring Tool Suite (STS)
  - IntelliJ IDEA

3. What we'll cover

This tutorial will show you how to use the following tech stack:

- Spring Boot (1.5.9)
- Spring Security
- Spring MVC
- JPA
- Thymeleaf
- MySQL
- Bootstrap (UI presentation)
- Maven (3.3.9)
- STS (Spring Tool Suite)
- Java 8
- Packaging (JAR)

Let’s begin step by step.

4. Project Creation

There are a few ways to create a spring boot project:

- To start from scratch
- To use STS (File→New→Spring Starter Project
- To use Spring INITIALZR on https://start.spring.io/
- etc.

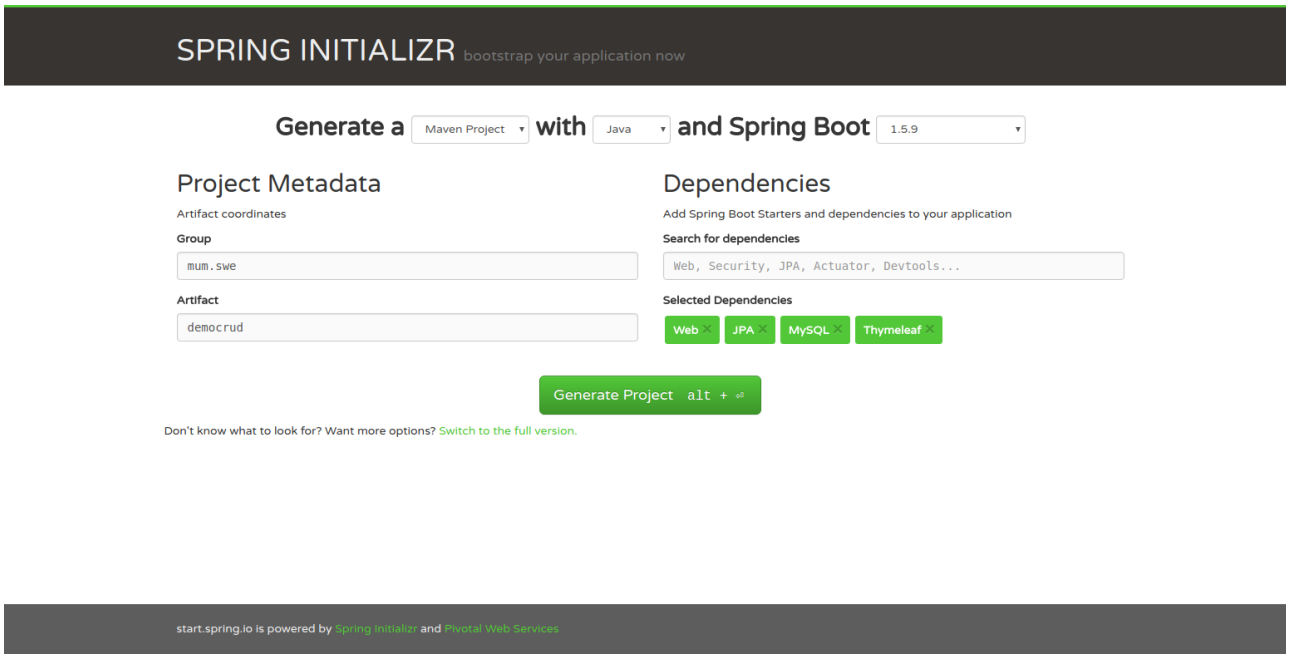
We will use Spring INITIALZR on https://start.spring.io/.

***Note**:*

- *Spring Initializr provides an extensible API to generate quickstart projects. (https://github.com/spring-io/initializr/)*

1. Go to https://start.spring.io.
2. Fill out the form as shown below. We need a **Maven Project** with **Java** and **Spring Boot 1.5.9**.
   Group: **mum.swe**
   Artifact: **democrud**

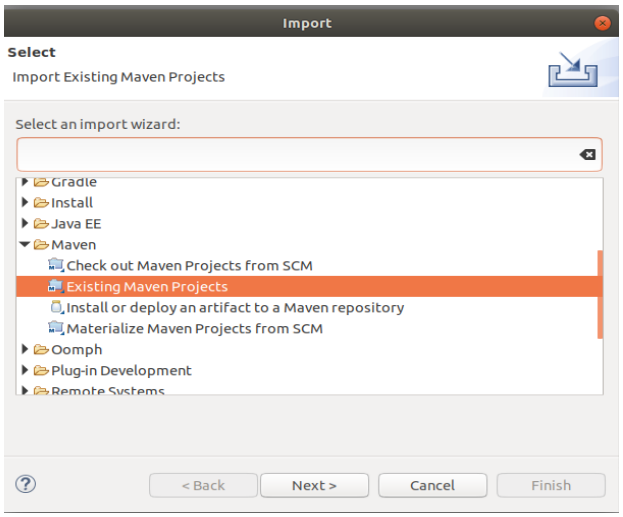
Enter and select your dependencies what you need in the “Search for dependencies” input. We need **Web, JPA, MySQL** and **Thymeleaf** libraries.



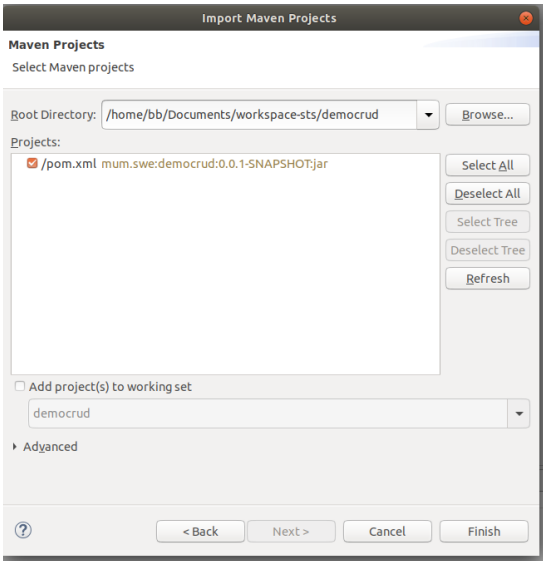
Then, click on **Generate Project** button and it will download a zip file (democrud.zip) with maven project.

5. Import project into Sprint Tool Suite (STS)

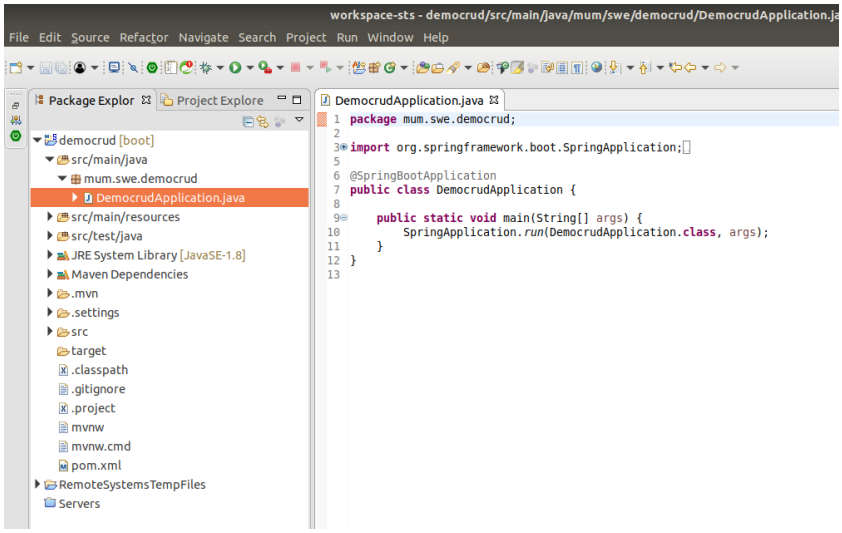
1. Unzip the zip file.
2. Select a menu, File→Import...
3. Import into STS as “Existing Maven Projects”.



4. Choose the root directory of the project, where the pom.xml file is located, and click Finish button.



5. It will display the project in the package explorer.



6. pom.xml file

In order to avoid strict HTML parsing issue and run Thymeleaf in “LEGACYHTML5” mode, we need to add an extra dependency library, called NekoHTML, in our pom.xml file.

```
<!-- https://mvnrepository.com/artifact/net.sourceforge.nekohtml/nekohtml -->
<dependency>
    <groupId>net.sourceforge.nekohtml</groupId>
    <artifactId>nekohtml</artifactId>
    <version>1.9.22</version>
</dependency>
```

***Notes**:*

- *A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project. (https://maven.apache.org/what-is-maven.html)*
- *Maven is a build automation tool used primarily for Java projects. (https://en.wikipedia.org/wiki/Apache_Maven)*
- *NekoHTML is a simple HTML scanner and tag balancer that enables application programmers to parse HTML documents and access the information using standard XML interfaces. (http://nekohtml.sourceforge.net/)*

7. Model Creation

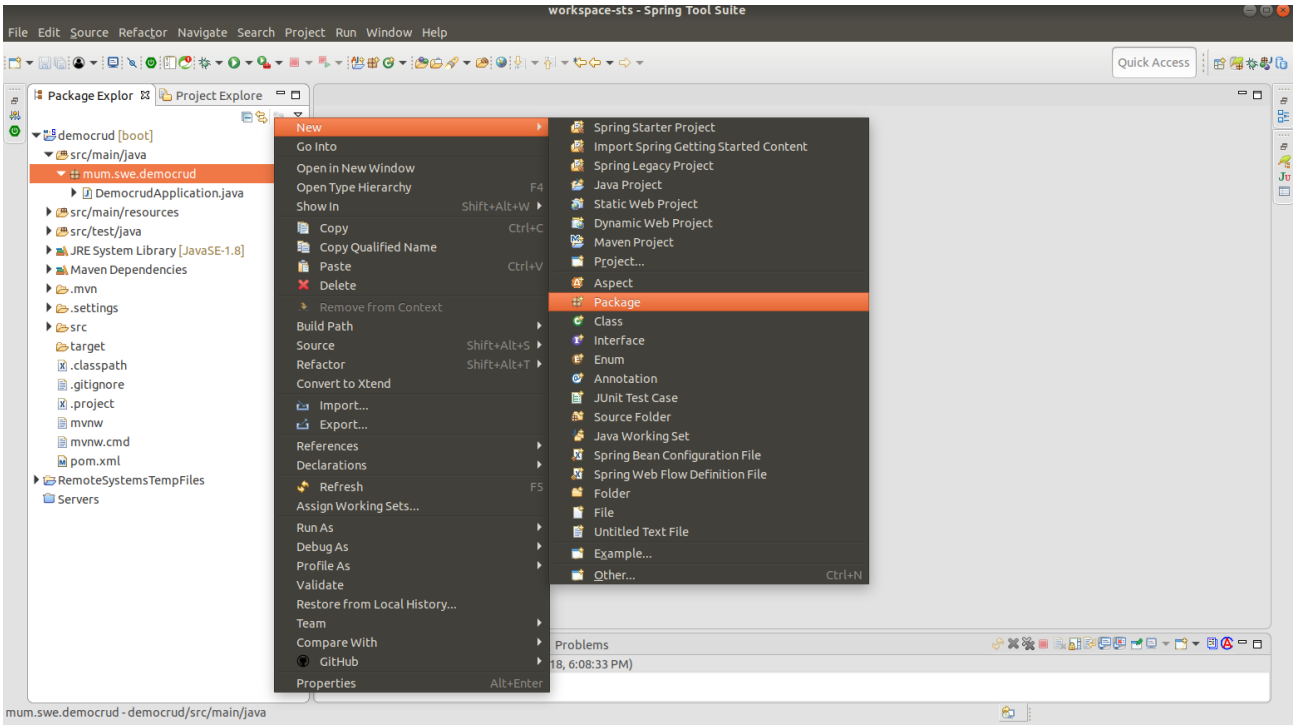
Now let’s create our first model class called Student. It is also known as Entity class.

7.1 Student model

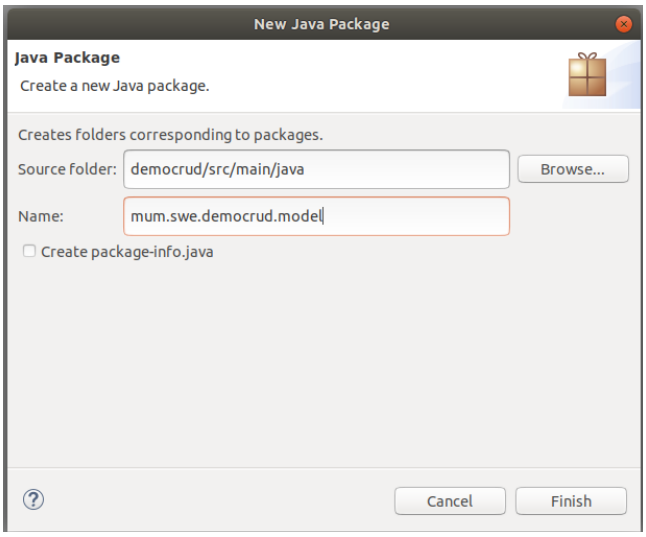
1. Create a package called “model“.

Following the best practice for package naming convention create a package called “model” inside our main package.

- Select the main package and right click. Select a menu, New→ Package.



- Enter a package name, *“mum.swe.democrud.model”, and click Finish.*



2. Create “Student” class in our “model” package.

This class includes the fields with simple validation annotations which are provided by Hibernate.

```
package mum.swe.democrud.model;

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

import org.hibernate.validator.constraints.Email;
import org.hibernate.validator.constraints.NotEmpty;

@Entity
@Table(name = "students")
public class Student {

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

	@Column(name = "first_name")
	@NotEmpty(message = "*Please provide first name")
	private String firstName;

	@Column(name = "last_name")
	@NotEmpty(message = "*Please provide last name")
	private String lastName;

	@Column(name = "email")
	@Email(message = "*Please provide a valid Email")
	@NotEmpty(message = "*Please provide an email")
	private String email;

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getFirstName() {
		return firstName;
	}

	public void setFirstName(String firstName) {
		this.firstName = firstName;
	}

	public String getLastName() {
		return lastName;
	}
```

```
	public void setLastName(String lastName) {
		this.lastName = lastName;
	}

	public String getEmail() {
		return email;
	}

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

8. Service Layer

We will create our student service layer which consists interface and implementation.

8.1 Interface

1. Create a package called “service” inside our main package, “mum.swe.democrud.service”.
2. Create “StudentService” interface in our “service” package.

```
package mum.swe.democrud.service;

import java.util.List;

import mum.swe.democrud.model.Student;

public interface StudentService {
	List<Student> findAll();
	Student save(Student student);
	Student findOne(Long id);
	void delete(Long id);
}
```

8.2 Implementation

1. Create “impl” package inside our service package, “mum.swe.democrud.service.impl”.
2. Create “StudentServiceImpl” class which implements “StudentService” interface in “impl” package.

```
package mum.swe.democrud.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import mum.swe.democrud.model.Student;
import mum.swe.democrud.repository.StudentRepository;
```

```
import mum.swe.democrud.service.StudentService;

@Service("studentService")
public class StudentServiceImpl implements StudentService {

	@Autowired
	StudentRepository studentRepository;

	@Override
	public List<Student> findAll() {
		return studentRepository.findAll();
	}

	@Override
	public Student save(Student student) {
		return studentRepository.save(student);
	}

	@Override
	public Student findOne(Long id) {
		return studentRepository.findOne(id);
	}

	@Override
	public void delete(Long id) {
		studentRepository.delete(id);
	}

}
```

9. Repository Layer

1. Create a package called “repository” inside our main package, “mum.swe.democrud.repository”.
2. Create “StudentRepository” interface which extends JpaRepository in our “repository” package.

```
package mum.swe.democrud.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import mum.swe.democrud.model.Student;

@Repository("studentRepository")
public interface StudentRepository extends JpaRepository<Student, Long> {
}
```

10. Controller Layer

We need two controller called HomeController and StudentController. HomeController is reponsible for managing home page. StudentController manages Student model’s CRUD functionality.

1. Create a package called “controller” inside our main package, “mum.swe.democrud.controller”.

2. Create “HomeController” class in our “controller” package.

```
package mum.swe.democrud.controller;

import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HomeController {

	@RequestMapping(value="/", method = RequestMethod.GET)
	public String home(){
		return "home/index";
	}
}
```

3. Create “StudentController” class in our “controller” package.

```
package mum.swe.democrud.controller;

import java.util.List;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.validation.BindingResult;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.servlet.ModelAndView;

import mum.swe.democrud.model.Student;
import mum.swe.democrud.service.StudentService;

@Controller
public class StudentController {

	@Autowired
	private StudentService studentService;

	@RequestMapping(value="/students", method = RequestMethod.GET)
	public ModelAndView students(){
		List<Student> students = studentService.findAll();
		ModelAndView modelAndView = new ModelAndView();
		modelAndView.addObject("students", students);
		modelAndView.setViewName("student/list");
		return modelAndView;
	}
```

```
	@RequestMapping(value="/student", method = RequestMethod.GET)
	public String create(Model model){
		model.addAttribute("student", new Student());
		return "student/edit";
	}

	@RequestMapping(value = "/student", method = RequestMethod.POST)
	public String edit(@Valid @ModelAttribute("student") Student student,
			BindingResult result, Model model)  {

		if (result.hasErrors()) {
			model.addAttribute("errors", result.getAllErrors());
			return "student/edit";
		}
		student = studentService.save(student);
		return "redirect:/students";
	}

	@RequestMapping(value="/student/{id}", method = RequestMethod.GET)
	public String view(@PathVariable Long id, Model model){
		model.addAttribute("student", studentService.findOne(id));
		return "student/edit";
	}

	@RequestMapping(value="/student/delete/{id}", method = RequestMethod.GET)
	public String delete(@PathVariable Long id, Model model){
		studentService.delete(id);
		return "redirect:/students";
	}
}
```

11. Application class (Optional step)

We can declare two more annotation in application class. But it is optional.

```
@EnableAutoConfiguration
@ComponentScan("mum.swe.democrud")
```

Our application class is named “DemocrudApplication” which is located inside our main package (mum.swe.democrud).

Now, our application class looks like that.

```
package mum.swe.democrud;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;

@SpringBootApplication
@EnableAutoConfiguration
@ComponentScan("mum.swe.democrud")
public class DemocrudApplication {
```

```
	public static void main(String[] args) {
		SpringApplication.run(DemocrudApplication.class, args);
	}
}
```

12. ServletInitializer class (Optional step)

1. Create a package called “config” inside our main package, “mum.swe.democrud.config”.
2. Create “ServletInitializer” class inside our “config” package.

```
package mum.swe.democrud.config;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.support.SpringBootServletInitializer;

import mum.swe.democrud.DemocrudApplication;


public class ServletInitializer extends SpringBootServletInitializer {

	@Override
	protected SpringApplicationBuilder configure(SpringApplicationBuilder
application) {
		return application.sources(DemocrudApplication.class);
	}

}
```

13. application.properties file

In this file, we can define the configurations of our web application. For example: the configuration of database connection, what kind of template engine we use etc.

```
# ==============================
# DemoCRUD APPLICATION PROPERTIES
# ==============================

# ==============================
# = Thymeleaf configurations (ThymeleafAutoConfiguration)
# ==============================
#LEGACYHTML5
spring.thymeleaf.mode=LEGACYHTML5
spring.thymeleaf.cache=false
spring.thymeleaf.encoding=UTF-8
spring.thymeleaf.dialect=org.thymeleaf.extras.springsecurity4.dialect.SpringSecuri
tyDialect

# ==============================
# = DATA SOURCE
# ==============================
```

```
spring.datasource.url = jdbc:mysql://localhost:3306/democruddb
spring.datasource.username = dbuser
spring.datasource.password = pass2018
spring.datasource.testWhileIdle = true
spring.datasource.validationQuery = SELECT 1
spring.datasource.test-on-borrow=true
spring.datasource.validation-interval=10000
spring.datasource.log-validation-errors=true

# ================================
# = JPA / HIBERNATE
# ================================
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

We can see more configuration from spring documentation (https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html).

For now, the interesting part is that:

```
spring.datasource.url = jdbc:mysql://localhost:3306/democruddb
spring.datasource.username = dbuser
spring.datasource.password = pass2018
```

It says that to spring framework:

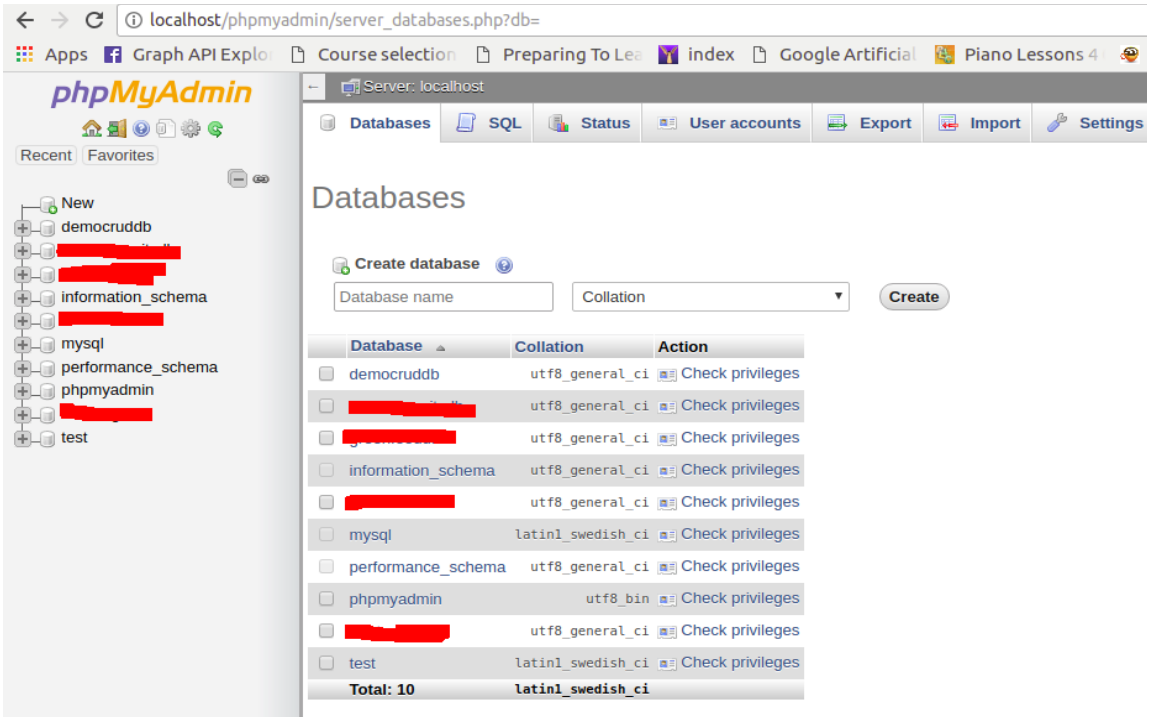
Database name: `democruddb`
Database username: `dbuser`
DB user's password: `pass2018`

14. MySQL Database

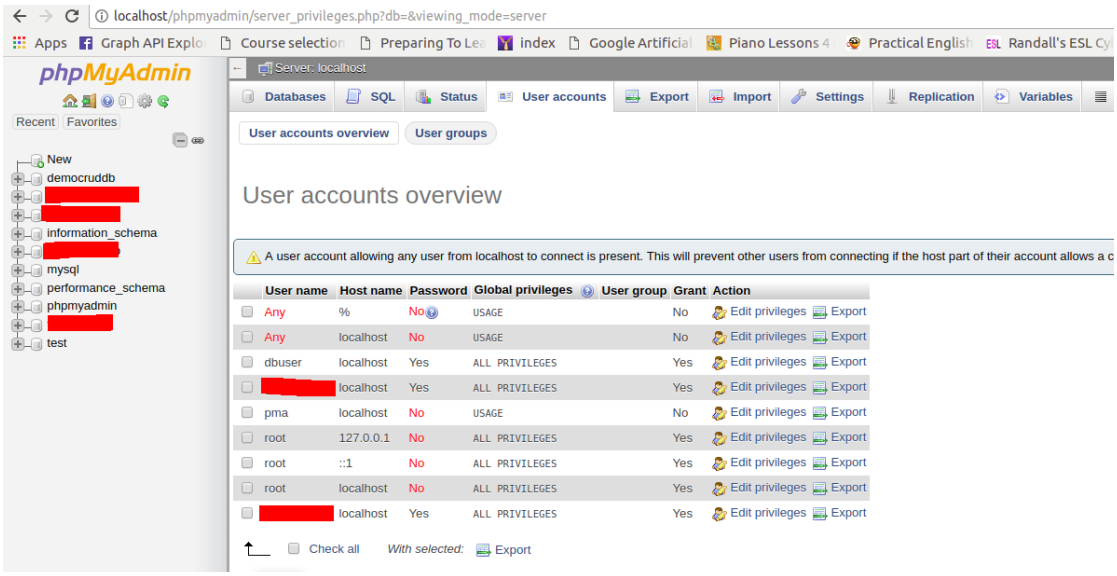
In order to create a database instance, as mentioned early, we have already installed MySQL in our computer.

1. We create database instance, called "democruddb".
2. Create a new database user in MySQL, called "dbuser" and give it full access. Set a password, "pass2018".

If you use phpMyAdmin, "democruddb" will look like this:



“dbuser” will look like this:



15. Views

We will use Thymeleaf as a template engine, Bootstrap as a UI presentation and jQuery as a Javascript library to manipulate HTML document.

Notes:

- *Thymeleaf is a modern server-side Java template engine for both web and standalone environments (http://www.thymeleaf.org/).*
- *Bootstrap is a free and open-source front-end web framework for designing websites and web applications (https://getbootstrap.com/).*
- *jQuery is a cross-platform JavaScript library designed to simplify the client-side scripting of HTML (https://jquery.com/).*

15.1 Static files

Static files are located in "static" folder. We will use Bootstrap and jQuery. You can download Bootstrap from https://getbootstrap.com/ and jQuery from https://jquery.com/.

We will create two folder, "css" and "js" in "resources/static" folder. Copy bootstrap.min.css file to "css" folder. Copy "bootstrap.bundle.min.js" and "jquery.min.js" to js folder.

15.1.1 resources/static/css/homelayout.css

```
#mainpanel { padding: 20px; }
.validation-message { color: red; }
```

15.1.2 resources/static/css/studentlist.css

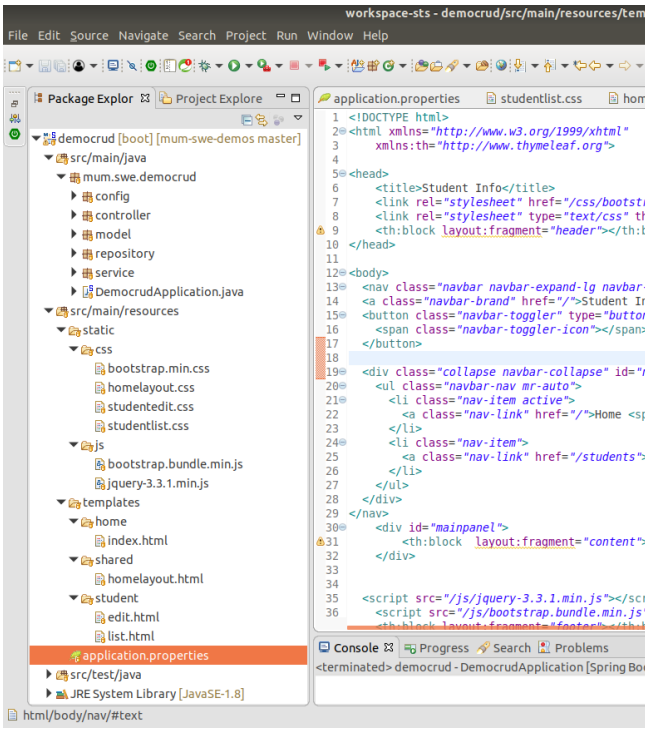
```
#linkBackHome { float: right; }
```

15.1.3 resources/static/css/studentedit.css

```
/* empty file*/
```

15.2 Templates

Template file locates in “resources” folder. We will create a couple of html pages for home page and student information. After creating html view files, our file structure will look like this:



15.2.1 reosources/shared/homelayout.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">

<head>
      <title>Student Info</title>
      <link rel="stylesheet" href="/css/bootstrap.min.css" />
      <link rel="stylesheet" type="text/css" th:href="@{/css/homelayout.css}" />
      <th:block layout:fragment="header"></th:block>
</head>

<body>
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
  <a class="navbar-brand" href="/">Student Info</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-
expanded="false" aria-label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse" id="navbarSupportedContent">
```

```
    <ul class="navbar-nav mr-auto">
      <li class="nav-item active">
        <a class="nav-link" href="/">Home <span class="sr-
only">(current)</span></a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href="/students">Student List</a>
      </li>
    </ul>
  </div>
</nav>
      <div id="mainpanel">
            <th:block  layout:fragment="content"></th:block>
      </div>


  <script src="/js/jquery-3.3.1.min.js"></script>
      <script src="/js/bootstrap.bundle.min.js"></script>
      <th:block layout:fragment="footer"></th:block>
</body>
</html>
```

15.2.2 templates/home/index.html

```
<!DOCTYPE html >
<html layout:decorator="shared/homelayout"
      xmlns:th="http://www.thymeleaf.org">
<body>
      <th:block layout:fragment="header">
            <link rel="stylesheet" type="text/css" th:href="@{/css/home.css}" />
      </th:block>

      <th:block layout:fragment="content">

            <h2>Welcome to home page!</h2>
      </th:block>

      <th:block layout:fragment="footer">
      </th:block>

</body>
</html>
```

15.2.3 templates/student/list.html

```
<!DOCTYPE html >
<html layout:decorator="shared/homelayout"
      xmlns:th="http://www.thymeleaf.org">
<body>
      <th:block layout:fragment="header">
      <link rel="stylesheet" type="text/css" th:href="@{/css/studentlist.css}" />
      </th:block>
```

```
	<th:block layout:fragment="content">
		<h2>Student List</h2> <br/>
		<a href="/student">Create a student</a>
		<a href="/" id="linkBackHome" >Back to home</a><br/><br/>
		<table class="table table-striped">
			<thead>
				<th>First name</th>
				<th>Last name</th>
				<th>Email</th>
				<th>Actions</th>
			</thead>
			<tbody>
				<tr th:each="student : ${students}">
					<td th:text="${student.firstName}"></td>
					<td th:text="${student.lastName}"></td>
					<td th:text="${student.email}"></td>
					<td>
						<a th:href="@{'/student/' + $
{student.id}}">Edit</a> 
						<a th:href="@{'/student/delete/' + $
{student.id}}">Delete</a>
					</td>
				</tr>
			</tbody>
		</table>
	</th:block>

	<th:block layout:fragment="footer">
	</th:block>

</body>
</html>
```

15.2.4 templates/student/edit.html

```
<!DOCTYPE html >
<html layout:decorator="shared/homelayout"
	xmlns:th="http://www.thymeleaf.org">
<body>
	<th:block layout:fragment="header">
		<link rel="stylesheet" type="text/css"
th:href="@{/css/studentedit.css}" />
	</th:block>

	<th:block layout:fragment="content">

		<h2>Student Info</h2> <br/>

		<form class="form-horizontal" action="/student" method="POST"
th:object="${student}">
			<input type="hidden" name="id" th:value="${student.id}" />
		  <div class="form-group">
		    <label class="control-label col-sm-2" for="firstname">First
name:</label>
```

```
                <label th:if="${#fields.hasErrors('firstName')}"
th:errors="*{firstName}"
                                                    class="validation-
message"></label>
                <div class="col-sm-10">
                  <input type="text" class="form-control" id="firstname"
name="firstName"  th:field="*{firstName}">
                </div>
              </div>
              <div class="form-group">
                <label class="control-label col-sm-2" for="lastname">Last
name:</label>
                <label th:if="${#fields.hasErrors('lastName')}"
th:errors="*{lastName}"
                                                    class="validation-
message"></label>
                <div class="col-sm-10">
                  <input type="text" class="form-control" id="lastname"
name="lastName"  th:value="${student.lastName}">
                </div>
              </div>
              <div class="form-group">
                <label class="control-label col-sm-2" for="email">Email:</label>
                 <label th:if="${#fields.hasErrors('email')}" th:errors="*{email}"
                                                    class="validation-
message"></label>

                <div class="col-sm-10">
                  <input type="email" class="form-control" id="email" name="email"
th:value="${student.email}">
                </div>
              </div>
              <div class="form-group">
                <div class="col-sm-offset-2 col-sm-10">
                  <button type="submit" class="btn btn-default">Submit</button>
                </div>
              </div>
            </form>
            <br/>

            <a href="/students">Student List</a>
      </th:block>

      <th:block layout:fragment="footer">
      </th:block>

</body>
</html>
```

16. Thymeleaf Page Layouts

Usually websites share common page components like the header, footer, menu and possibly many more. These page components can be used by the same or different layouts. There are two main styles of organizing layouts in projects: **include style** and **hierarchical style**. Both styles can be easily utilized with Thymeleaf.

- **In include style** pages are built by embedding common page component code directly within each view to generate the final result.

```
<body>
    <div th:insert="footer :: copy">...</div>
</body>
```

- **In hierarchical style**, the templates are usually created with a parent-child relation, from the more general part (layout) to the most specific ones (subviews; e.g. page content). Each component of the template may be included dynamically based on the inclusion and substitution of template **fragments**.

We can see more information from http://www.thymeleaf.org/doc/articles/layouts.html.

16.1 Usage of layout fragments

16.1.1 Example layout page

template/shared/layout.html

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:th="http://www.thymeleaf.org">

<head>
      <title>Student Info</title>
      <link rel="stylesheet" href="/css/bootstrap.min.css" />
      <th:block layout:fragment="header"></th:block>
</head>

<body>

      <div id="mainpanel">
            <th:block  layout:fragment="content">This is a layout page.</th:block>
      </div>


      <script src="/js/jquery-3.3.1.min.js"></script>
      <script src="/js/bootstrap.bundle.min.js"></script>
      <th:block layout:fragment="footer"></th:block>
```

```
</body>
</html>
```

The most important thing about the above example is layout:fragment="content". This is the layout:fragment="content" is the heart of the decorator page (layout). You can also notice, that header and footer are included using Standard Thymeleaf Layout System.

16.1.2 Usage of layout:

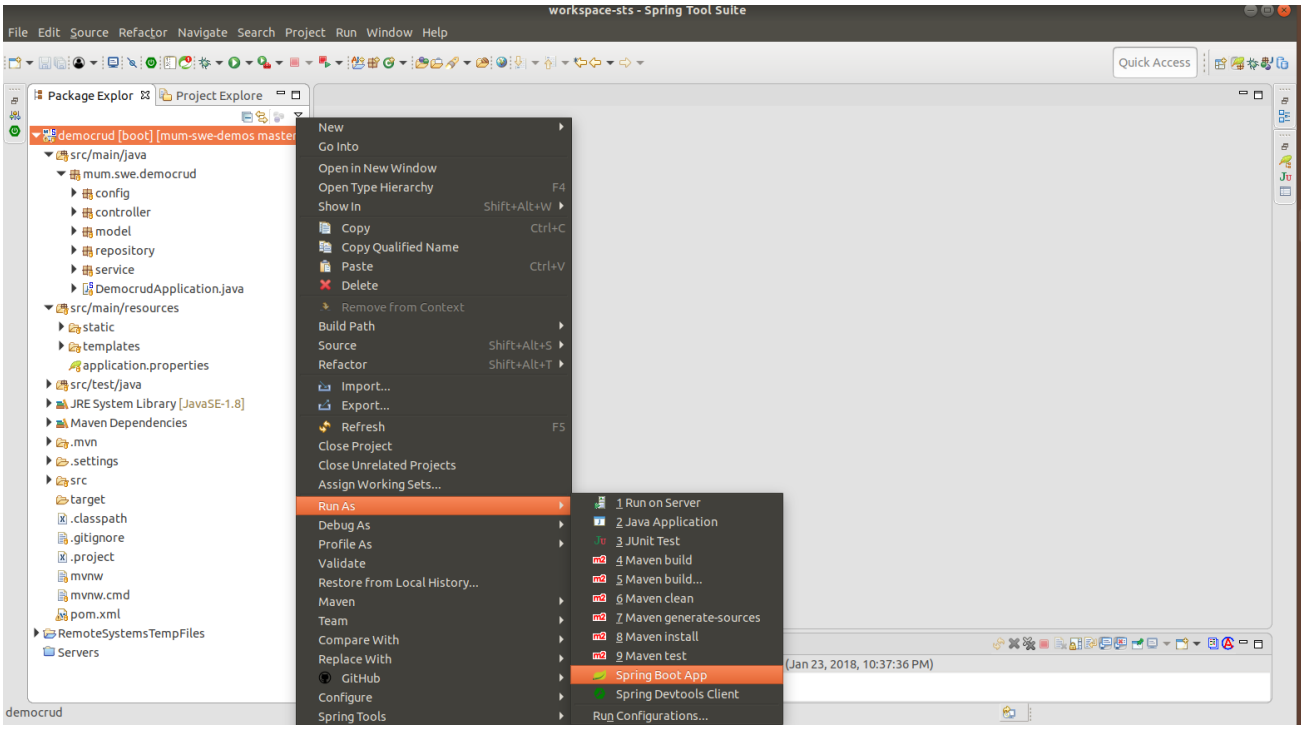
template/student/list.html

```
<!DOCTYPE html >
<html layout:decorator="shared/layout"
      xmlns:th="http://www.thymeleaf.org">
<body>
      <th:block layout:fragment="header">
            <link rel="stylesheet" type="text/css" th:href="@{/css/list.css}" />
      </th:block>

      <th:block layout:fragment="content">
            <h2>This is a list page.</h2>
      </th:block>

      <th:block layout:fragment="footer">
      </th:block>

</body>
</html>
```

In this case, layout.html will be a decorator of list.html file. It means that the content of `<th:block layout:fragment="content">` will be shown in `<div id="mainpanel">` container . Then and, "`This is a layout page.`" content will be replaced by "`<h2>This is a list page.</h2>`".

17. Run the application

In order to run the application, select the project and right click. Select a menu, “Run As” → “Spring Boot App”.



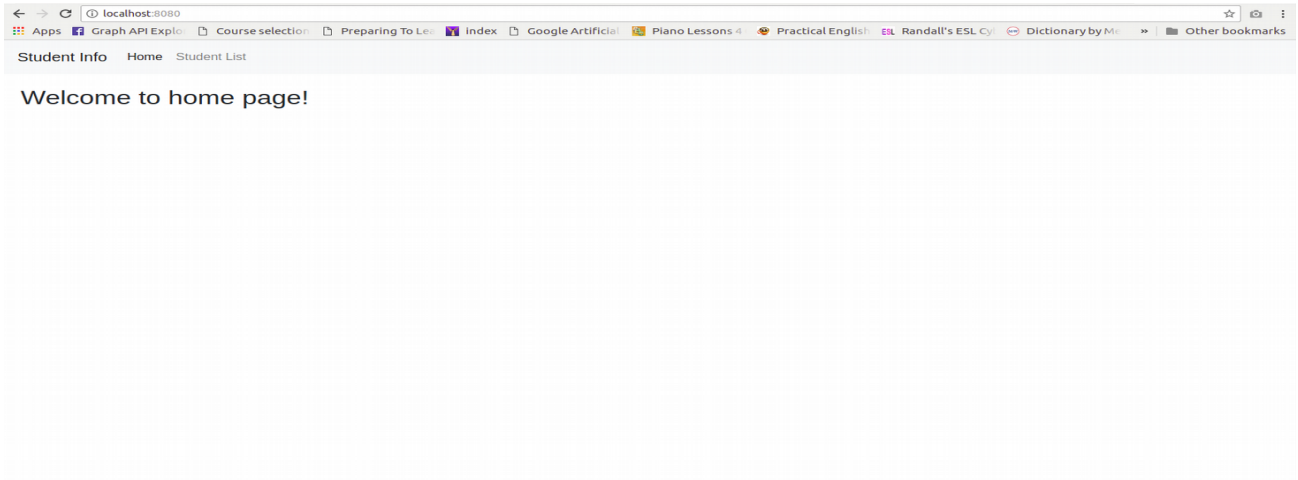
The application will run on port 8080 as default (http://localhost:8080).

Note:

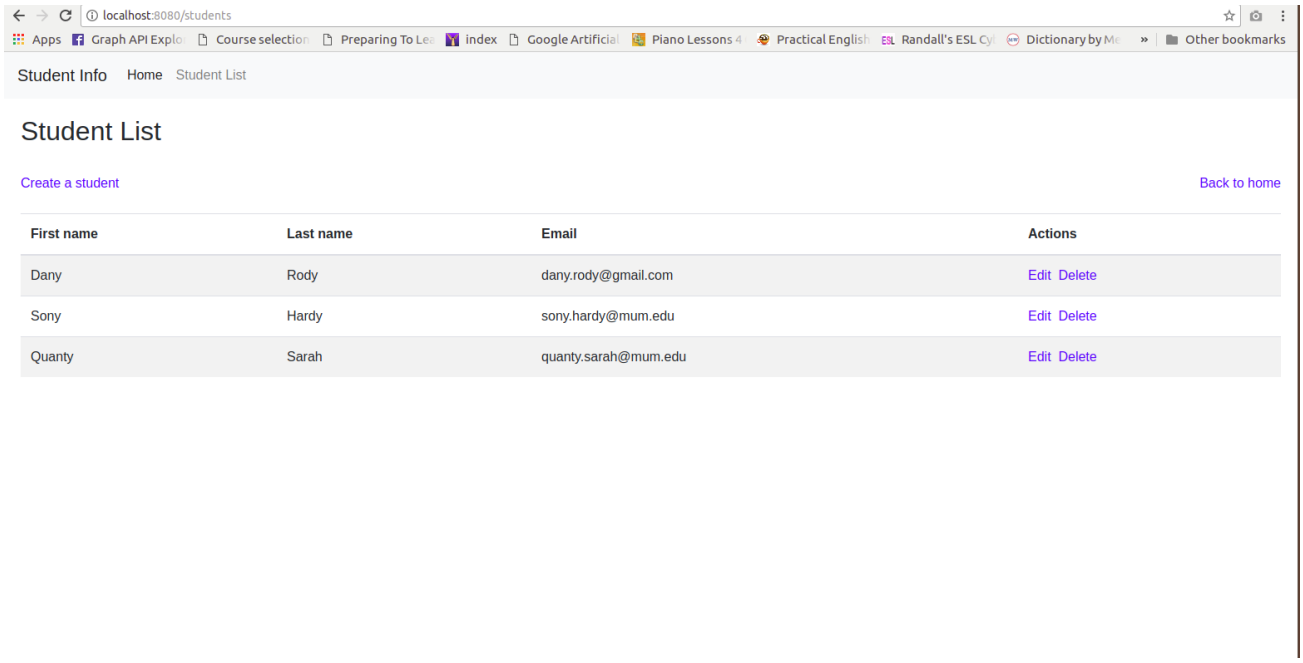
- *Spring Boot will create the database structure if we have provided in the right way our MySQL credentials in the application.properties file.*

17.1 Screenshots

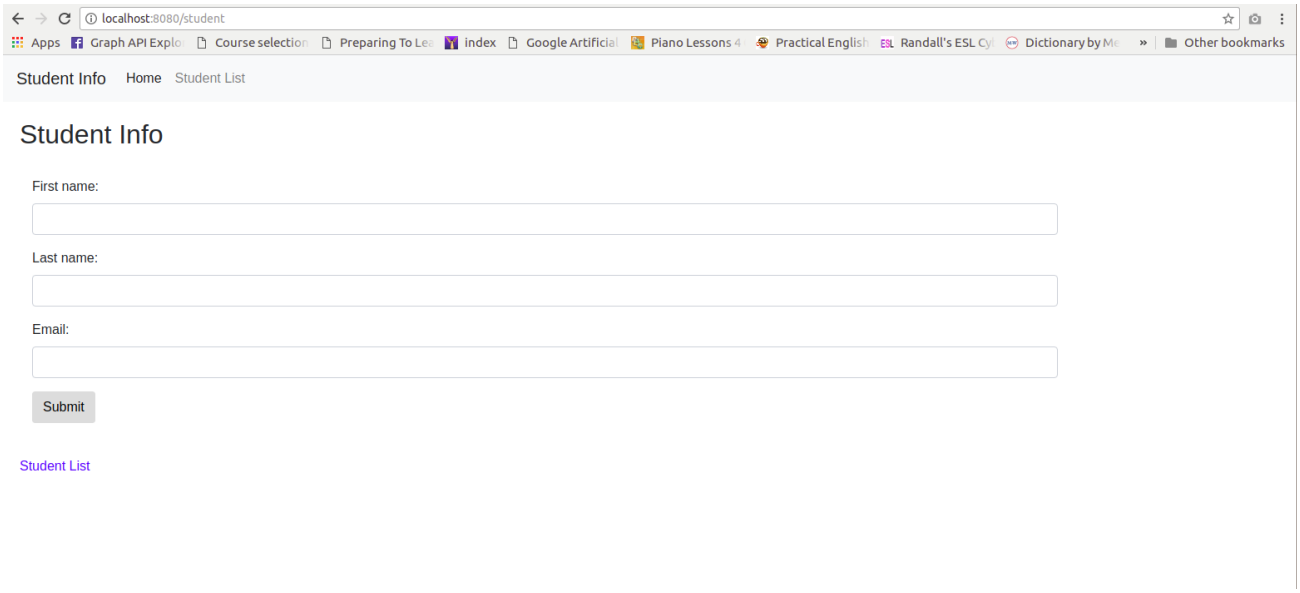
Home page



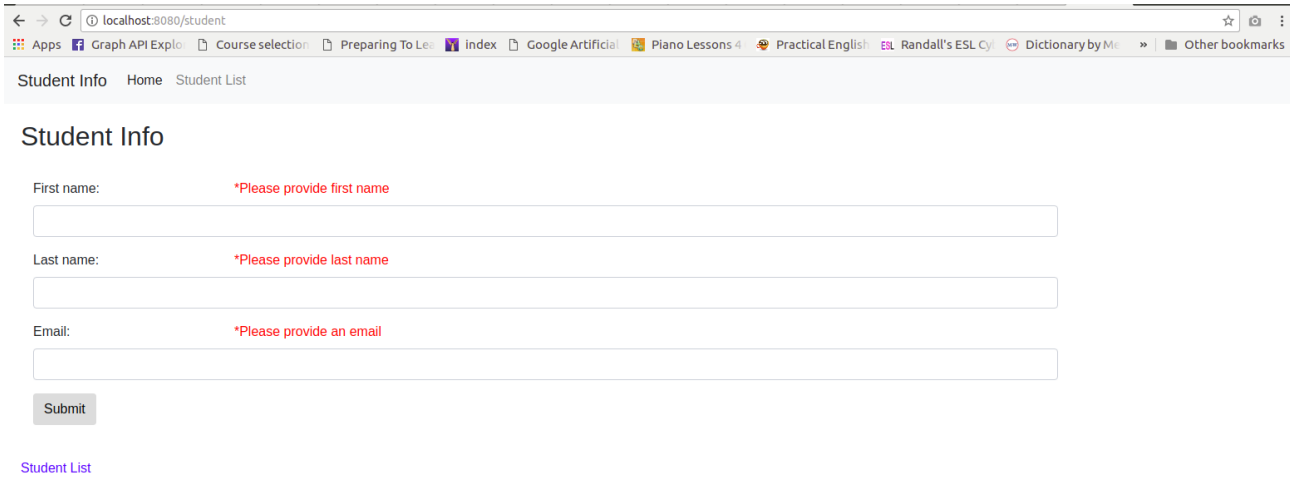
Student List page with data



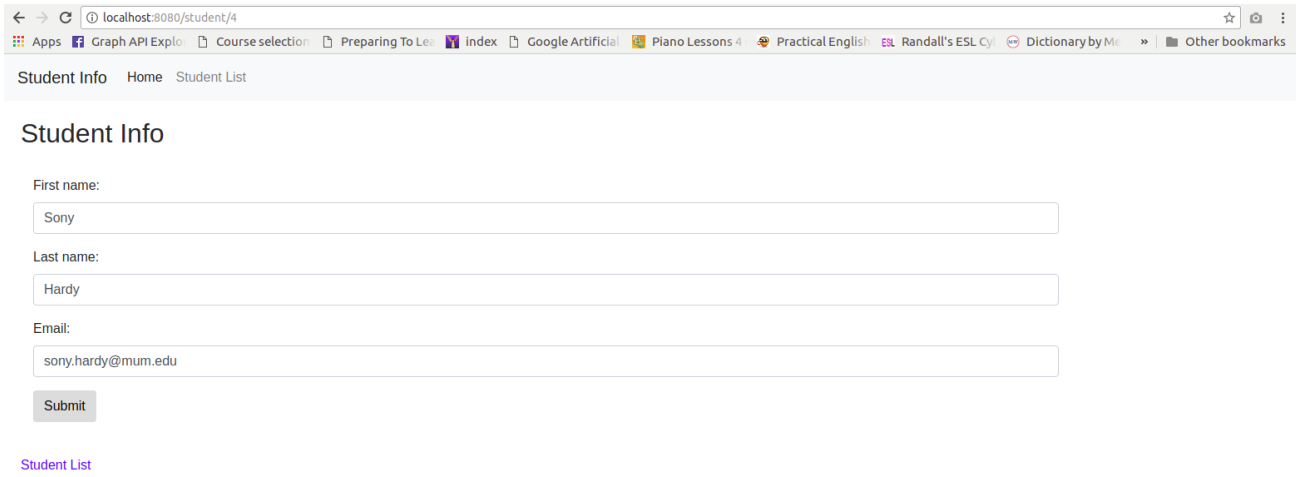
“Create a student” page



Student Info - Validation



Edit Student



18. Summary

Congratulations! You built a simple web application with full CRUD functionality using Spring Boot, Spring MVC, JPA, Hibernate, Thymeleaf and Bootstrap and saved data in MySQL database.

The complete source code of the project is available in github at the following URL: https://github.com/bbatjargal/mum-swe-demos/tree/master/democrud