SoftwareDesignDoc

tobiassontom

November 2020

1 Introduction

This project aims to create a system that handles the booking of rental vehicles, and the return of them. It should also be reasonably easy for a business owner to have the system customised in terms of the UI and the way data is stored. The project should be considered production ready when a customer of a business can use the application to book and return a vehicle. And also being easy to customise in terms of UI and data storage method.

2 Scenarios

Günter's gas station

An owner of a gas station, called Günter has vehicles for rent. Günter wants to make it easier for his customers to rent his vehicles and to save himself time. Currently, he has to manually write down every booking on a piece of paper, and make sure that there are no double-bookings. He would like a system where the customer can book by themselves, and which also keeps track of double-booking scenarios. Since he has his own gas station, he wants his logo showing in the UI. He also wants the system to not lose the bookings if the power goes out, or if the computer running the system breaks.

Ursula's used vehicles

Ursula is the owner of a used car business. They buy an sell cars, but they also rent out cars that they haven't sold yet. She wants to spend more time with customers that buy cars rather than to handle the renting. She wants a system where the customers can book from home, and also in the store. She is fine with keeping the standard looks of the UI and she just wants the system to be up and running as fast as possible.

3 Proposed Solution

The project will follow the MVVM-pattern to decouple the UI from the underlying logic, since the business owner might want to modify/change it. The data storage logic will be placed in service-classes so that the business owner choose the data storage method for themselves. If the business owner would like to use a No-SQL database such as MongoDB, the only thing that would need to change is the service implementation, since the Views, ViewModels, and Models won't have to care how data is stored. The solution will be based on WPF C#.NET Framework for an easy development process.

UI

The standard UI that comes with the system will have a start page that contains two buttons, one that takes the customer to a booking page, and one that takes the customer to a return page.
On the booking page, the following information should be required.

- Personal identification number
- Vehicle Category
- Date of booking

There should exist a button named "Rent" that the customer presses when all the required information is entered. When the button is pressed the customer should be presented with a "Thank you" message and their booking number and the date from when the booking is valid. The price should be displayed upon return of the car since we cannot calculate the price without knowing exactly how far the customer will drive or how long they will have the vehicle.
On the return car page, the following should be required information.

- Personal identification number
- Booking number
- Current mileage of the vehicle

When the customer has filled in this information, there should exist a button "Return vehicle" that when pressed shows a nice message along with the price, booking number and personal identification number.

Database

The standard database implementation will use postgres (relational SQL). The following SQL query describes what the columns in the table will look like, and what types they will be.

```
CREATE TABLE bookings (
        bookingnumber text,
        pidnr NUMERIC(10) NOT NULL,
        vehicle text NOT NULL,
        dateofbooking numeric NOT NULL,
        mileage NUMERIC NOT NULL,
        PRIMARY KEY(bookingnumber)
);
```

Business logic

The Model classes for the project should be based on the following ones

- Booking
- Customer
- Vehicle
- Van
- SmallCar
- MiniVan

The different variations of vehicles should all inherit from the Vehicle class, this structure will allow for easy extendability in case there is a need for more types of vehicles, this will save time and energy since vehicles share lots of the same code. Having the customer as a separate class and not just a variable in booking is good for future functionality where there may be a need for a login page where more than the customers personal identification number is needed.
A booking will register the following information

- Booking number
- Customer personal ID number
- Vehicle category
- Date and time of booking
- Current vehicle mileage

This means that the system will have to generate a booking number, and store it along the information the customer supplied.
A return of the vehicle will require the following information

- Booking number
- Date and time of vehicle return
- Current vehicle mileage

The things the customer will have to supply the system with will be personal ID number(for security) and booking number, and date of return. then the system can gather the rest of the information from the database. The system can then calculate the price for the renting and display it to the customer.

A class called **BookingService** will contain the validation of bookings and customer information. It will handle the database implementation so that it is decoupled from the rest of the system.

Testability

The systems business logic should be comprised of many smaller methods that have a return values, this will make the implementation easy to unit test. Large methods that have ”void” return value should be broken down into smaller ones that have a return value that can be asserted as either correct for the given input, or wrong for the given input.