# Thesis Ride share web application

#### Introduction

Ride-share (Carpooling) is a brand new type of travelling ,the mean idea of it is to share car journeys in order to reduce each person’s cost for the trip . it is more eco-friendly as well since it’s a good way to use up the full seating capacity of a car , which will left unused if only the driver is using the car. Authorities often encourage ride-share , as it can reduce air pollution , traffic congestion on the roads , and even the needs for parking lot , which is quite important for major big cities .

So how does carpooling goes ? Well usually drivers will post their ride and passengers will join part of or whole journey based on their needs , and share the travel cost . Drivers and passengers will contact each other to make an appointment for pick up places and may negotiate special need (such as large luggage) and price .

In order to post or share the carpooling information , we need such a platform . There are quite many platform for it , website , carpooling agency , pick-up point , carpooling groups and so on . With the development of mobile and website , the carpooling app is becoming more and more popular. After learning three years in University of Debrecen , it would be my pleasure to make this carpooling web application .

But why web application ? Instead of desktop or mobile apps ? Well the web application is more portable , light . You can open it any where on any phone . All you need is a web broswer , which is in every phone and computer. Apart from that , as web apps are cloud based , it is more easier to sync all the data and info between all the devices .

There are several reason that I choose to develop a web application instead of mobile or desktop applications :

1. No need for installation

Since the application is web based , you can do not need to download any installation pack . All you need is a web broswer and a URL .

1. Automatic updates

Having to manually download and install updates on a regular basis is a hassle. There are no two ways about it. Even when the program automatically downloads them for you, you still need to approve it, wait a bit, and relaunch the app. It sounds like a small thing, but you’d be surprised how much time is saved when the updates are automatic and whenever you open an application it’s always the latest stable version

Cross-Platform Availability

For the vast majority of web applications, the only prerequisite is internet access. They aren’t reliant on the hardware and system specifications to run. As a result, you can launch them from whatever device or platform that has a web browser. Since the components that are responsible for the app functionality are on the server, it doesn’t matter whether you launch it from Windows, Mac, Linux, or anything else.

Mobile Access

Related to the previous point, platform-independence also means mobility. This means most web applications can also run on mobile devices. Depending on the nature and the functionality of the program, it can sometimes be limited due to the size of the screen or less precise navigation. For example, a complex multi-track audio mastering web studio would be better suited for a screen larger than a phone. But the vast majority of enterprise web solutions function perfectly well regardless of the system. This allows you to take your work anywhere and still keep your finger on the pulse of your business processes.

Light on Computer Resources

Web services eat up significantly less processing power. Certainly, your browser still runs on your computer, and the more tabs you have open, the more memory is used. However, it’s nowhere near comparable to the desktop apps in that regard. Not everyone has a powerful PC. Not being able to do your job because of the technical limitations of your machine is frustrating. Web applications work pretty much the same regardless of how expensive your processor is.

why web app

//https://digitalskynet.com/blog/Desktop-App-vs-Web-App-Comparative-Analysis

For web application , we usually devide it into front end and back end . The back end is more about dealing with database , handling data ,server , and implementing main logic and then provide the apis to front end , while front end is more about user interface and user interaction , reciving and sending data from back end .

//Front end back end picture here

As for our tech trace , I will use java and springboot frame work for my back end and typescript and react frame work for my front end .

## Backend part

#### PostgresSQL

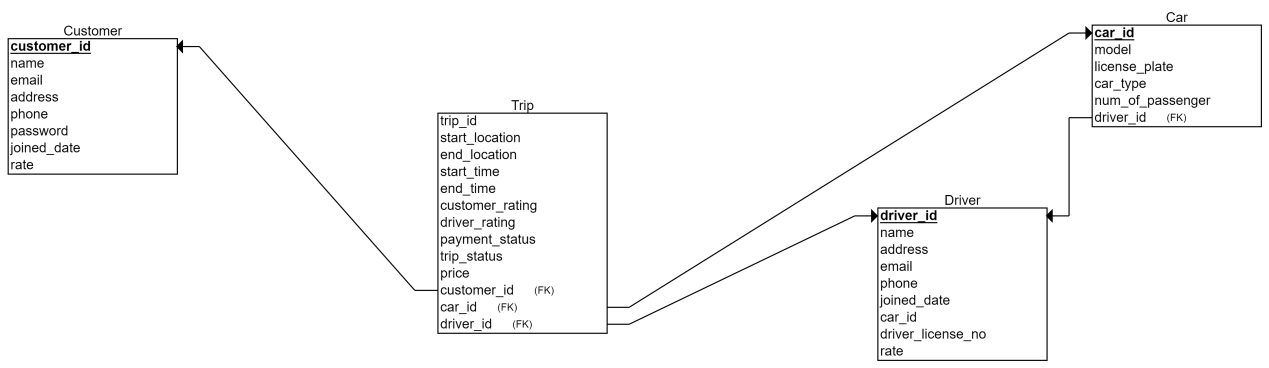
###### Introduction

PostgresSQL is a relational database management system (RDBMS) , which means it stores data in the form of tables. It was built to be feature-rich, extendable and standards-compliant. In the past, Postgres performance was more balanced - reads were generally slower than MySQL, but it was capable of writing large amounts of data more efficiently, and it handled concurrency better.

###### Database in carpooling

As we are building our carpooling application , we need a place to store our data, that place is database

//db design picture

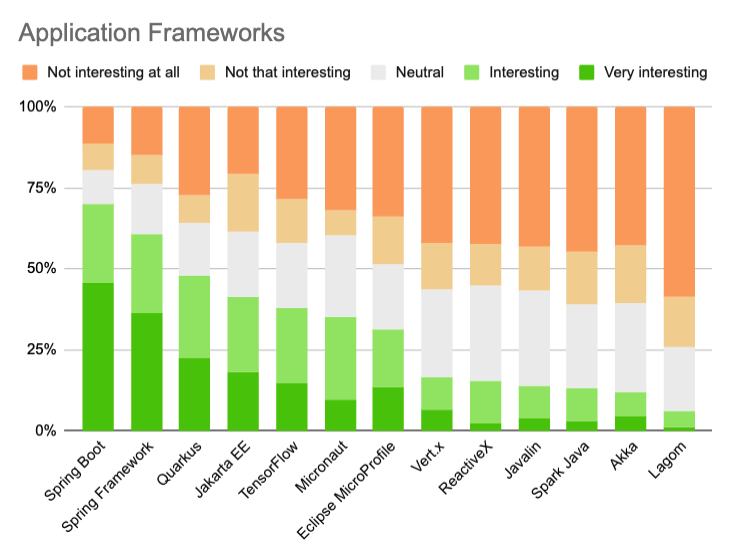
here

#### Springboot

###### What is spring boot

Spring boot is a famous java application frame work ,Spring Boot is an open source, microservice-based Java web framework. The Spring Boot framework creates a fully production-ready environment that is completely configurable using its prebuilt code within its codebase. The microservice architecture provides developers with a fully enclosed application, including embedded application servers.

//https://www.jrebel.com/blog/what-is-spring-boot



Spring boot is dominating in java

//https://jaxenter.com/java-trends-top-10-frameworks-2020-168867.html

###### Building Entity In Spring Boot

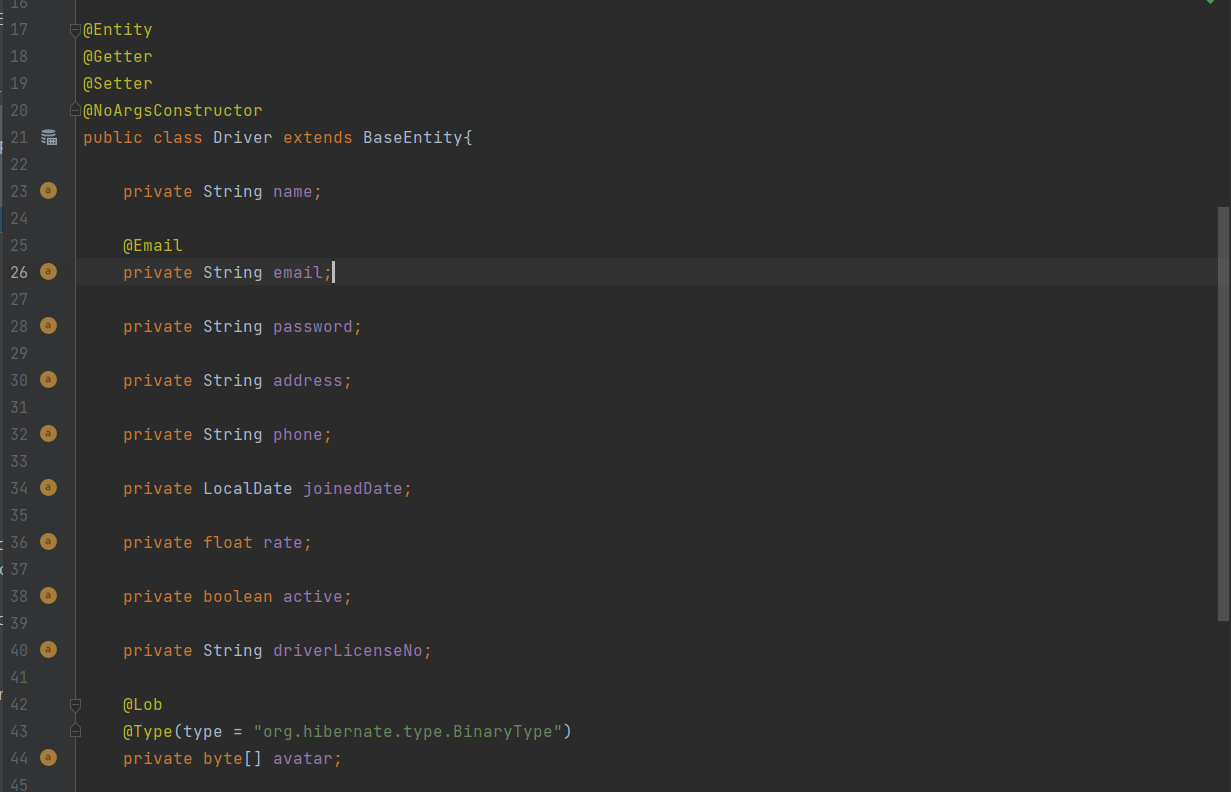
After we have our database design , we need to link / map it to spring boot using java so that we can manipulate data directly using java , where we will need our entity class

Base Entity



For every entity they do needs something in common : an id and version . Id is the primary key for each table or entity , version is for concurrency control .

Since they do share some attributes , it would be wise to put them into a abstract class and let the other entity classes to extend it . So that they will have a auto increased id for their primary key .

After the preparation , we are ready to map our entities . Let’s take our Driver entity as an example :

Firstly we need to extend our Base Entity class and mark @Entity annotation to tell spring boot it’s an database entity .

Attributes

For attributes we have :

name : driver’s name ,

email with @Email annotation in order to validate the string is a valid email form .

password , address ,

driver’s phone for contact ,

joinedDate : Driver’s join date in the community ,

Rate :driver’s rate from customer ,

active : when driver registered himself with an email , the active will be set to false since we haven’t confirm the email yet ,and driver cannot login when active if false . it will be set to true when the email is confirmed,

driverLicenseNo : confirm the driver is able to drive,

avatar : drivers can upload their avatar to the database .Note that the pictures in database in an byte array .

Relations

After the attributes we have to handle our relations between tables , here is an example in driver class that how we can handle it in spring boot :

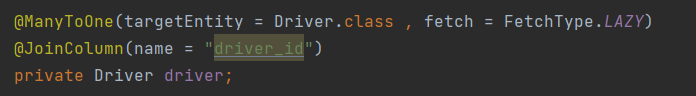
For driver table we have two relations :

1. One driver can have multiple cars
2. One driver can publish multiple trips

They are both one to many relations , but how do we handle it in spring boot ?



As in the picture , we annotate them with @OneToMany annotation ,since it’s one to many , we represent them as a list here. As we actually store driver id in Car and Public trip table , in those classes , we need to do the following :



We annotate driver entity with @ManyToOne and specify the target entity in Car Entity class and PublicTrip entity class respectively.

###### Map Entities into database using Liquibase

For now we have our database relational model and entities classes ready , but inside our database is till empty . We need to map all our design into database tables . We can surly use CREAT TABLE statements in SQL to build such table , but each time we flush our database we need to type or past those SQL again , is there a way to do it automatically ?

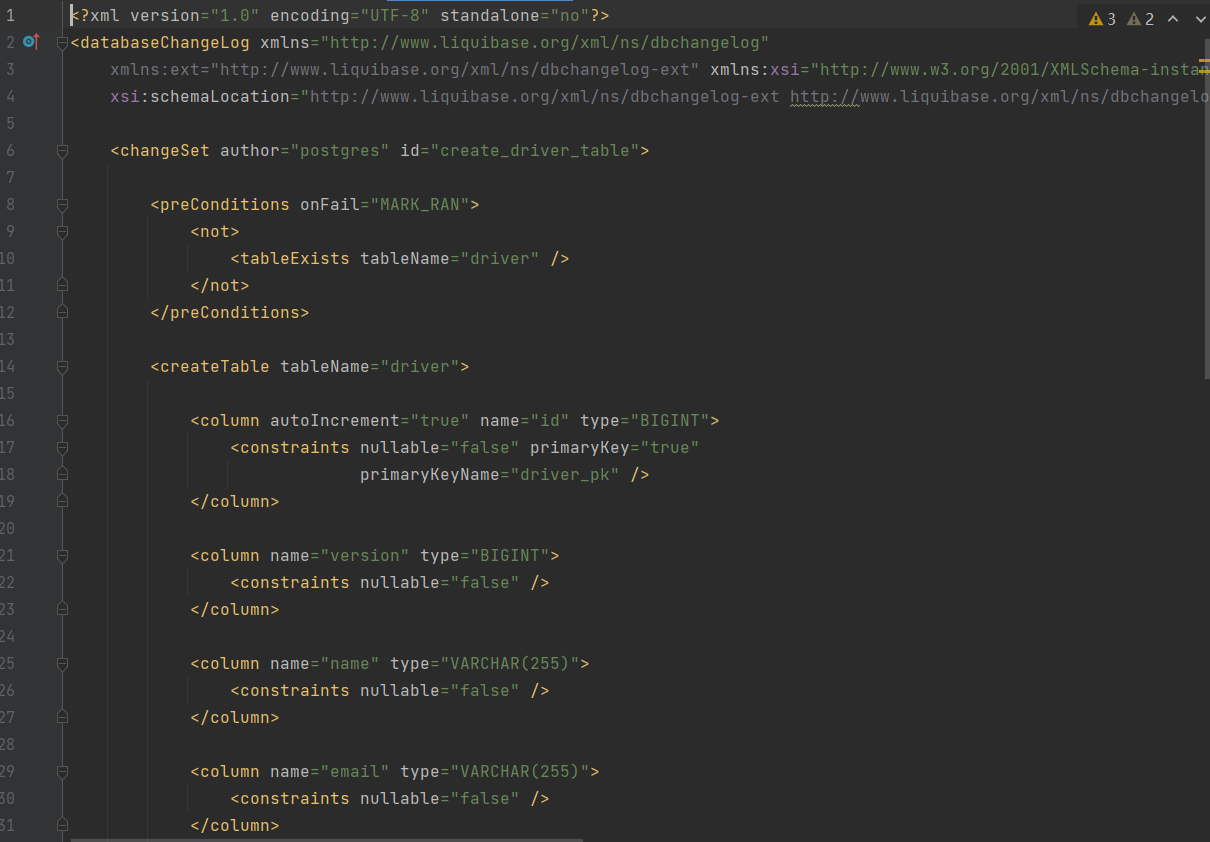
The answer is surely yes , we can use liquibase to help to build our database .

Liquibase is an open-source solution for managing revisions of your database schema scripts. It works across various types of databases and supports various file formats for defining the DB structure. The feature that is probably most attractive in Liquibase is its ability to roll changes back and forward from a specific point — saving you from needing to know what was the last change/script you ran on a specific DB instance.

//https://dzone.com/articles/introduction-to-liquibase-and-managing-your-databa

Building table using liquibase changelog

We can use liquibase changelog to create our tables , it can be SQL ,XML , JSON or YAML format , for this project we will use XML format :



This is how change logs look like . Inside our change log , we have our change set , it can be multiple change sets , but for each we need to specify the id . We use XML tags to “represent ” sql statements , for which liquibase will transfer it into sql statement and do as we want .

We use <createTable> tag to create our table , <column> tag to create attributes for our table , we can also add <constrains> tag to add some constrains to the column , such as not null , unique and so on .

Dealing with relations in liquibase

Unlike in springboot , we don’t specify the relation in both table , we don’t need to do anything in driver table change set,since we will put our driver Id inside publicTrip table instead of doing vice versa . For one to many relationship in RDBMS , we get the id from one side and put it in many side .So in public\_trip table change set , we will state the relationship .

Since public\_trip is many to one relationship to both driver table and car table , we will state like this in change set :

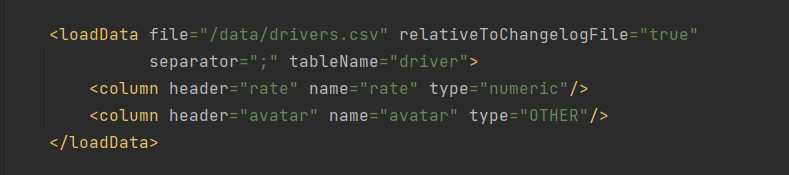


All we need is to specify ForeignKeyName and reference label . In reference , it should be the table name here they are table driver and car .

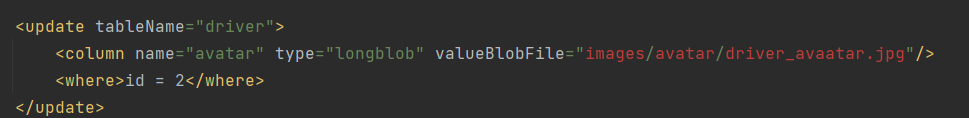
// Todu : add db constrains

Insert data into database using Liquibase

After creating the database , we can insert some data into database as test data . Inserting data we need another change set for sure , inside change set we will use <loadData> tag .



Then we put our data into csv files and put the link inside the tag , then liquibase will do its magic . Note that when inserting pictures into database , csv is a bit hard to handle that , we can use update statement instead .



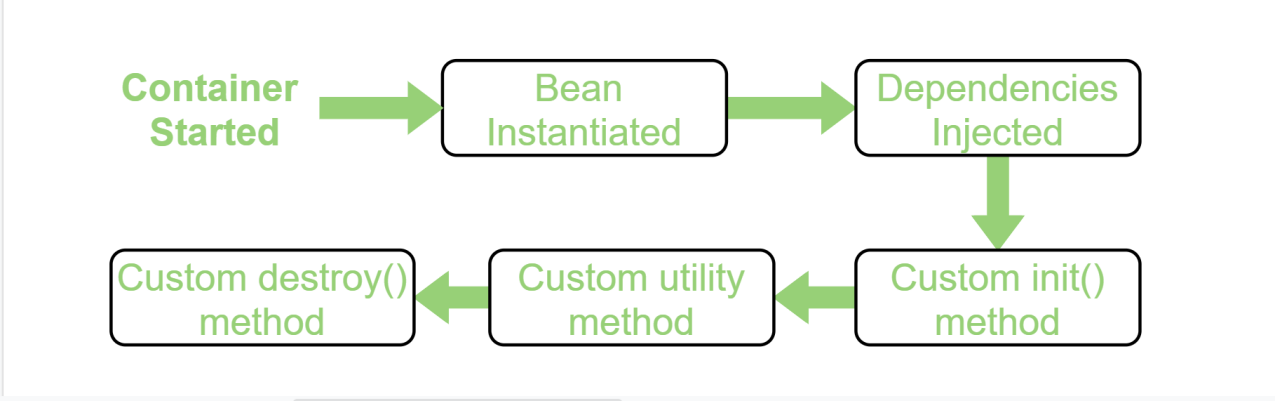
After finishing our model layer and mapping everything into database , we are finally read to store our data , but what’s next ?

###### Spring boot beans and life circle

The lifecycle of any object means when & how it is born, how it behaves throughout its life, and when & how it dies. Similarly, the bean life cycle refers to when & how the bean is instantiated, what action it performs until it lives, and when & how it is destroyed. In this article, we will discuss the life cycle of the bean.

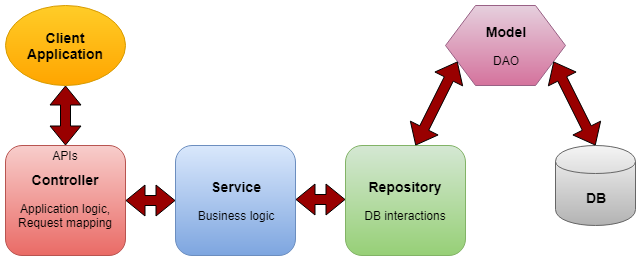
Bean life cycle is managed by the spring container. When we run the program then, first of all, the spring container gets started. After that, the container creates the instance of a bean as per the request, and then dependencies are injected. And finally, the bean is destroyed when the spring container is closed. Therefore, if we want to execute some code on the bean instantiation and just after closing the spring container, then we can write that code inside the custom init() method and the destroy() method.

The following image shows the process flow of the bean life cycle.



//https://www.geeksforgeeks.org/bean-life-cycle-in-java-spring/

###### Layers in spring boot



Controller- Controller layer contains the \*application logic, mapping the user request to particular functions and passing the user input to service layer to apply the business logic.

Service- This is the layer between the controller and repository which performs the \*business logic and validation logic. The controller passes the user input to the service layer and after applying the business logic, it is passed to the repository layer.

Repository- The layer which interact with the database CRUD operations via the DAOs(data access objects).

Model- Is the simple POJO classes which is acting as the DTO(Interact with application level data transfer) or DAO(Interaction with database operations)

Since we have already have our model entity ready , we can go to the repository layer .

Repository layer

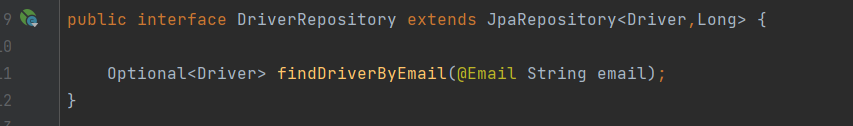
Spring Data takes this simplification one step further and makes it possible to remove the DAO implementations entirely. The interface of the DAO is now the only artifact that we need to explicitly define.

In order to start leveraging the Spring Data programming model with JPA, a DAO interface needs to extend the JPA specific Repository interface, JpaRepository. This will enable Spring Data to find this interface and automatically create an implementation for it.

By extending the interface, we get the most relevant CRUD methods for standard data access available in a standard DAO.

//https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa

Since our when mainly interact with the database CRUD operations , provide data access, we can using the api that spring provided by extending JpaRepository interface.



When extending , we need to provide two parameters , first is the entity class that you are having CRUD operation with , the second is the id type , we are using Long as our id , so we will put Long there.

By extending the interface, we’ve got the most useful CURD functionality here . Interface JpaRepository provides useful method including:

findAll() : returns all the objects T that meets requirements

getById(Id id) : get the object by entity id (primary key)

deleteById(Id id) : delete the entity by id provided ;

save(S entity) : create or update the entity by the entity passed in as parameter .

With all these methods ,we can build our CURD functionality easily , without writing SQL statements ourselves .

3. Custom Access Method and Queries

As discussed, by implementing one of the Repository interfaces, the DAO will already have some basic CRUD methods (and queries) defined and implemented.

To define more specific access methods, Spring JPA supports quite a few options:

simply define a new method in the interface

provide the actual JPQL query by using the @Query annotation

use the more advanced Specification and Querydsl support in Spring Data

define custom queries via JPA Named Queries

The third option, Specifications and Querydsl support, is similar to JPA Criteria, but uses a more flexible and convenient API. This makes the whole operation much more readable and reusable. The advantages of this API will become more pronounced when dealing with a large number of fixed queries, as we could potentially express these more concisely through a smaller number of reusable blocks.

The last option has the disadvantage that it either involves XML or burdening the domain class with the queries.

3.1. Automatic Custom Queries

When Spring Data creates a new Repository implementation, it analyses all the methods defined by the interfaces and tries to automatically generate queries from the method names. While this has some limitations, it's a very powerful and elegant way of defining new custom access methods with very little effort.

Let's look at an example. If the entity has a name field (and the Java Bean standard getName and setName methods), we'll define the findByName method in the DAO interface. This will automatically generate the correct query:

public interface IFooDAO extends JpaRepository<Foo, Long> {

Foo findByName(String name);

}

This is a relatively simple example. The query creation mechanism supports a much larger set of keywords.

In case the parser can't match the property with the domain object field, we'll see the following exception:

java.lang.IllegalArgumentException: No property nam found for type class com.baeldung.spring.data.persistence.model.Foo

3.2. Manual Custom Queries

Now let's look at a custom query that we'll define via the @Query annotation:

@Query("SELECT f FROM Foo f WHERE LOWER(f.name) = LOWER(:name)")

Foo retrieveByName(@Param("name") String name);

For even more fine-grained control over the creation of queries, such as using named parameters or modifying existing queries, the reference is a good place to start.

Service Layer

The repository layer will be referenced from service layer . As service layer serves as a transactional barrier and houses both application and infrastructure services. Furthermore, the public API of the service layer is provided by the application services. They often serve as a transaction boundary and are in charge of authorizing transactions. Infrastructure services provide the “plumbing code” that connects to external tools including file systems, databases, and email servers. These approaches are often used by several application services.

//https://www.baeldung.com/spring-service-layer-validation#:~:text=A%20service%20layer%20is%20a,the%20controller%20and%20service%20layers.

Let’s take CustomerService for example :



For service class we need to annotate it with @Service annotation , it indicate that current class belong to service layer , It is a specialization of @Component Annotation as well .

Inside service layer we will handle transections with our repository layer and provide the data to controller layer . Here we have the basic CURD functionalities to create , get , delete , update a customer and get all customer information . The data filtering and preprocessing is also inside service layer , we can use stream to filter our data and only send the necessary data to controller , then frontend .

Controller Layer

A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services. REST stands for representational state transfer and was created by computer scientist Roy Fielding

//https://www.redhat.com/en/topics/api/what-is-a-rest-api#:~:text=A%20REST%20API%20(also%20known,by%20computer%20scientist%20Roy%20Fielding.

Controller layer is mainly responsible for processing incoming REST API requests , process the data and pass them down to the service layer . And returning the response to the front end . Let’s take driver Controller as an example :



The controller class needs to be annotated with @Controller annotation or @RestController . The @RequestMapping annotation is used to map web requests to specific handler class or methods . All the method in driver class will be after “api/driver” path . We use constructor to inject spring boot beans . For each methods , we will annotate them with different annotations depends on the type of request sent from front end :

@GetMapping : mapping HTTP GET requests onto read handler methods.

@PutMapping : mapping HTTP PUT requests onto update handler methods.

@PostMapping : mapping HTTP POST requests onto create handler methods.

@DeleteMapping : mapping HTTP DELETE requests onto delete handler methods.

**CrossOrign (CROS)**

@CrossOrigin : for local develop environment //Todo

**PathVariable**

Path variable is the variable that you can pass within the url , usually quite useful when you want to access to certain item in a list based on id . When annotated with @PathVariable , the template variable in url will be mapped and set as method parameters .

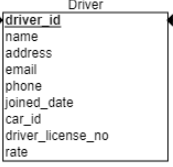
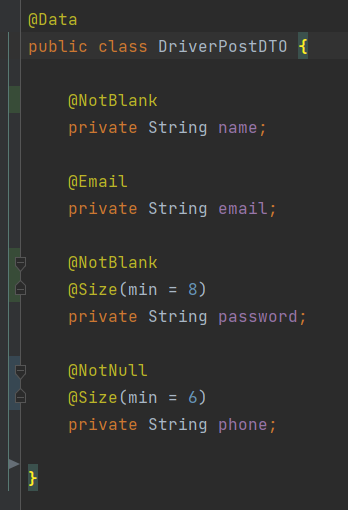
Here we notice that the object that we returning to the front end side is actually not entity but DTOs , what is DTO and how do we get them ?

DTO(Data Transfer Objects)

A data transfer object is an object that is used to encapsulate data before sending to the frontend .

When the frontend is rerquesting or posting data , it won’t always be all the data in an entity , it can be some necessary attributes in an entity or multi entities , since now return the whole entity or all the required entity to front end is not wise at all , that’s why we need to encapsulate them into a DTO , no matter request or response .It can also be useful when a method takes more than five parameters , it’s better to collect them into DTO .

Let’s take DriverPostDTO for an example :



DriverPostDTO is used for driver registration . There are quite many attributes in driver table , but we surely don not want to fill them all when registering , maybe just some necessary info , and we will make our profile better later . That when we need our DTO , since we only need some necessary attributes , we can put them into the post DTO , to tell front end what to send .

**Spring Data validation in DTO**

Here we annotated the field with some data validation annotations .

@NotBlank : indicates that name must not be null and must contain at least one non-whitespace character

@NotNull : indicates that the element should not be null

@Size : indicates that the password should have at least 8 characters and phone number should have at least 6 characters.

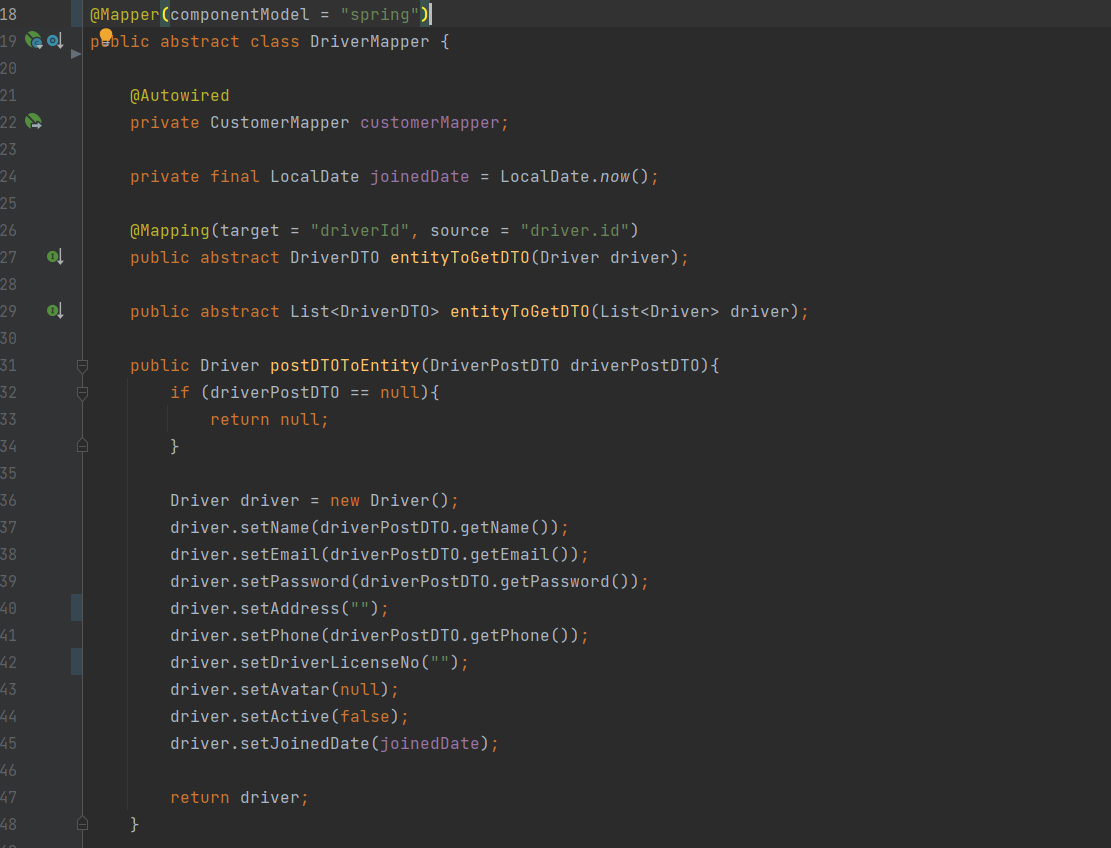
Now we have received our DTO from the front end , but we need to transfer them into an entity in order to save them into database . But we only have four attributes here , how should we deal with the others and how can I transfer it into the driver entity ? That is when we need our mapper to map a DTIO into entity or vice versa.

Mapping with MapStruct

MapStruct is a code generator that greatly simplifies the implementation of mappings between Java bean types based on a convention over configuration approach.

The generated mapping code uses plain method invocations and thus is fast, type-safe and easy to understand.

In order to map our drivePostDTO into a driver entity , we need a driver mapper . Since we are using mapStruct , it can be an abstract class , and the mapping code will be auto generated .



Inside driver mapper , we can declear our methods as abstract , then mapStruct will auto generated the code base on DTO and entity . If there are some attributes missing our name dismatch , we can annotate it with @Mapping annotation to tell mapStruct the source or add ingore = true statement to tell mapstruct to ignore it .

If we would like to add some customize logic while mapping , we can simply implement the mapping method . For postDTO we set the corresponding field to entity , for some field is auto generated : The joined date should be decide by local date , the active field should be false since driver just registered and certainly haven’t confirm his or her email yet . We will set the active back as soon as the email is confirmed .

After mapping , we can get our entity object and ready to save it into database .

After we have our controllers , DTOs and relevent logic , our backend is almost ready for requests and responses from front end .

#### Introduction to Docker

###### What is Docker

Docker is an open source containerization platform. It enables developers to package applications into containers—standardized executable components combining application source code with the operating system (OS) libraries and dependencies required to run that code in any environment. Containers simplify delivery of distributed applications, and have become increasingly popular as organizations shift to cloud-native development and hybrid multicloud environments.

Developers can create containers without Docker, but the platform makes it easier, simpler, and safer to build, deploy and manage containers. Docker is essentially a toolkit that enables developers to build, deploy, run, update, and stop containers using simple commands and work-saving automation through a single API.

Why are we using docker

Docker can make developing application either on your own or team much easier to manage . Let’s say we need a specific Java version (JDK17.0.1) and postgres version (14.1)for our current application , in order to let my application to run on the other computers properly , the other computers needs to have the same Java and postgres version installed , which made a huge work just for running the application . And different application may require different version as well . That when docker and containers come to play .

Imagine docker container as a box , it contains everything our application needs to run . Source code , dependencies , version and runtime environments are all in it . So this application can run on our computer in isolation , independent from all other processes on our computer , which makes it easier for me and any other people who want to run the application on their computer . It also helps to deploy our product to sever , we don’t need to configure that much on our sever since all the configuration is set up in docker container already .

**Docker vs virtual machines**

Since virtual machines can solve the similar problem , why should we use docker instead of virtual machines ? The reason is their different mechanisms .

Virtual machines has it’s own full operating system , which mean typically slower to start and run , while docker containers share the host’s operating system and runs or starts quicker . Docker containers are much more lightweight and faster in this occasion . But for some occasion you will prefer virtual machines instead of docker , both of them have their advantages.

###### How docker works

Images and containers

Docker images are like the blue prints of the container . They have the things store in them , including :

Runtime environment : versions like JDK 17 and postgres 14.1 and so on

Application code : the code itself

Dependencies : like Maven dependencies

Extra Configuration : environment variables and so on

Commands : command to run the app or make it to work

Images have them store inside but now running them . They are a independent file system from your computer and read-only , which means once an image is created it cannot be changed . If you need to apply some new changes you need to create a new brand new image .

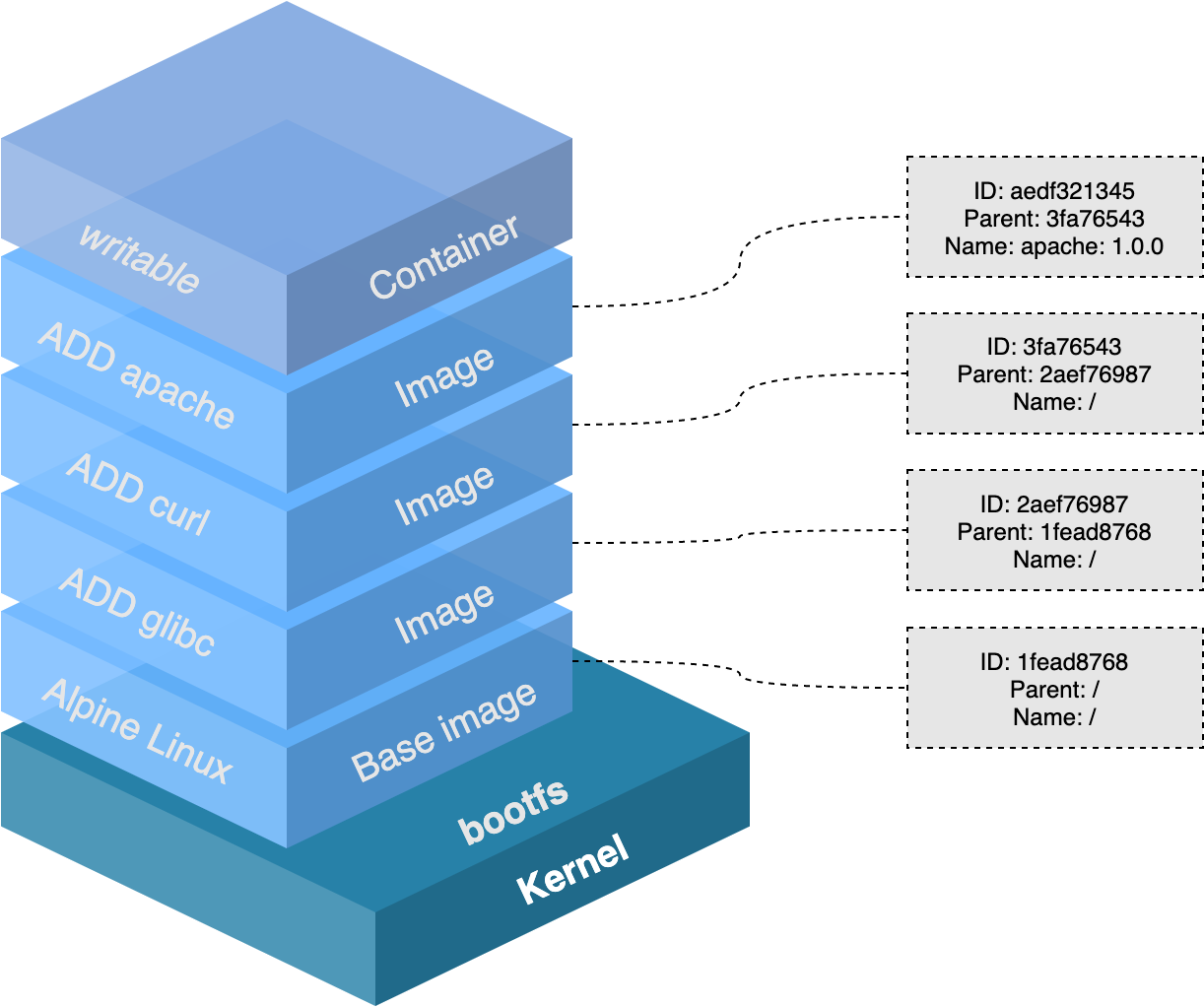
Containers are runnable instances of images , it is a process exactly outlined as the image we created . Container is also an independent process , meaning they run independently from any other processes in our computer . The whole thing is like a box , our application is running in it isolated , packaged with everything our application need to run with .

Since everything is isolated and independent , we can just share our image if the others would like to run the process on their computer . They don’t have to set up the environment or have specific Java version installed , since everything is stored in our blueprint --- image .And the application will run exactly the same way on my computer .

How images are made

Docker images are made up from different layers . where each layer add something else to the image incrementally , the order of the images does mater .

//https://ragin.medium.com/docker-what-it-is-how-images-are-structured-docker-vs-vm-and-some-tips-part-1-d9686303590f



In image we start with our base image or also as known as parent image , which is the first layer in the image . It describes the operating system and runtime environment that we want . And we build our image by adding other layers on it such as our source code , install dependencies and so on . Usually we can get our pre-made first layer from docker hub .

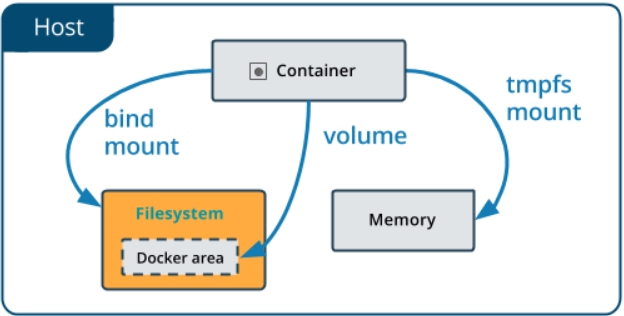
**Layer Caching**

Since image is build by layers being stacked on top of each other , each time we adding layers we are essentially changing the image , and we are giving docker some work to do , to add something to the image when we try to build it , and it takes some time .

Every time we change something , like our source code , we need to build a new image to pick it up , since our old image is based on old code and read-only . In order to rebuild the image docker is going rebuild by adding those layers in turn , which might be quite time-consuming when building the layer from base layer . The actual is , docker will caches our image layers when after building it , which ends up with a cached version image with every layers . Every time docker tries to build an image and work through the layers , it will look into the caches and try to find a image that it can use for the new image that we are creating in order to reduce the workload and building time .

###### Volumes in Docker

Since our image is read-only and our container is based on our image . Imagine such a case , we made some code changes in our app , and we would like to pick up this change . We can stop our container and rerun it , but it does not work , since the container is based on our image and our image does not change until we build a new one . So in this case we have to build a new image and a new container based on it to pick up our change , even we have layer caching to save our time but still quite a complex process . But there is indeed a solution for it , that is when volumes come to play .



//https://prakashkumar0301.medium.com/docker-volume-ba830fbbdde5

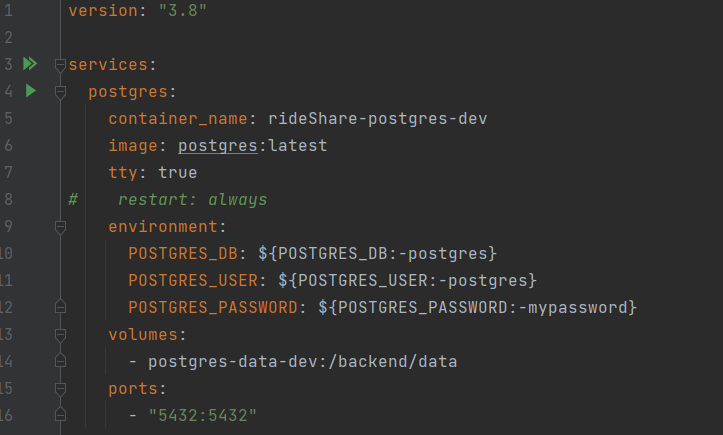
Volumes is a feature of docker that allow us to specify folders on our computer that are avaiable to run containers , and we can map those folder to specific folders inside the docker container , so that if something changed in our computer in that folder , the change can be reflected in the folder and mapped to container . With volumes we can pick up our code changes easily.

Volumes provides us a way that we can make changes and preview them without rebuilding the images all of the time . But one thing is important , the image will not change . Volume is just mapping the changes into container and our image remains as it was . If we want to share the latest changes to the others we have to rebuild the image . But it is quite useful when we are building and testing our application .

###### Docker Compose

So far we have discussed the images , volumes and containers . Every time we would like to dockerize something it would be quite a long process , with many lines of command and we need to type them in terminal one by one . And sometimes we may have multiple project and we would like to run those containers at once , since one may depend on another , like our front end is depend on backend and backend is depend on database . We can run those containers one by one by typing the commands , but it can be messy and complicated . That is why we need to introduce a easier way to manage our containers which is something called docker compose .

Docker compose is a tool which give us a way to make a single docker compose file that contains all the container configuration and dependencies . Let’s take the database as example :



Inside docker compose file , we need to specify version in the first line , and put our container configuration into services . Inside postgres config , we specify :

container\_name : the container name .

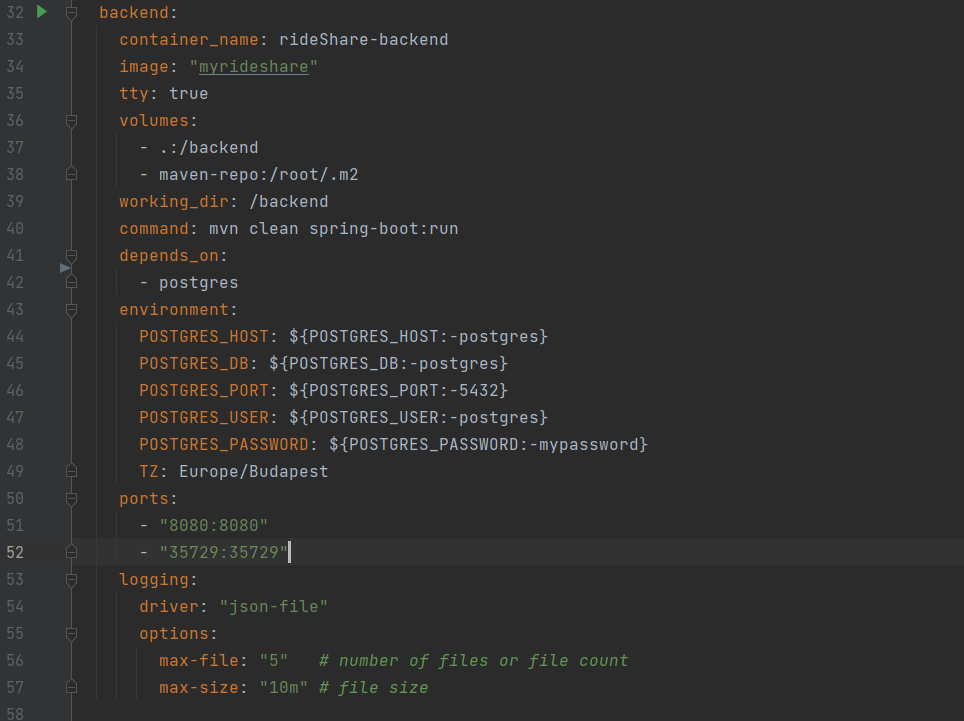
image : the base image that we need to use .

environment : the environment variables , here is mostly the database configure and authrization

volumes: The list of volumes that we need , use relative path .

ports : port mapping , we map 5432 port in docker container to port 5432in our machine .

###### Dockerize our spingboot application



Inside docker compose file , we specify :

container\_name : the container name .

image : the base image that we need to use .

volumes: The list of volumes that we need , use relative path .

working\_dir : specify the working directory in our docker container.

command : the command that we need to run our app properly

depends\_on : indicates that our backend container depends on postgres container

environment : the environment variables , here is mostly the database configure and authrization

ports : port mapping , we map 8080 port in docker container to port 8080 in our machine .

logging : logging configures

After configured everything in our docker compose file , we just need to simply type :



to run our application , which is much simplier than creating images volumes and containers step by step .

## Frontend part

#### Introduction to TypeScript

#### Introduction to React