RECRUITING TEST
BACKEND ENGINEER - IoT - CONCEPT REPLY

Test scenario : mobility on demand
Applicant : Stefanie SZABO
Test duration : 17.09.18 - 24.09.18

DISCLAIMER

I hereby assure that all written notes, drawings, diagrams, and source code were created by myself and without the help of third parties. Ideas, texts, code, and other material that came from external sources were marked as such and provided with citations as appropriate
Everything that was not marked as a quote or citation is my own "intellectual property".
None of the following was presented to other enterprises as solution for a test nor has it been published or shown to others.

I am aware of possible (legal) consequences if any of the aforementioned statements prove untrue.

Wiesbaden, 23.09.2018, S. Szabo

INTRODUCTION

The tests provided deal with creating services for a mobility on demand scenario. As far as I understand, mobility on demand describe services like Uber or Lyft (or, in a broader sense, BlaBlaCar or Mitfahrzentrale) that connect two types of users:

- the customer, who wants to be picked up and dropped off at certain given times and locations, and
- the service providers, who use their private cars to transport the customers as to their request.

The connection is provided via an application that is supposed to be created during this test. It consists of

- Task 1 : create a database scheme in UML
- Task 2 : implement a management service for the data stored in the DB from Task 1
- Task 3 : implement a scheduling service that chooses a car ("service provider") based on the demands of the customer
- Task 4 : create and implement test cases for the management and scheduling services
- Bonus Task : design, implement, and test a dashboard for a service tracking the cars

Unfortunately, I got a little stuck on task 2 and 3 and did not finish everything in detail. The bonus task was only started in that I created some mock-up sketches of how a service dashboard could look like.
I apologize for this and am aware that this will not help my cause.

TASK 1 : DATABASE SCHEME DESIGN

To describe it very simply, our service creates a 'relationship' between the customer (henceforth user) and the service provider (henceforth simplifyed as car) by means of a booking demand.
This means:

- a user places a demand for transportation.
  They specify a
  - pick-up location,
  - the earliest pick-up time,
  - the drop-off location,
  - the latest drop-off time,
  - and further requests towards the car depending on his traveling circumstances (e.g., luggage, group size, pets, children, accessibility needs, etc.)
- based on the demand, a car is chosen from a list.
  Those cars have features themselves, e.g.
  - a model,
  - a type,
  - a location that determines how fast they can react to a demand,
  - a number of seats,
  - luggage space,
  - and so on.

The actual means of how the car optimally meeting the user's demands is chosen is not yet important. First, we need to specify the features and attributes of our users, cars, and demands. After that, we can finally determine their relation in the database via an UML diagram.

The User

The user is the person who wants to travel and requests a car. This request is constrained by several demands. However, these demands may vary from request to request. Some aspects of the user stay as they are - those not only verify the identity (for administration purposes and the driver's information), but may also influence his travel circumstances.
For the purpose of this test, I chose the following features:

USER

feature	example/meaning	value in DB
user ID	alphanumeric user ID for internal identification	varchar(15) - primary key
user PWD	password for user account. Security is important! (only hash is stored!)	varchar(25)
uMail	user's mail address. In a real scenario, this should obviously be encrypted.	varchar(50), null possible
uPhone	user's phone number. Should in reality be encrypted!	varchar(15)
uName	user's nickname	varchar(20)
uAge	user's age. Must be greater than 18.	tinyint >= 18
uGender	male, female, or other	enum('M', 'F', 'OTHER')
uAccess	accessibility options in case of special needs	varchar(100)
uOrderStatus	checks if user has placed a request or not	boolean

S. Seidl
Recruiting Test
Concept Reply

The car

The car is the transportation means a user can request. It has several features that may fulfill the requests a user has upon booking. Generally, they make traveling easier or more comfortable!
For the purpose of this test, I chose the following features:

CAR

feature	example/meaning	value in DB
carID	alphanumeric car ID for internal identification	varchar(15) - primary key
carLoc	in real scenario, car location should be GPS coordinates - to make things easier, I use an ~~string~~ integer.	~~varchar(25)~~ integer
carPlate	plate number - the user should find their car!	varchar(20)
carModel	manufacturer name and model name	varchar(50)
carType	e.g. city car, SUV, sportscar, ...	varchar(20)
cEngine	petroleum, diesel, hybrid or electric	enum('P', 'D', 'H', 'E')
cNumSeats	amount of available passenger seats	tinyint, >= 1
cLugSpace	for ease of test: small, medium or large	enum('S', 'M', 'L')
cComfort	e.g. AC, DVD player, leather seats, ...	varchar(100)
cSpecialServ	transport of children, pets, and/or travelers with disabilities	enum('KIDS', 'PETS', 'SPECIAL NEEDS')
cOrderStatus	checks availability of cars for demands	boolean

The demand

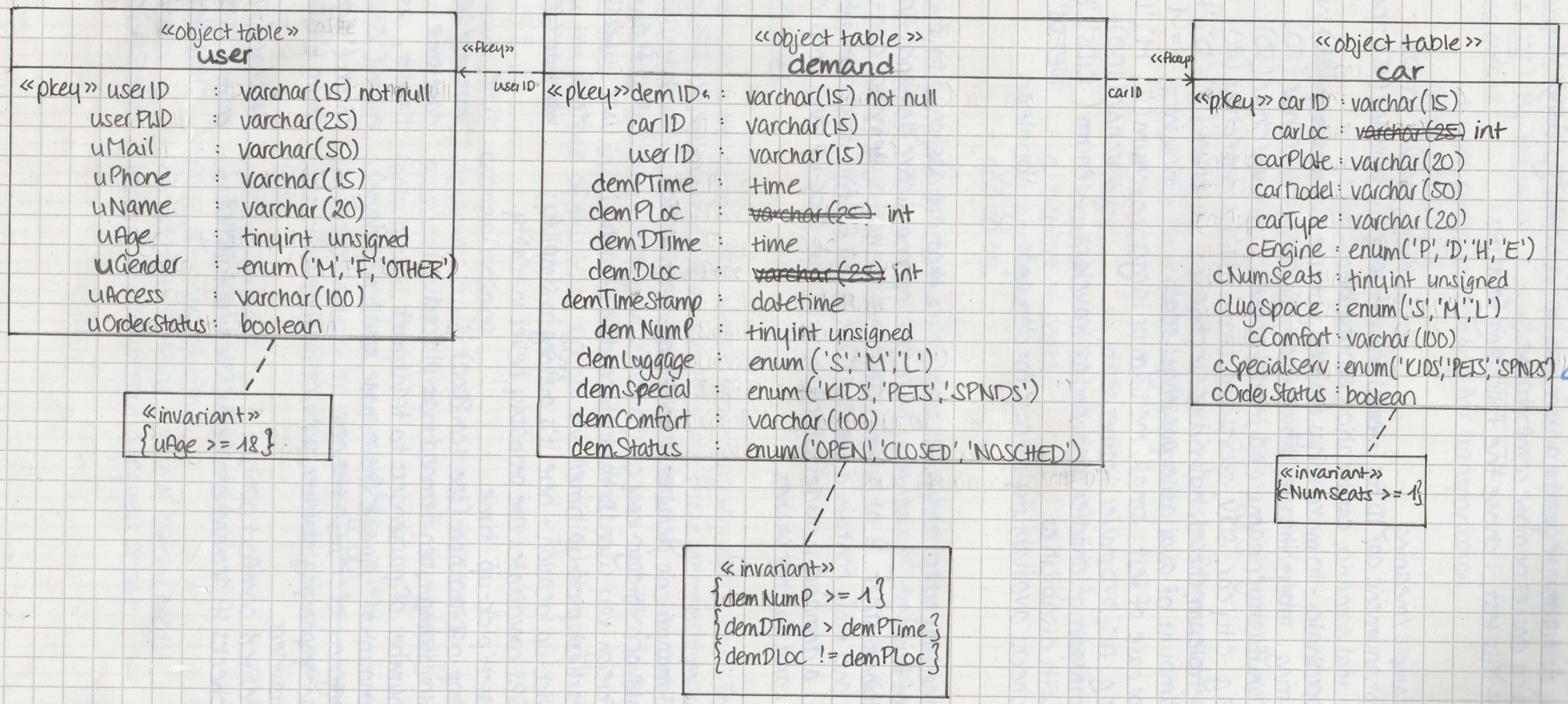
Demands are placed by users and fulfilled by cars that are assigned to it. For the purpose of this test, a user may have one open or active demand, and a car may fulfill one demand at a time. That means that a demand contains a user ID and may contain a ~~the~~ car ID if a car fulfills its constraints. Apart from that, a demand includes information about pick-up and drop-off time and location and further wishes.

DEMAND

feature	example/meaning	value in DB
demID	alphanumeric demand ID for internal identification	varchar(15) - primary key
carID	null or a foreign-key car ID that is assigned	null or fkey carID
userID	foreign-key user ID that placed the demand	fkey userID
demPTime	earliest pick-up time	time ~~varchar(25)~~ int
demPLoc	pick-up location - here, it's a ~~string~~ integer to simplify; GPS coordinates are probably better in reality.	
demDTime	latest drop-off time	time ~~varchar(25)~~ int
demDLoc	drop-off location (see demPLoc)	
demTimeStamp	timestamp when demand was placed	datetime
demNumP	number of passengers to transport	tinyint, >= 1
demLuggage	amount of luggage (here: none, small, med., large)	enum('S', 'M', 'L')
after → ~~demUrgency~~ bout it	~~urgency of transportation~~	~~boolean~~
demSpecial	accompanying children, pets, or special needs travelers	enum('KIDS', 'PETS', 'SPNDS')
demComfort	desired comfort options	varchar(100)
demStatus	checks if demand can be fulfilled/is fulfilled	enum('OPEN', 'PROGRESS', 'NOSCHED')

The UML Diagram

Finally, let's turn this into a diagram!



To bring the diagram to paper, I used the thread "Database design using UML Class Diagram" by user3167689 from Stack Overflow and the tutorial "Part II. Design and Implementation of SQL-92 databases" from the computer science department of the TU Cottbus ~~from~~ as help and guidance.

- Links: https://stackoverflow.com/questions/21358774/database-design-using-uml-class-diagram
  - https://oxygen.~~in~~informatik.tu-cottbus.de/IT/Lehre/MDD-Tutorial/#d5e126

TASK 2: MANAGEMENT SERVICE

S. Szabo
Recruiting Test
Concept Reply

In the next step, we have to implement a management service for our mobility-on-demand scenario. For cars, users, and demands, it is supposed to allow adding, editing, and removing operation. The task itself allows to use an 'in-memory' database and recommends using REST and HTTP.

Since I'm most experienced with C++, I decided to implement the service in this language. Determining a basic structure was fairly easy - I wanted to create a class for each object (i.e. cars, users, and demands). Via set- and get-functions, I was able to specify constraints for all the attributes an object has, thus fitting them to the database.

A bigger challenge (and time sink) were the dozens of questions that occurred during the programming. In a real scenario, one may most likely ask colleagues if they are available - which you obviously can't do in a test!
Some questions I had - and eventually answered - were:

- What is the optimal representation for an 'in-memory' database table?
  (I eventually chose a map after pondering over many weird alternatives. Maps are advantageous for their sorting and storage of data as ordered pair - ensuring that keys are unique and always refer to their object)
- How do I update my 'database table' after changing an object?
  (In the end, I 'brute-forced' it by just overriding a map entry with the same ID as the changed object)
- What is a good representation for locations?
  (At first, I thought strings were a good idea - until I had to write a function that updates the car location.)
- How do I make sure that a user only places one demand at a time without having to include a user variable in the demand?
  (In the end, I added a map<string, User> as private variable to the Demand class and created a function to check it)

More to my solutions can be found in the code - I tried to be very diligent and extensive with the comments. I'm aware that this is exaggerated even for a collaborative setting - it serves more to explain my thoughts.

One final remark - in the end, I decided to omit the HTTP/REST part. While I theoretically understand what to do and how the requests work, and even found a fitting framework for a RESTful API in C++ (Casablanca) ... I started confusing myself with questions and approaches and took too much time.
That part really made me wish I could ask someone for help - Stack Overflow couldn't save me!

TASK 3: SCHEDULING SERVICE

Actually, I found the scheduling service to be easier than task 2. Admittedly, that partially stems from the fact that I tried to keep it very easy.

At first, I had to decide which car features are absolutely crucial and cannot be optional. I found those to be

- the number of available seats (to leave nobody behind)
- the amount of luggage space (to leave no suitcase behind)
- the special service options (to leave no kids, pets or people with disability behind)

Of course, the scheduler needs to respect the pick-up and drop-off specifics. To ensure that the car will pick up the user at their desired time and location, the scheduler will assign the car closest to the pick-up location.

Furthermore, I assumed that cars wouldn't have a 'range limit'. In reality, there would be a lot more to take into account: actual distance and traffic data to determine whether the driver can make it to the pickup in time and driver preferences and limits - however, I believe that that would take a little more than seven days (and to be honest, more knowledge than I have right now!).

From there, my scheduling idea is simple: The features 'available for booking', 'number of seats', 'luggage space', and 'special service' each get a weight of one. The maximum weight a car can have is, therefore, 4.
The scheduler looks through my car 'table' and calculates the weight of each car. Only those with weight 4 will be considered further. Those 'candidate cars' are stored separately. The scheduler then calculates the distance to the pickup location for each candidate. The closest car will be chosen and the statuses of the car and the demand updated.

I am not sure at all if that is enough and have several more ideas to enhance this - but unfortunately, it's Saturday already and I need to move on about yesterday ...

TASK 4: MANAGEMENT AND SCHEDULE SERVICE TESTING

I have never done actual unit-tests for my own code before! Right now, my testing experience is limited to some CANoe stuff and HiL-testing with existing test cases. But I'm excited to learn something new!

To find a good test framework, I started research, - looking at Stack Overflow, reddit, and the article "Exploring the C++ unit testing framework jungle" by Noel Llopis. I also contemplated throwing the towel. Ultimately, I decided on Catch.

Before starting to implement the test cases, I first thought about what I would need to test. I identified the following questions - however, I wasn't aware that systematic methods exist (I'm sorry)!

GENERAL QUESTIONS:

- are all attributes in the (specified) ranges?
- are required attribute values not empty?

„DATABASE" QUESTIONS:

- do the operations "add to db", "edit object", and "remove from db" work as intended?
- what happens if one tried to...
  - ... add an already existing object to the DB?
  - ... remove an object that is not in the DB?
  - ... edit an object that is not in the DB?

„DEMAND" IN PARTICULAR:

- do the assigned user (and car) exist in the database?
- is the drop-off time later than the pickup time?
- do pick-up and drop-off location differ from each other?
  - what would happen if I tried to enter a drop-off time that's earlier than the pick-up time or if pick-up- and drop off location were the same?
- what happens if one further tried to...
  - ... add a car to a demand that is already booked?

- ... try and create a demand for a user that already has an open demand?
- ... delete a car or user that are currently assigned to a demand?
- ~~are the cases 'car is found that fits the demand' and '~~

SCHEDULING QUESTIONS

- do the maps on which the scheduler works exist?
- distance calculation:
  - does the car exist?
- comparison car-demand:
  - does the car exist?
  - does the demand exist?
  - is the demand open and the car available?
- scheduling:
  - does the map over cars exist?
  - does the demand exist?
  - ~~is the car~~ is the demand open?
  - does the status update work as intended?
- closing a demand?
  - do the car, user, and demand exist?
  - has the car arrived (i.e. does its location correspond to the drop-off location)?
  - are the statuses updated as intended? (car and user are available again)
  - is the demand deleted properly?

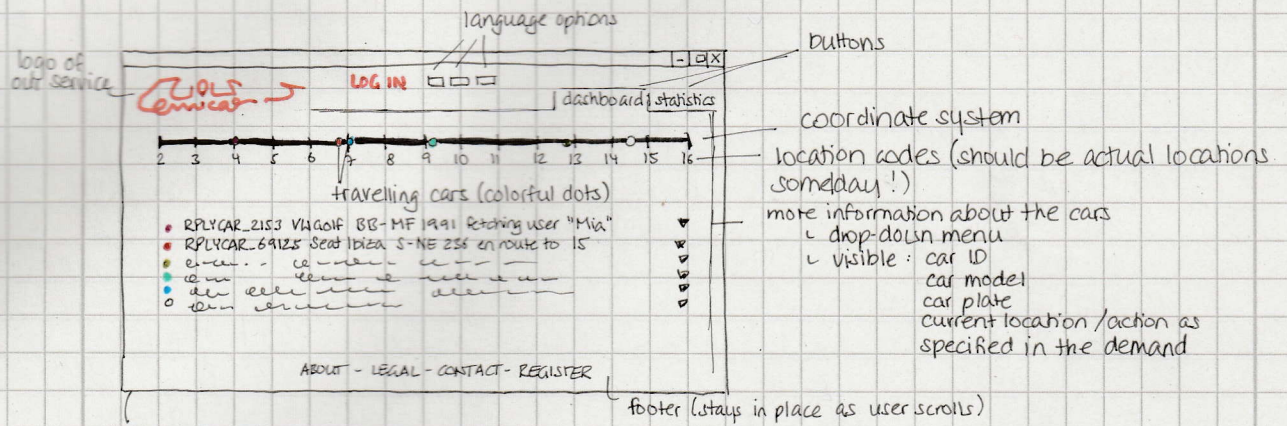
Via github, I downloaded Catch - which is, thankfully, pretty lightweight. - and implemented the test cases according to my questions.

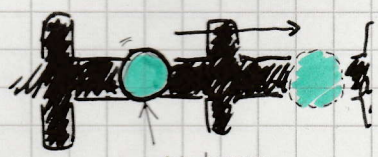
BONUS TASK: SERVICE DASHBOARD

Last, but not least, the task specifies the design, implementation, and test of a dashboard service. It is supposed to show an 1D-coordinate system and present the car's' current positions on the coordinate system. Moreover, a bar chart shall show all the cars' travelled distance.

As of right now (Sunday at 16:00), I only have enough time left for mock-ups and some notes on the implementation.

So without further ado, let's mock up!



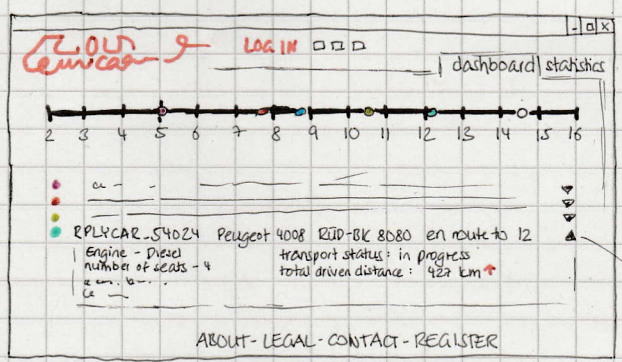


dots represent cars
color is randomly chosen
move along as the car location updates (every 30 seconds?)

on hover:
row with information about the car gets highlighted:

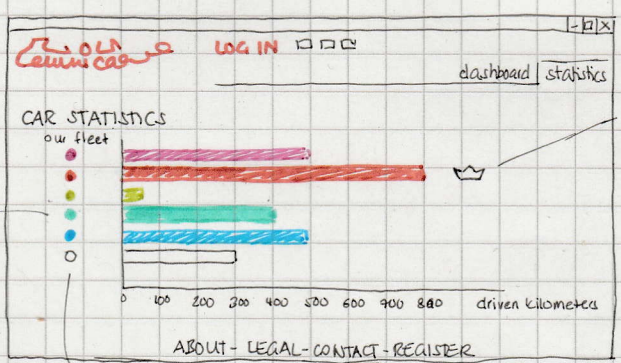
~~RPLYCAR_54024 Peugeot 4008 RUD-BK 8080 en route to 12~~

color scheme:
white background, black coordinate system/type
font: something easily readable, e.g. Flama, Candara, ...
orange logo (to fit Reply!)
colorful dots/bar chart



clear, simple shapes

shows more information about the car when clicked
only one of those extended menus open at a time (accordeon menu)

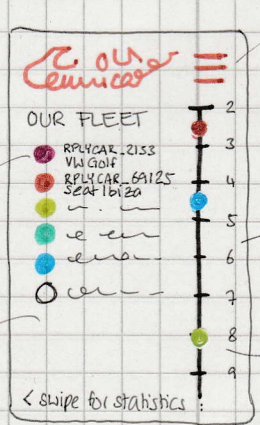


dot colors correspond to colors on dashboard

car with most driven kilometers gets a tiny crown :)

on click on a colored dot → return to dashboard, info menu ~~of~~ the corresponding car open

and on mobile?



hamburger menu (LOG IN REGISTER ABOUT CONTACT LEGAL NIGHTMODE ↑ inverts colors)

touch car dot to see info

scrollable?

everything should be a little smaller

dots move up and down