# 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

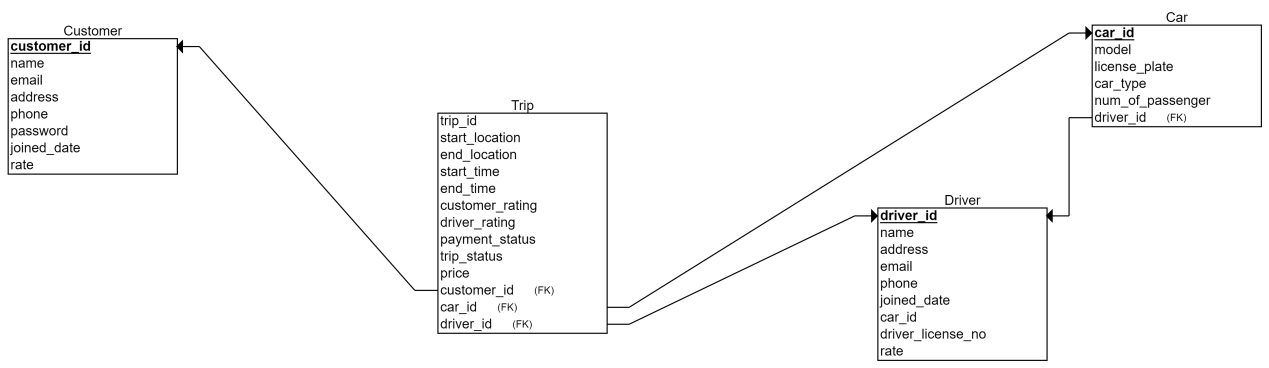
###### Introduction

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

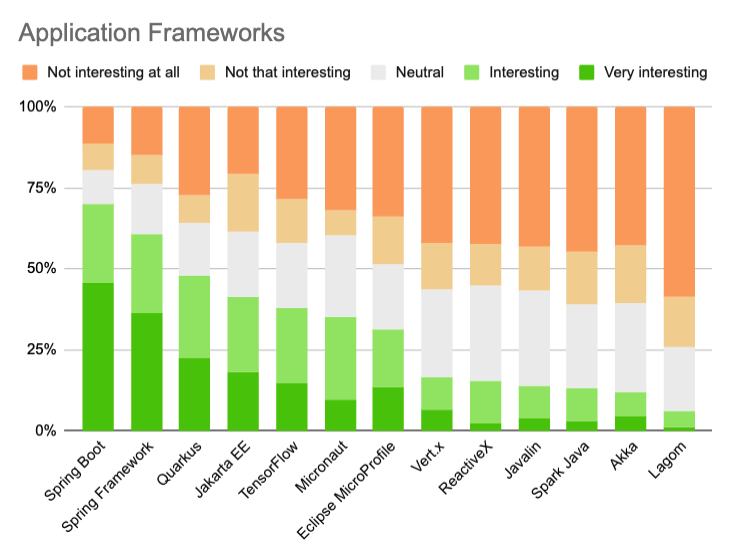
here

#### 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

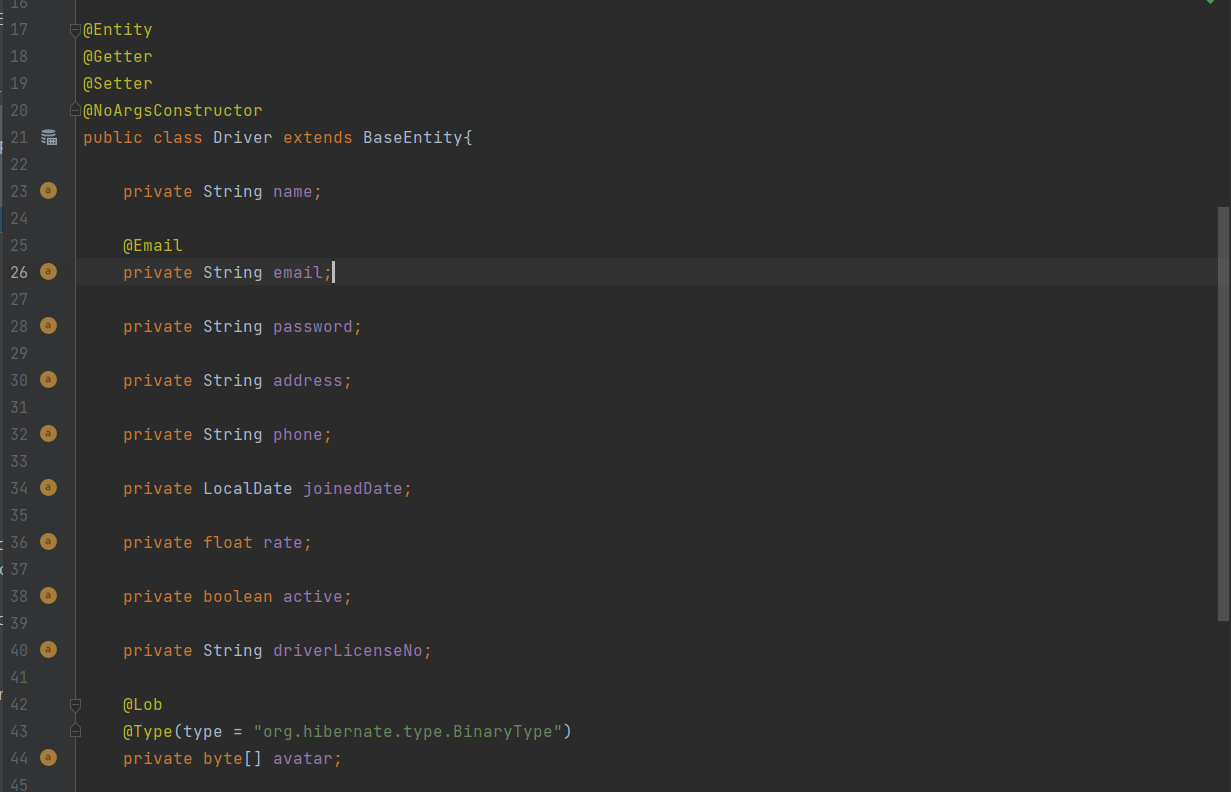
After we have our database design , we need to link / map it to spring boot using java so that we can manipulate data directly using java , where we will need our entity class

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 :

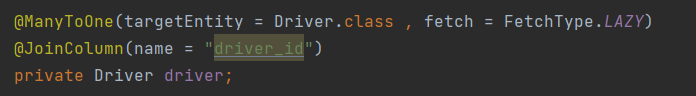
For driver table we have two relations :

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 ?

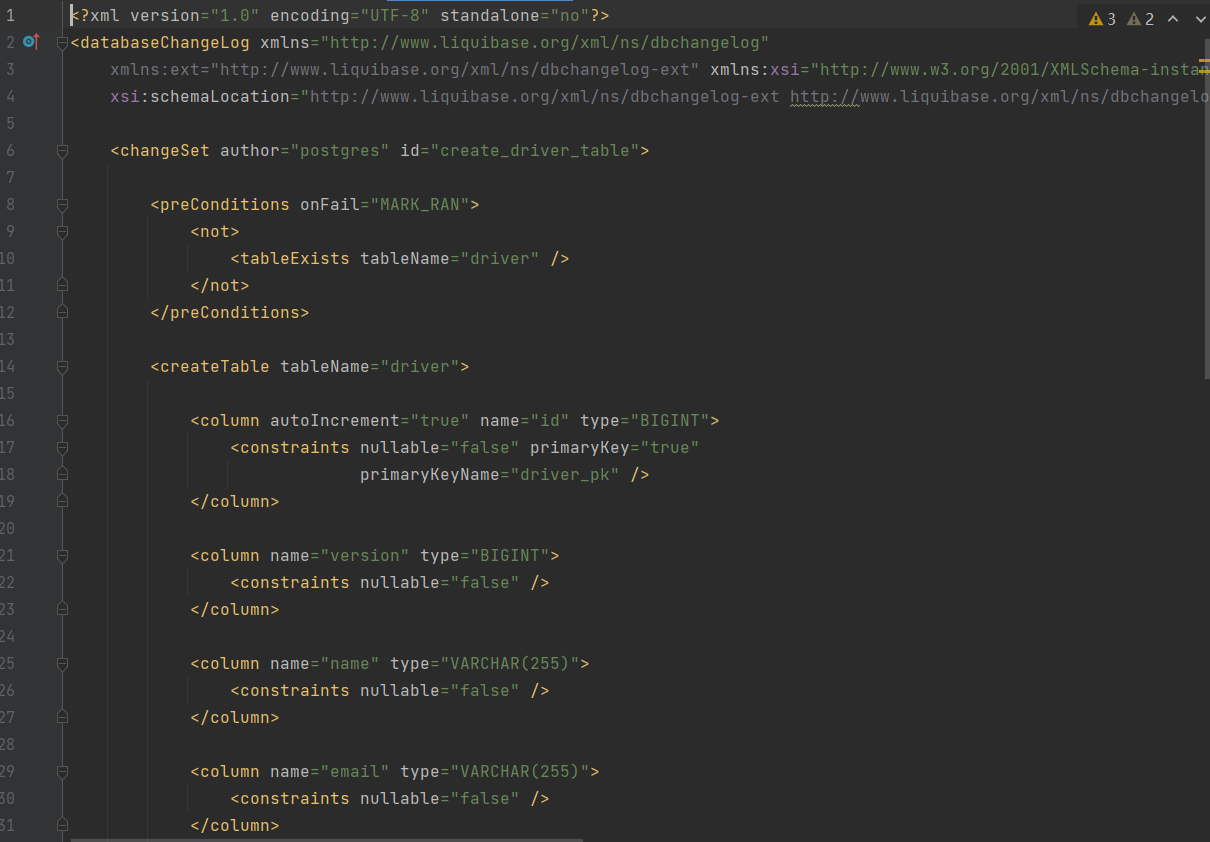
The answer is surely yes , we can use liquibase to help to build our database .

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 :

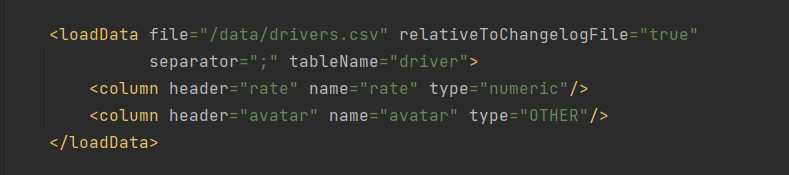


All we need is to specify ForeignKeyName and reference label . In reference , it should be the table name here they are table driver and car .

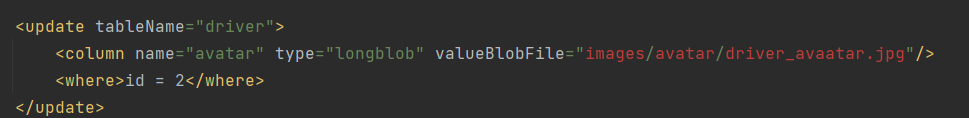
// Todu : add db constrains

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 .

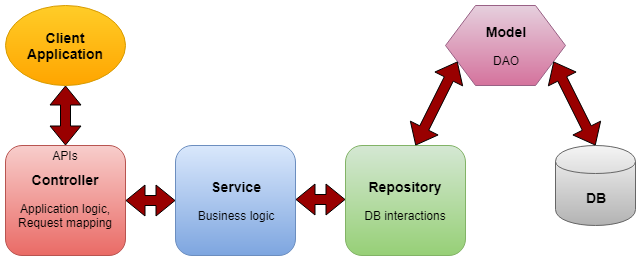


Then we put our data into csv files and put the link inside the tag , then liquibase will do its magic . Note that when inserting pictures into database , csv is a bit hard to handle that , we can use update statement instead .



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

###### 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)

After our model layer , we will go to repository layer .

#### Introduction to Docker

## Frontend part

#### Introduction to TypeScript

#### Introduction to React