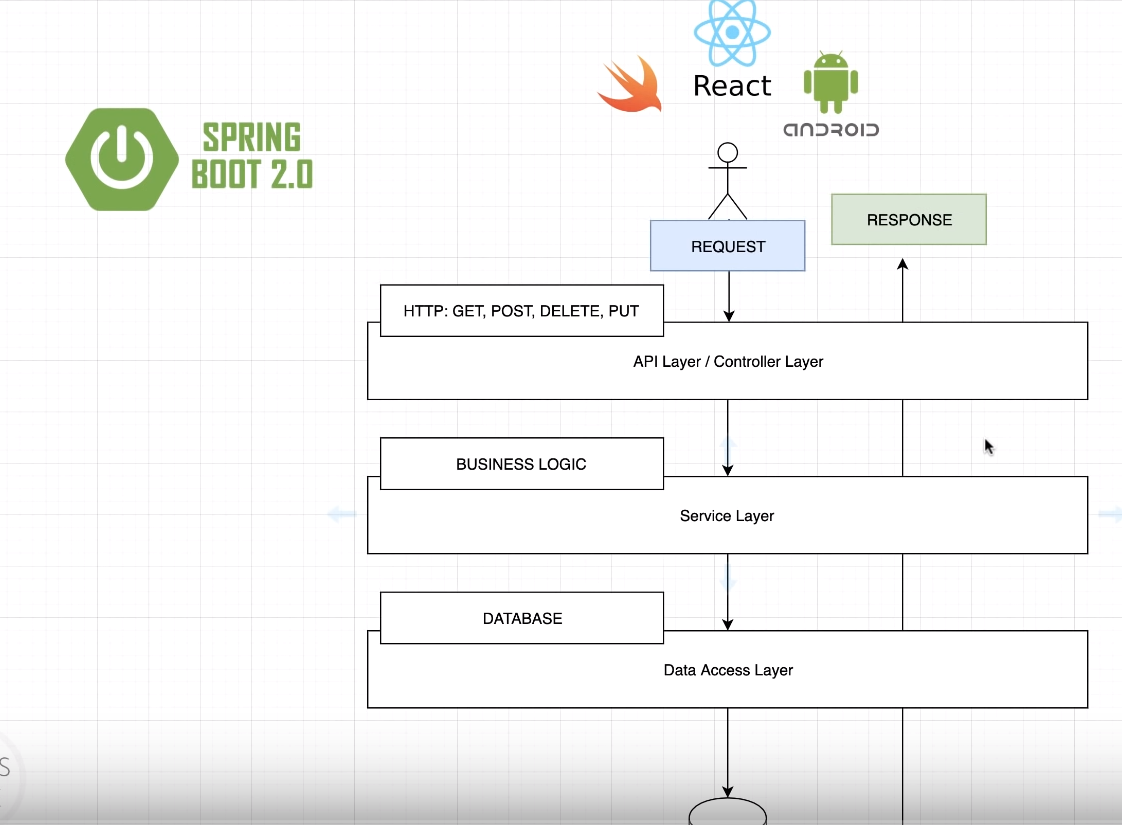
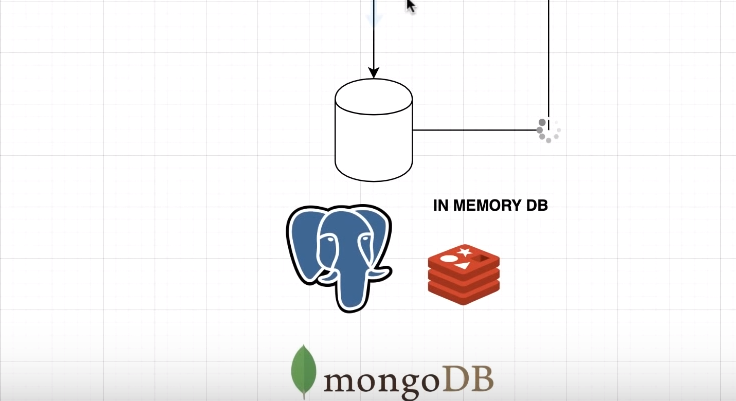
Spring Boot

Course: <https://youtu.be/vtPkZShrvXQ>





* **API Layer/Controller Layer** is where we can receive the request.
* **Service Layer** is where we handle the request and all business logic that you may have within your application.
* **Data Access Layer** is where you use DBs to perform some various credit operations.

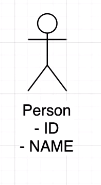
**Go for the spring initializer :** <https://start.spring.io/>

After initializing, we will start from bottom to top. We will create multiple packages as follows:

**(we are starting from Data access layer to all the way up)**

* com.freeCodeCamp.demo.api : where the API will live.
* com.freeCodeCamp.demo.model
* com.freeCodeCamp.demo.service
* com.freeCodeCamp.demo.dao

These packages represent the previous diagram. We will start with the model. Our model will be a person with these properties (ID & Name). So, we created **Person.class** in **model**.





Now, we will create interface called **PersonDao.I** in **dao** inwhich we will define the operations allowed or the actual contract for anyone wishes to implement this interface. One more advantage, we can later use the dependency injection to switch between implementations with just one line of code.



Now, we will create a class called “FakePersonDataAccessService” to implement our interface methods in **dao**.package.



Now, we will define the actual service in **service package** called **PersonService**

**@@******

@21:40

Now, after finishing the service let’s go and implement the actual API in **api package** class called **PersonController.**

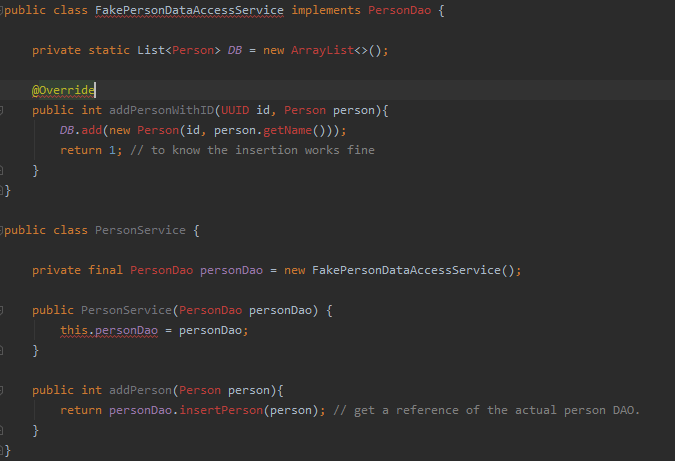
****

@22:50

**Up to this point,** we are having a normal JAVA classes. We are not using Spring Framework. Now let’s configure everything to start using Spring boot framework.

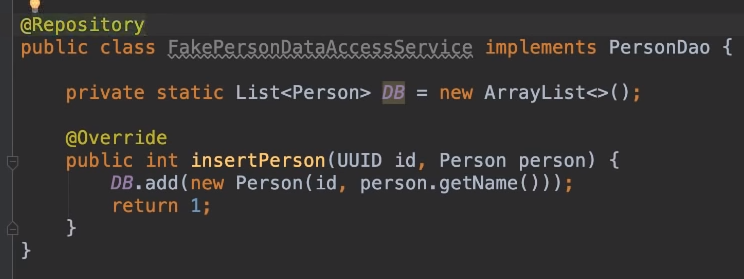
* Let’s start with actual person Dao (**FakePersonDataAccessService**). >> @Repository
* Service Class (PersonService) >> @Service
* **How to use the dependency injection? (My Explanation)**

In normal JAVA way, how can we link between these two classes (PersonService & FakePersonDataAccessService )



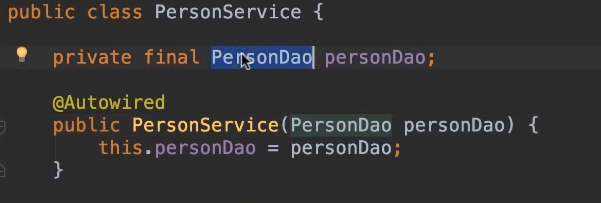
So, it’s hard coded. If there are any new changes you will change it manually in the code. However, this is how can we do it in spring boot.

***Step 01***: you need to tell spring that this class needs to be instantiated as Bean, so we can inject it later in other classes. You can tell spring that this class needs to be instantiated as Bean by giving it @Component or any other specific annotations such as @Service @Repositry.

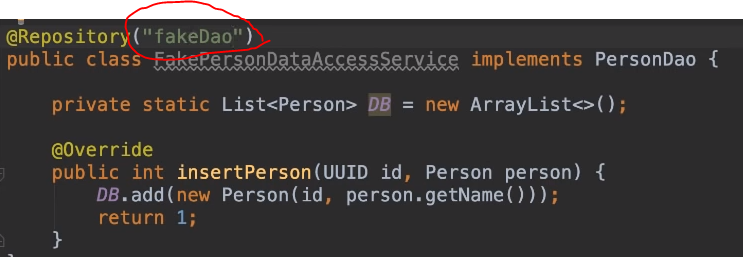


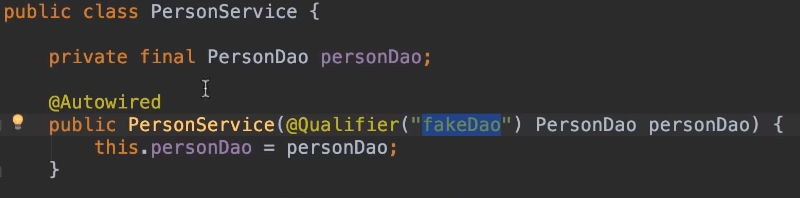
***Step02:*** Go to the other class that you need to use the instantiated class in it then reference it to the instantiated class. For example, in (**PersonService**) we referenced it to **PersonDao** interface which has child class (**FakePersonDataAcc…**).

***Step 03 :*** via the constructor, inject the reference to it then @Autowired it.



Notice, that because we have only one class that implements the interface it will understand it directly. But, if you have multiple implementations then you need to use @Qualifier as follows.





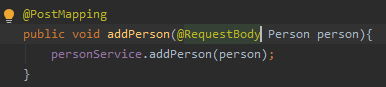
====================================

API Layer where we use HTTP methods in the controller class.

@26:30

* **Implementing POST.**

Now, we will give the controller class **@RestController.** Then, we will give the addPerson method **@PostMapping.**

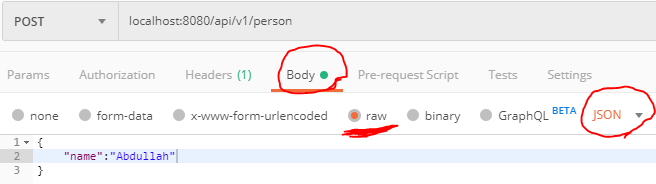


**@RequestBody**

It means, take the request body then shovel it inside the object person.

**@RequestMapping** : before your class then define your URLs inside it.

Now, since we don’t have a client such as React App or Android App, we will use postman as our client. We will post to this URL : localhost:8080/api/v1/person As shown below. Notice, we did not send id since it’s generated already in the service.



But remember, we need to post these properties (name and id) but we should first define them in Person class as show here. **@JsonProperty** means take this property and convert it to JAVA class. Also, the **@JsonProperty** annotation is used to map property names with JSON keys during serialization and deserialization. It takes a String attribute that specifies the name that should be mapped to the field during serialization.

