# **Introduction to Web Development**

* Task 1: [Install Gradle](https://docs.gradle.org/current/userguide/installation.html)
* [Task 2: Web Application setup with Spring Initializr](https://docs.google.com/document/d/1_uszW91jt9bCq8JVuia_bOfKdRcUZK-41k1ZdZ32QLc/edit)

# **Task 2: Web Application setup with Spring Initializr**

1. To check if Gradle is properly installed type gradle -v in the terminal

Table

Description automatically generated

1. **Go to** [Spring Intializr](https://start.spring.io/) to generate a Spring Boot project. Create a new project with the configurations shown below. Click on “GENERATE” button to download the project package.

Graphical user interface, text, application

Description automatically generated

1. Unzip the package, and openthe project using IntelliJ Idea. The following project structure will be shown.

It will take some time to download the relevant packages for your gradle project.

**Folder Structure:**

Text

Description automatically generated

1. Right click on the ‘Web Project” folder, and select “Open Module Setting”

A screenshot of a computer

Description automatically generated with medium confidence

1. Go to Project, select “Download JDK” from the Project SDK section. (This will take 3-5 mins to download the latest JDK (Java development Kit)

*Graphical user interface, text

Description automatically generated*

1. Once downloaded, select the latest JDK (openjdk-16) version from the Project SDK section.

*Graphical user interface, text

Description automatically generated*

1. Go to ‘build.gradle’ file, comment off the implementation for spring-boot-start-data-jpa under the dependencies section.

Text

Description automatically generated

1. Right click on the ‘WebProjectApplication’ file and run the file.

*Graphical user interface

Description automatically generated*

*Text

Description automatically generated*

1. **Open** your browser with the following url: <http://localhost:8080>.

* You should see the error message
* **Do not** worry about the error message; in the following section you will learn about defining the mapping and adding behavior to the web Application.

Graphical user interface, text, application, email

Description automatically generated

## **Useful Links:**

* [What is a Web Framework?](https://www.goodfirms.co/glossary/web-framework/)
* [Frontend vs Backend](https://www.geeksforgeeks.org/frontend-vs-backend/)
* [MVC Architecture](https://www.w3schools.in/mvc-architecture/)
* [MVC Design Pattern](https://www.geeksforgeeks.org/mvc-design-pattern/)
* [What is Gradle](https://docs.gradle.org/current/userguide/what_is_gradle.html) Link 1
* [What is Gradle](https://www.simplilearn.com/tutorials/gradle-tutorial/what-is-gradle) Link 2
* [ANT vs Maven vs Gradle](https://www.baeldung.com/ant-maven-gradle)
* [Spring Boot Introduction](https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm)
* [Building an application with Spring Boot](https://spring.io/guides/gs/spring-boot/)
* [Spring Boot Documentatio](https://docs.spring.io/spring-boot/docs/2.2.5.RELEASE/reference/html/documentation-overview.html#boot-documentation-first-steps)n
* [Spring Boot Gradle plugin reference guide](https://docs.spring.io/spring-boot/docs/2.4.5/gradle-plugin/reference/htmlsingle/)
* [Build a Complete RESTful Service using SpringBoot 2.0 and Gradle](https://namila007.medium.com/build-a-complete-restful-service-using-springboot-2-0-and-gradle-e9e381a114d8)
* [Kotlin vs Groovy](https://www.folio3.com/mobile/blog/kotlin-vs-groovy/#:~:text=Kotlin%20is%20an%20open%2Dsource,Java%20Virtual%20Machine%20(JVM).&text=Groovy%20is%20an%20object%2Doriented,based%20on%20the%20Java%20platform.)

# Web Development with Spring Boot 1 - Dependencies Injection

* [Task 1: Setup URL Routing and Templates for Web Application](https://docs.google.com/document/d/13k0AXUOaHop5R-rX4gUh4m76jcN0PnqZGNbFI1_Dizg/edit#heading=h.mb6ohdrt6jxi)

# **Task 1: Setup URL Routing and Templates for Web Application**

|  |
| --- |
| Download the ‘WebProject\_Start’ provided, and copy the relevant files and folders to the WebProject that you have setup with Spring Boot Framework.  **Folder Structure:**  Graphical user interface, application  Description automatically generated |
| The web application is based on Spring MVC. As a result, you need to configure Spring MVC and set up view controllers to expose these templates.  Create a new package folder ‘security’ in the WebProject folder. Add a ‘MvcConfig’ class file to the folder.    Graphical user interface  Description automatically generated with medium confidence |
| The following listing shows a class that configures Spring MVC in the application:  Text  Description automatically generated  Reference Link:  <https://spring.io/guides/gs/securing-web/> |
| Open your browser with the following url: <http://localhost:8080>.  You should see the index.html page in the homepage.  Graphical user interface, website  Description automatically generated  **Note that all pages’ links are not ready yet~~~~~~~~~~~** |
| [Thymeleaf](http://www.thymeleaf.org/) is a Java template engine for processing and creating HTML, XML, JavaScript, CSS, and text. You are able to make use of [Thymeleaf](http://www.thymeleaf.org/) template engine to Create a separate navbar.html and footer.html to be inserted in all your main html files (e.g. index, aboutus, product, productform pages)  Reference Link:  <https://www.baeldung.com/thymeleaf-in-spring-mvc>    Create a new directory ‘fragments’ under the ‘templates’ folder. Create 2 new html5 files and name them as ‘navbar.html’ and ‘footer.html’ separately.  Graphical user interface, text  Description automatically generated with medium confidence |
| In the navbar.html, replace the codes using the following example:  <!DOCTYPE html>  <html lang="en" xmlns:th="https://www.thymeleaf.org/">  <head>     <meta charset="UTF-8">     <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />     <title>Title</title>  </head>  <body>     <nav th:fragment="navbar"  class="navbar navbar-expand-lg navbar-dark bg-dark ">         <a class="navbar-brand" href="https://www.generation.org/">BrandLogo</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">                                               <!-- margin right - push items to the right -->                 <li class="nav-item active"><a class="nav-link"  th:href="@{index}"> HOME </a></li>                 <li class="nav-item "><a class="nav-link"  th:href="@{aboutus}"> ABOUT </a></li>                 <li class="nav-item"><a class="nav-link" th:href="@{products}"> PRODUCTS </a></li>                 <li class="nav-item"><a class="nav-link" h th:href="@{productform}"> PRODUCTS FORM </a></li>                 <li class="nav-item"><a class="nav-link" href="#" data-toggle="modal" data-target="#exampleModal"> CONTACT </a></li>             </ul>             <form class="form-inline my-2 my-lg-0">                 <input class="form-control mr-sm-2" type="search" placeholder="Search" aria-label="Search">                 <button class="btn btn-outline-success my-2 my-sm-0" type="submit">Search</button>             </form>         </div>     </nav>     <div th:fragment="menuModel" class="modal fade" id="exampleModal"  tabindex="-1" role="dialog" aria-labelledby="exampleModalLabel" aria-hidden="true">         <div class="modal-dialog">             <div class="modal-content">                 <div class="modal-header">                     <p>Contact details</p>                 </div>                 <div class="modal-body">                     <p>For more enquiries, please contact +65-97887434 or email at contactsg@generation.org</p>                 </div>                 <div class="modal-footer">                     <a class="btn btn-primary" data-dismiss="modal">Close</a>                 </div>             </div>         </div>     </div>  </body>  </html> |
| In the footer.html, replace the codes using the following example:  <!DOCTYPE html>  <html lang="en" xmlns:th="https://www.thymeleaf.org/">  <head>     <meta charset="UTF-8">     <title>Title</title>  </head>  <body>     <div th:fragment="footer" class="container my-5">         <div class="navbar navbar-expand-lg navbar-dark bg-dark fixed-bottom">             <div class="container">                 <p class="navbar-text pull-left">© 2021 Generation: You Employed, Inc. | Terms of Use | Privacy Policy</p>                 <a href="#" class="navbar-btn btn-danger btn pull-right">Sitemap</a>             </div>         </div>  </div>  </body>  </html> |
| From the index.html. aboutus.html, products.html and productform.html, remove the codes for navigation, modal popup for contact, and footer.  Include the thymeleaf engine in all your html pages.  <html lang="en" xmlns:th="https://www.thymeleaf.org/">  Replace navigation and modal popup with the following codes:  <div th:insert="fragments/navbar.html :: navbar"></div>  <div th:insert="fragments/navbar.html :: menuModel"></div>    Replace footer with the following codes:  <div th:insert="fragments/footer.html :: footer"></div> |
| Open your browser with the following url: <http://localhost:8080>.  You should see the index.html page in the homepage, and able to link to the other pages from the navigation. |

* [Task 2: Dependencies Injection using Spring Framework](https://docs.google.com/document/d/1NHWMaiQwaBwF35YNTRjqcclD-dZol6PUx1JHwxqr2zc/edit)

# **Task 2: Dependencies Injection using Spring Framework**

## **Implement a Service Layer (With basic setup of Controller layer and Repository layer)**

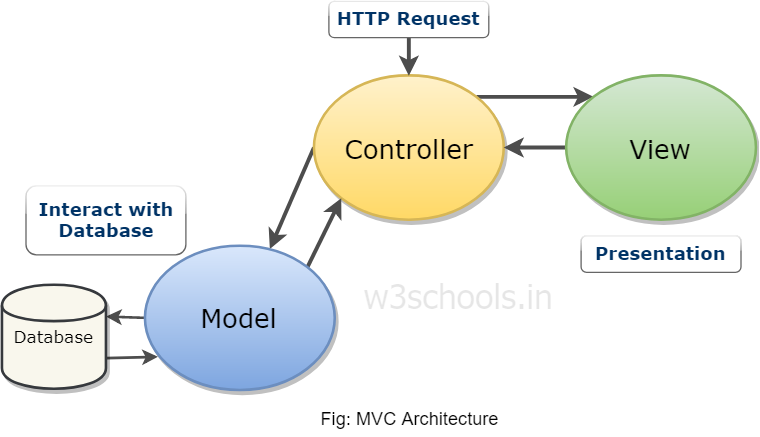
|  |
| --- |
| 1. **Create an Item class in the Repository->Entity folder.** (We will edit and add in more codes for the Item class in the next lesson)   Graphical user interface, application  Description automatically generated |
| 1. **Create an ItemDto class in the Controller->dto folder.** (DTO stands for Data Transfer Object, we will cover that in the next few lesson)   Graphical user interface  Description automatically generated |
| 1. Populate the **ItemDTO Class in the Controller layer** with the appropriate attributes, Constructor, Getter and Setter methods for the Product Web Application. The ItemDTO Class will interface with the Client (Browser) for any HTTP request (e.g. GET, POST, PUT, DELETE methods) and wired to the ItemService Class to perform getItem, addItem, findItem, updateItem and deleteItem.   Text  Description automatically generated |
| 1. Populate the **Item Class in the Repository layer** with the appropriate attributes, Constructor, Getter and Setter methods for the Product Web Application. Item Class will get the request from the ItemDTO Class from the Controller layer to perform the necessary CRUD operation to the database.   Text  Description automatically generated  Text  Description automatically generated |
| 1. **Create ItemService Interface and ItemServiceMySQL class in the Service folder.** (DTO stands for Data Transfer Object, we will cover that in the next few lesson)   A picture containing text  Description automatically generated |
| 1. **From the ItemService Interface**, provide the interface of the services that are necessary to perform for the Item (e.g. list all item, save item, delete item, and find item by Id)   Text  Description automatically generated |
| 1. From the ItemServiceMySQL Class, setup the various services that are necessary to perform for the Item (e.g. list all item, save item, delete item, and find item by Id) through the Repository layer for the CRUD operation to the database (Will implement the constructor and methods in the next lesson).   Text  Description automatically generated |
| Congratulations! You are ready for the next step on Spring Data JPA! |

## Useful Links:

* [Creating a template using Thymeleaf](https://education.launchcode.org/java-web-development/chapters/thymeleaf-views/create-template.html)
* [Secure a web application](https://spring.io/guides/gs/securing-web/)
* [Serve Static Resources with Spring](https://www.baeldung.com/spring-mvc-static-resources)
* [Decoupled Architecture](https://www.cloudamqp.com/blog/why-is-application-decoupling-a-good-thing.html)
* [Dependencies Injection](https://www.vogella.com/tutorials/DependencyInjection/article.html)
* [Dependencies Injection with Java code](https://www.codejava.net/coding/what-is-dependency-injection-with-java-code-example)
* [Spring Boot Service Components](https://www.tutorialspoint.com/spring_boot/spring_boot_service_components.htm) (@Service and @Autowired annotation)
* [Java Bean](https://www.javatpoint.com/java-bean)
* [Spring Boot website](https://spring.io/projects/spring-boot)

**Spring Boot Architecture and Workflow**

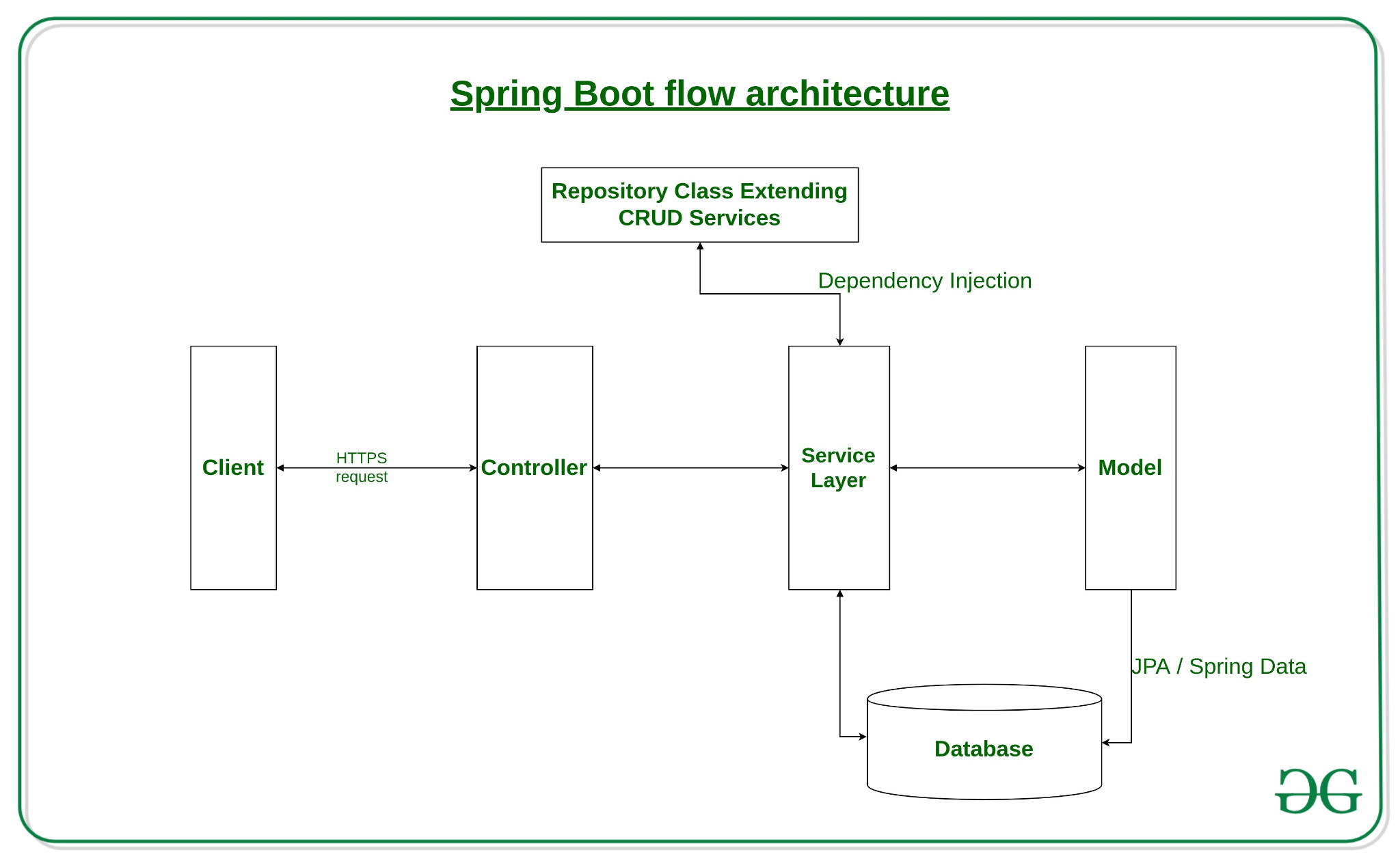
MVC is abbreviated as Model View Controller is a design pattern created for developing applications, specifically web applications.



With the emergence of the MVC model, creation of application takes different aspects individually into consideration. These aspects of the application are:

* UI Logic
* Input logic
* Business Logic

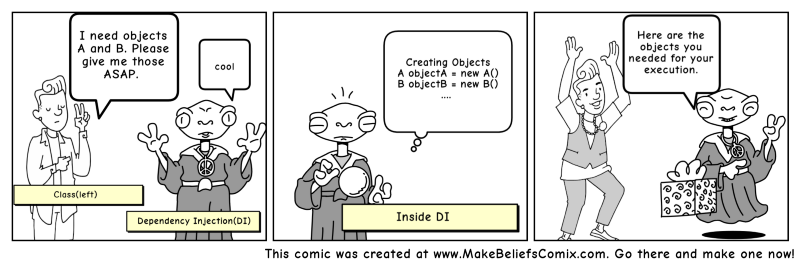
Reference Link: <https://www.w3schools.in/mvc-architecture/>



* **Repository:** Creating a data access layer needs just a repository class instead which is implementing CRUD (Create, Read, Update, Delete) operation containing class. Spring Boot provides an interface calledCrudRepositorythat contains methods for CRUD operations.
* A client makes the https request(PUT/GET).
* **Controller:** The request then goes to the controller and the controller mapped with that route as that of request handles it, and calls the service logic (business logic).
* **Service:** Business logic is performed in the service layer which might be performing the logic on the data from the database which is mapped through JPA (Java Persistence API) with model/entity class.
* Finally, a page (with the data) is returned in the response if no error occurred.
* Reference Link: <https://www.geeksforgeeks.org/introduction-to-spring-boot/>

**Dependency Injection (DI)**

In [software engineering](https://en.wikipedia.org/wiki/Software_engineering), **dependency injection** is a technique whereby one object (or static method) supplies the dependencies of another object. A dependency is an object that can be used (a [service](https://en.wikipedia.org/wiki/Service_(systems_architecture))).



Reference Link: <https://www.freecodecamp.org/news/a-quick-intro-to-dependency-injection-what-it-is-and-when-to-use-it-7578c84fa88f/>

**Spring @Service Annotation**

Spring @Service annotation is used with classes that provide some business functionalities. Spring context will autodetect these classes when annotation-based configuration and classpath scanning is used. @Service holds the logic for the behavior of your interface.

**Spring @Autowired Annotation**

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.

<https://www.javatpoint.com/autowiring-in-spring>

# **Web Development with Spring Boot 2 - Spring with JPA**

* [Task 1: Setup Product Database](https://docs.google.com/document/d/1on6IfyiEKw-IOwQ9LRJX0Pi-mVsTXXz3kkbWfhUdZxw/edit)

# **Task 1: Setup Product Database**

Manually Create Connection Account:

CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin';

GRANT ALL PRIVILEGES ON \* . \* TO 'admin'@'localhost';

ALTER USER 'admin’@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'passw0rd';

|  |
| --- |
| In the SQL Workbench, open the product\_table.mwb in the EER Diagram  Graphical user interface, application  Description automatically generated |
| Export the SQL script using the Forward Engineer SQL Script function.  Graphical user interface, text, application  Description automatically generated  Create a new database connection with the name ‘productList’, set Username to ‘admin’ and give a password “passw0rd’  Graphical user interface, application  Description automatically generated |
| Create the database schemas with the SQL script with the setup of the relevant tables.  Graphical user interface, text, application  Description automatically generated |
| Right click on Item Table, and select Table Data Import Wizard. Import ‘productList.csv’ file provided  Graphical user interface, text, application, chat or text message  Description automatically generated  Graphical user interface, text, application  Description automatically generated |

* [Task 2: Using Spring Data JPA to access MySQL](https://docs.google.com/document/d/1Z18G7fTzCLWu9N_cLfY2OpbCLs0U1cEG2RsWm9XDVU4/edit#heading=h.9qptvn12qkwh)

# **Task 2: Using Spring Data JPA to access MySQL**

|  |
| --- |
| In build.gradle, uncomment the Data JPA starter package, and add in the jdbc starter and mysql connector packages  Text  Description automatically generated |
| In application.properties, add the following setting:  spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl  spring.datasource.url=jdbc:mysql://${MYSQL\_HOST:localhost}:3306/itemsdb  spring.datasource.username=admin  spring.datasource.password=passw0rd  server.port = 8080 |
| Create an ‘ItemRespository’ interface in the repository folder.  Graphical user interface, text, application, chat or text message  Description automatically generated |
| Add in the following codes and rebuild the Project folder.  Text  Description automatically generated |
| Open the ‘Item’ Class. Import the relevant library and include the Annotation to the methods.  < MAPPING > Identify ID = PRIMARY KEY …….@Entity @Id @GeneratedValue  Text  Description automatically generated |
| In ItemServiceMySQL Class, include the ItemRepository interface to perform CRUD operation to the database.  Text  Description automatically generated |

## **Useful Links:**

* [Java Persistence API](https://www.vogella.com/tutorials/JavaPersistenceAPI/article.html)
* [ORM Overview](https://www.tutorialspoint.com/hibernate/orm_overview.htm)
* [Spring Data JPA](https://www.journaldev.com/17034/spring-data-jpa)
* [Accessing Data with Spring Data JPA and MySQL](https://attacomsian.com/blog/accessing-data-spring-data-jpa-mysql)
* [Accessing data with MySQL](https://spring.io/guides/gs/accessing-data-mysql/)
* [CRUDRepository Package](https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html)

# **Web Development with Spring Boot 3 - REST API with Spring Boot**

* [Task 1: Create REST API with Spring Boot](https://docs.google.com/document/d/1ZE0LJNlFpA9oX07TDyeaB8WRjqwmcD40LNUDvKtqK7o/edit)

# **Task 1: Create REST API with Spring Boot**

|  |
| --- |
| Create an ‘ItemController’ Class in the controller folder.  A screenshot of a computer  Description automatically generated with low confidence |
| Implement the Controller that maps to ItemRepository to display all records from the database. **(Note: add @CrossOrigin for all Mapping annotations)**  Text  Description automatically generated |
| Goto Intelli J , select MyWebProjectionApplication class and rebuild. After rebuild is done. Then rerun MyWebProjectionApplication.  . \_\_\_\_ \_ \_\_ \_ \_  /\\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \  ( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \  \\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )  ' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_\\_\_, | / / / /  =========|\_|==============|\_\_\_/=/\_/\_/\_/  :: Spring Boot :: (v2.5.0-RC1)  2021-05-25 15:00:56.006 INFO 41154 --- [ main] o.g.M.MyWebProjectApplication : Starting MyWebProjectApplication using Java 16.0.1 on Remys-MacBook-Air.local with PID 41154 (/Users/remylim/Downloads/MyWebProject/build/classes/java/main started by remylim in /Users/remylim/Downloads/MyWebProject)  2021-05-25 15:00:56.018 INFO 41154 --- [ main] o.g.M.MyWebProjectApplication : No active profile set, falling back to default profiles: default  2021-05-25 15:01:01.147 INFO 41154 --- [ main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.  2021-05-25 15:01:02.200 INFO 41154 --- [ main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 1004 ms. Found 1 JPA repository interfaces.  2021-05-25 15:01:05.992 INFO 41154 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)  2021-05-25 15:01:06.047 INFO 41154 --- [ main] o.apache.catalina.core.StandardService : Starting service [Tomcat]  2021-05-25 15:01:06.047 INFO 41154 --- [ main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.45]  2021-05-25 15:01:06.772 INFO 41154 --- [ main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext  2021-05-25 15:01:06.772 INFO 41154 --- [ main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 10230 ms  2021-05-25 15:01:08.422 INFO 41154 --- [ main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...  2021-05-25 15:01:12.361 INFO 41154 --- [ main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.  2021-05-25 15:01:12.705 INFO 41154 --- [ main] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]  2021-05-25 15:01:13.413 INFO 41154 --- [ main] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.4.30.Final  2021-05-25 15:01:15.224 INFO 41154 --- [ main] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}  2021-05-25 15:01:16.042 INFO 41154 --- [ main] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.dialect.MySQL8Dialect  2021-05-25 15:01:19.902 INFO 41154 --- [ main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]  2021-05-25 15:01:19.941 INFO 41154 --- [ main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'  2021-05-25 15:01:21.544 WARN 41154 --- [ main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning  2021-05-25 15:01:22.202 INFO 41154 --- [ main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'  2021-05-25 15:01:22.860 INFO 41154 --- [ main] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page template: index  2021-05-25 15:01:24.373 INFO 41154 --- [ main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''  2021-05-25 15:01:24.444 INFO 41154 --- [ main] o.g.M.MyWebProjectApplication : Started MyWebProjectApplication in 31.806 seconds (JVM running for 37.685)  2021-05-25 15:01:24.447 INFO 41154 --- [ main] o.s.b.a.ApplicationAvailability : Application availability state LivenessState changed to CORRECT  2021-05-25 15:01:24.457 INFO 41154 --- [ main] o.s.b.a.ApplicationAvailability : Application availability state ReadinessState changed to ACCEPTING\_TRAFFIC  2021-05-25 15:01:41.218 INFO 41154 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'  2021-05-25 15:01:41.220 INFO 41154 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'  2021-05-25 15:01:41.224 INFO 41154 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 4 ms  Open <http://localhost:8080/item>, all records in the itemdb table from the database will be shown.  Text, letter  Description automatically generated |

|  |
| --- |
| Edit the Controller Class to ItemService interface for REST API implementation.  Text  Description automatically generated  Text  Description automatically generated  Text  Description automatically generated |
| Launch Postman to test the REST API.  GET METHOD TO GET ALL ITEMS  http://localhost:8080/item/all  Graphical user interface, text, email  Description automatically generated |
| GET METHOD TO GET AN ITEM WITH THE ID  <http://localhost:8080/item/2>  Graphical user interface, text, application, email  Description automatically generated |
| PUT METHOD TO UPDATE AN ITEM WITH THE ID  <http://localhost:8080/item/2>  {     "name": "new",     "description": "new",     "imageUrl": "images/t-shirt1.jpg",     "style": "new",     "price": 15.0  }    Graphical user interface, text, application, email  Description automatically generated |
| POST METHOD TO ADD AN ITEM  <http://localhost:8080/item/add>  {     "name": "new2",     "description": "new2",     "imageUrl": "images/t-shirt1.jpg",     "style": "new2",     "price": 30.0  }  Graphical user interface, text, application, email  Description automatically generated |
| DELETE METHOD TO DELETE AN ITEM  <http://localhost:8080/item/3>  Graphical user interface, text, application, Teams  Description automatically generated |
| Check the itemdb table for the changes.  Graphical user interface, text, application  Description automatically generated |

* [Task 2: Display and add items from Web Application](https://docs.google.com/document/d/1rWdoyf6gfqFe_B_IYfvtmsqycrM4dRhM7l85l7a1LAc/edit)

# **Task 2: Display and add items from Web Application**

## **Display items to Product.html**

|  |
| --- |
| Edit the **‘productListing.js”** as follow:  Graphical user interface, text  Description automatically generated |
| Edit the **‘productController.js”** to fetch the product list from the database and display on the product.html.  Text  Description automatically generated  Text  Description automatically generated  Text  Description automatically generated |
| Launch the **Product.html** page to show the products from the database.  <http://localhost:8080/products>  Graphical user interface, application, Word, Teams  Description automatically generated |
|  |

## **Add items from ProductForm.html (with upload of Product images to the local server)**

|  |
| --- |
| In the **productForm.html**, edit the Form tag to include enctype attribute for file uploading function. For all input tags, add in the name attribute to pass the input values to the backend for storing in the database. |
| Create a **‘productForm.js”** under the ‘static/js’ folder and add in the js code as follow: |
| Edit the **‘productController.js”** using FormData to fetch the form values and pass the values to the backend for processing. |
| Open the **‘ItemController.java’** file, edit the POST mapping to the following. |
| In the **‘item.java’** from the Entity folder, edit getImageUrl method. |
| Create a ‘**component**’ folder, and create a ‘**FileUploadUtil.class’** in the folder. |
| In the ‘**FileUploadUtil.class’** , add in the following code: |
| In the ‘MvcConfig.class’ file, include the following codes: |
| In the ‘WebProject’ folder, add the ‘productImages/images’ directory to store all the Product images for the web application. |
| Launch the **ProductForm.html** page to add products to the database.  <http://localhost:8080/productform> |

## **Useful Links:**

* [What is REST](https://www.codecademy.com/articles/what-is-rest)
* [Building REST services with Spring](https://spring.io/guides/tutorials/rest/)
* [HTML Form enctype attribute](https://www.w3schools.com/tags/att_form_enctype.asp)
* [Upload Images using Spring Boot](https://www.codejava.net/frameworks/spring-boot/spring-boot-file-upload-tutorial)
* [Using FormData](https://developer.mozilla.org/en-US/docs/Web/API/FormData/Using_FormData_Objects)
* [@RequestMapping](https://www.baeldung.com/spring-requestmapping)
* [Spring MVC - Get, Post, Put Delete](https://www.appsdeveloperblog.com/spring-mvc-postmapping-getmapping-putmapping-deletemapping/)
* [@RequestParam](https://www.baeldung.com/spring-request-param)
* [@RequestBody](https://www.baeldung.com/spring-request-response-body)
* [@PathVariable](https://www.baeldung.com/spring-pathvariable)
* [CROS support in Spring Framework](https://spring.io/blog/2015/06/08/cors-support-in-spring-framework)

# **SECURE API : Spring Boot Security Form Authentication with JDBC and MySQL**

### **Step by Step Instruction:**

Setup MySQL database

Ensure MySQL connection is connected, click connection(ProductList)

CREATE TABLE IF NOT EXISTS ‘itemsdb’,`users` (

  `user\_id` int(11) NOT NULL AUTO\_INCREMENT,

  `username` varchar(45) NOT NULL,

  `password` varchar(64) NOT NULL,

  `role` varchar(45) NOT NULL,

  `enabled` tinyint(4) DEFAULT NULL,

  PRIMARY KEY (`user\_id`)

);

INSERT INTO `users` (`username`,`password`,`role`,`enabled`)

VALUES ('namhm',

'$2a$10$XptfskLsT1l/bRTLRiiCgejHqOpgXFreUnNUa35gJdCr2v2QbVFzu',

'ROLE\_USER', 1);

INSERT INTO `users` (`username`,`password`,`role`,`enabled`)

VALUES ('admin',

'$2a$10$zxvEq8XzYEYtNjbkRsJEbukHeRx3XS6MDXHMu8cNuNsRfZJWwswDy',

'ROLE\_ADMIN', 1);

Graphical user interface, text, application

Description automatically generated

[Spring Boot Security Form Authentication with JDBC and MySQL](https://www.codejava.net/frameworks/spring-boot/form-authentication-with-jdbc-and-mysql)

### **Additional Setup:**

|  |
| --- |
| Add in the following dependencies to the build.gradle file  implementation 'org.springframework.boot:spring-boot-starter-security'  implementation 'org.thymeleaf.extras:thymeleaf-extras-springsecurity5' |
| In MvcConfig, add in the following to the view controller.  registry.addViewController("/login").setViewName("login");  registry.addViewController("/logout").setViewName("index"); |
| With reference to our Product web application:   * Create a WebSecurityConfig Class in the security folder * use the following configuration to configure our form-based authentication.   @CrossOrigin  @Override  protected void configure(HttpSecurity http) throws Exception {     http.csrf().disable();     http.formLogin().loginPage("/login");     http.formLogin()             .defaultSuccessUrl("/productsform");     http.logout()             .logoutSuccessUrl("/index");     http.authorizeRequests()             .antMatchers("/", "/products", "/aboutus").permitAll()             .antMatchers("/productform/\*\*").hasRole("ADMIN")             .and()             .formLogin()             .loginPage("/login").permitAll()             .and()             .logout().permitAll();  } |
| Edit the login.html to include the code to map the login/logout to the view controller.  Text  Description automatically generated |
| Edit the navbar.html to show Login/Logout in the menu items  Text  Description automatically generated |

### **Form-Based authentication**

Form-Based authentication is a way in which a user's authentication is done by login form. This form is built-in and provided by spring security framework.

The HttpSecurity class provide a method formLogin() which is responsible for rendering login form and validate user credentials.

### **WebSecurityConfigurerAdapter**

WebSecurityConfigurerAdapter is a convenience class that allows customization to both WebSecurity and HttpSecurity. We can extend WebSecurityConfigurerAdapter multiple times (in distinct objects) to replicate the behavior of having multiple http elements.

[[Link](https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/config/annotation/web/configuration/WebSecurityConfigurerAdapter.html)]

**Useful Links:**

* [Spring Security Form Login](https://www.baeldung.com/spring-security-login)
* [BCrypt Hashing](https://auth0.com/blog/hashing-in-action-understanding-bcrypt/)
* [SHA Hashing](https://www.baeldung.com/sha-256-hashing-java)

## [**AuthenticationManagerBuilder**](https://docs.spring.io/spring-security/site/docs/4.2.20.RELEASE/apidocs/org/springframework/security/config/annotation/authentication/builders/AuthenticationManagerBuilder.html#jdbcAuthentication--)

Unit Testing Web Components

*testing with the Mockito framework for writing JUnit software tests in Java*

Useful Links:

* [JUnit Test](https://howtodoinjava.com/junit-4/)
* [JUnit Assert methods](https://howtodoinjava.com/junit-4/)
* [JUnit Assert APIs](https://junit.org/junit4/javadoc/4.12/org/junit/Assert.html)
* [JUnit Assert Examples](https://www.guru99.com/junit-assert.html)
* [Concept of Mocking](https://dzone.com/articles/the-concept-mocking)
* [Unit Testing with Mockito](https://www.vogella.com/tutorials/Mockito/article.html)
* [Mockito Documentation](https://site.mockito.org/)
* [Mockito Quick Guide](https://www.tutorialspoint.com/mockito/mockito_quick_guide.htm)
* [JUnit + Mockito example](https://automationrhapsody.com/mock-junit-tests-mockito-example/)
* [@TestInstance Annotation](https://www.baeldung.com/junit-testinstance-annotation)

|  |
| --- |
| Include the 2 testing packages to the dependencies |
| In the test folder->WebProject, create 2 test classes (1) ItemServiceMySQLTest and (2) ItemControllerTest |
| Text  Description automatically generated  Graphical user interface  Description automatically generated |
| Text  Description automatically generated  Text  Description automatically generated |
|  |