Step Guide

Step 1. Download zip file (Green button with name “Code” and on the bottom of dropdown is “Download ZIP” option) - from GitHub (https://github.com/HerNidza/PrimeHolding) -> Unzip that file you will be presented with SQL script “PrimeHoldingCRUDsqlScript” and One more zipped file that you should unpack that contains app (name of file that contains app is: “prime-holding-crud-nikola-stankov”)

Step 2. Import SQL script into MySQL workbench, name of the script is “PrimeHoldingCRUDsqlScript”. Once imported, you should run it. Database tables and data will be imported.

Step 3. Importing project (in my case IntelliJ). File -> Open -> Choose unzipped file from end of step 1 (prime-holding-crud-nikola-stankov).

Step 4. Once imported -> under src/main/resources/application.properties - it is configured to be run on port: 8080. If for some reason 8080 is taken on your PC, feel free to change it to port that works for you (just swap number in application.properties under “server.port” that works for you).

Step 5. We are still in “application. properties” there is a line that starts with “spring.datasource.url” most of it probably stays the same **you should check port for MySQL if it is not 3306 configure it to yours**. **Be sure to use MySQL because dependency in pom.xml is set for MySQL.** In next two lines, spring.datasource.username and spring.datasource.password – should be according to your database.

Step 6. Running configuration. Under
src/main/java/com.nikola.primeholdingcrud/PrimeHoldingCrudApplication

You should do a right click of the mouse and select “ Run PrimeHoldingCrudApplication “. If the port in **Step 3 is 8080** in your browser you should place this URL http://localhost:8080/ if you changed port change numbers in a link to your selected port. Once placed in a browser that link should automatically redirect you to **employees/list** from there on you are fully in the application!

Structure of Application

I believe that structure that I have used is most commonly referred as the “Layered Architecture” or “Layered Architecture Pattern”. This architecture separates the application into distinct layers, where each layer has a specific responsibility and communicates with the layers above and below it. The entity layer defines data model, the DAO layer provides CRUD (Create, Read, Update, Delete) plus some some personal operations provided by me for the data model. Service layers provide the business logic that operates the data. Lastly, the Controller layer handles the user request and coordinates the interactions between layers. (Some extra pointer controller can return HTTP response typically in the form of view in our case - or JSON data)

Description (This is unordered I will place here additional functionalities)

For this project I have used Spring Boot, Hibernate, Spring Data JPA, MySql, Hibernate Validator.

- In **DAO** package each interface I have created extends (when the interface is implementing another interface we use “extends” not like usual “implements”). Once we do that it provides for as a set of generic CRUD operations for interacting with the database. Jpa Interface provides a set of methods that allow us to interact with data sources in a **simple** way. Without having boilerplate code. Example: we can use “findAll()” to retrieve all entities of a given type…. Also, in

EmployeeRepository I have created custom queries. **One is for Displaying 5 employees who completed the largest number of tasks in the past month name is: "findTop5AssigneesByTaskCount"** (**Query is created with the goal of calculating 30 days before – starting now. I am using localdate.now() as parameter. You can play with the data but you have to set the data directly in MySQL because I have added validation @Future for dueDate reason for this is it makes no sense to place due date in the past..**). **One more additional query that is simple yet useful is calculating average salaries of employees.**

- In **Entity** package I have created 2 mandatory entities and one extra "EmployeeFeedback". Each entity is marked with annotation @Entity to mark Java class as JPA entity – we are telling JPA to treat that class as persistent entity which can be mapped to a database table. In some entities, I have places validation, for example: @Email -> validates that annotated property is a valid email address, @Size, @PositiveNumber,@Past… **EXTRA: I have created custom validation for Employees when entering mobile phone to have prefix number +359 - (Bulgaria 😉)**
- **Service** package acts as an intermediary between presentation layer and the data access layer – providing a higher layer interface that encapsulates the details of how data is stored and retrieved. It consist of sets of service classes and interfaces which define the operations that can be performed on the data – Each service class or interface is responsible for a specific set of related operations. By encapsulating business logic in the service layer application becomes more flexible and easier to maintain. **EXTRA: I have created a search option for each of entities Employee can search by: fullName and Email, Task by: title and description and EmployeeFeedback by: positive and negative feedback.** Business logic is here, but it comes from DAO extending JpaRepository it provides (one of many methods) for us findBy()… method that we can tailor to our needs. Example findByFullName…
- **Controller** package in my case is responsible for handling HTTP requests from the user and returning appropriate response. It acts as an interface between user and business logic of the app. For example when user navigates to the URL associated with specific route in our app, the controller receives that request and retrieves necessary data from the Model and then renders an appropriate view using in our case Thymeleaf. **EXTRA: in StatisticsController I have implemented some statistics and also 3. Mandatory functionalities in the assignment.**
- **Thymeleaf** functionalities that I have used: **th:each ->** is used to iterate over a collection and apply the specified markup to each element in the collection, **th:field ->** is used to bind and input field to a model attribute or property, **th:text ->** is used to bind the text content of an element to a mode attribute or property, **th:object ->** is used to specify the object to be used as a current processing context in the template, **th:action ->** is used to specify the URL that the form should be submitted to, **th:if ->** is a conditional attribute that allows us to conditionally render or hide HTML elements (My case in the application is used for rendering and hiding validation errors when they happen)

P.S.

I have also implemented other way of updating and adding Tasks with a drop-down list so there is no mistake when updating and creating tasks. Two pages are kind a short 😉 I also left comments in code for extra stuff that I did.