

ParQer



MADE BY

Athusan Kugathasan	Martin Rune Rylund
s185098	s185107

Danmarks Tekniske Universitet

Avanceret objektorienteret programmering med C# og .NET F20

Date: 20/05/2020

 <https://github.com/mrylund/parking-app>

Table of Content

Table of Content	1
What is the purpose of the app?	2
Software used under development	2
Login Information	2
Installation	2
First considerations	4
Overview of functions to be implemented	4
The UI (User Interface)	5
The Code Implementation	11
MySQL implementation	11
Global Functions	11
User class	12
Further work	13
Mobile App	13
Overview of all vehicles	13
Adding ownership of multiple cars	13
Conclusion	13

What is the purpose of the app?

We have taken a point of view from the Student Dorm we live in.


At the residence, there are private parking spaces intended for the students living on the property, but what we often see is that a lot of vehicles from the outside are parked on these spots.

This is what we have tried to solve by designing this to give the supervisor/inspector of the place an easier way to determine if the vehicle is a student's or a guest of a student.

Software used under development

The softwares used while developing the application are the following:

 Visual Studio 2019 (IDE)

 GitKraken (Interface for easier version control)

 GitHub (WEB GUI for Git)

 Discord (Communication)

Login Information

- Admin Account
 - Username: admin
 - Password: admin
- Test account 1:
 - Username: test
 - Password: test
- Test account 2:
 - Username: sheila
 - Password: test123

Installation

In order to run the program there are a few prerequisites that must be met, this section will go through the installation steps.

Prerequisites

- C# Compiler
- MySqlConnection

Running and Compiling

NB: If using the archive provided in the assignment, jump to step 6

1. Download the source code from [mrylund/parking-app](https://github.com/mrylund/parking-app)
2. Extract the downloaded archive
3. Rename “App.config.EXAMPLE” to “App.config”
4. Open App.config and edit the following values to your database information
 - a. HOST
 - b. DATABASE
 - c. USERNAME
 - d. PASSWORD
5. Import “database.sql” file into the database you want to use
6. Compile the program with your C# compiler
7. Log in with following credentials
 - a. Username: admin
 - b. Password: admin
8. You can now add residents if you want.

First considerations

Our first thoughts were, what type of application this should be - desktop or mobile.

We ended up choosing to implement the desktop application first.

This would give us an overview of the project, and make it easier for us to then further down the road, make a mobile app version of the project.

Next up we had to consider what general functions needed to be implemented in the project.

We came up with a list of items, and these are shown below.

Overview of functions to be implemented

- **Login-page**
 - Username
 - Password
 - Login button
 - Forgot login (not prioritised)

- **Landing-page after login for resident**
 - My info
 - My cars
 - Add guest / Remove guest

- **Landing-page for manager**
 - Manage residents
 - *Create resident*
 - *Remove resident*
 - *View guest passes*
 - Check license-plate on car
 - *If parked illegally -> Do action*
 - Overview of vehicles

The UI (User Interface)

For the user interface we would have to consider both the resident and the supervisor. They would need to have different interfaces, as they don't have the same roles.

We started off making sketches of our app to make a visual representation - this gave us both an understanding of how each of us wanted the app to look.

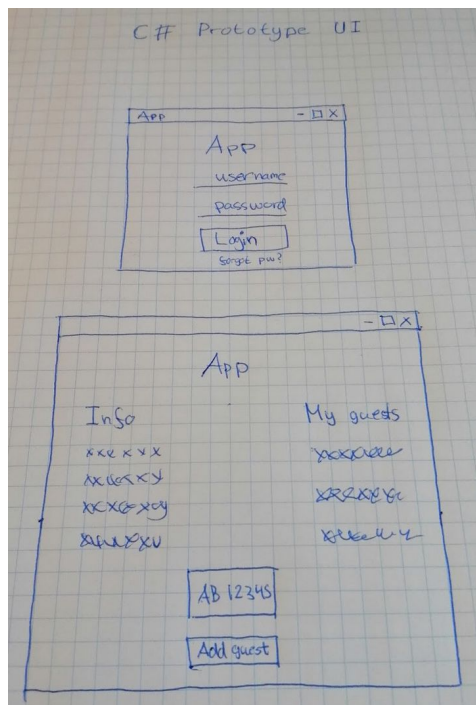


Figure 1: Prototype No. 1 of the App

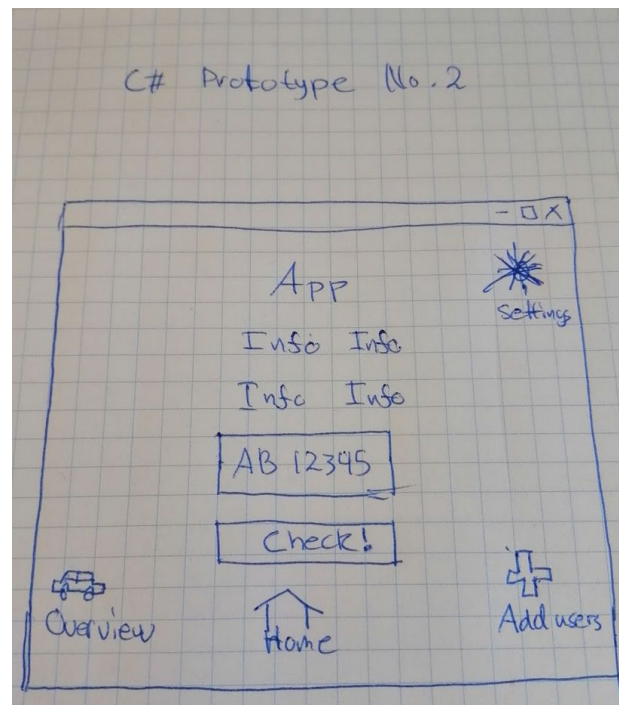


Figure 2: Prototype No. 2 of the App

We made a prototype of the login window (**Fig. 1 and first sketch**) as well as the window a resident is redirected to after they sign in (**Fig. 1 and second sketch**). We wanted the login window to be of a smaller size, because if not it would be too empty for its size.

After signing in as a *resident*, we wanted the resident to be able to see their information as well as manage their guests.

Then we designed a prototype of how the page a supervisor/ inspector would see after logging in, and this can be seen on **Figure 2**.

This window contains the option to look up a license plate and the ability to go to different windows i.e. a window with the ability to give an overview of all residents, their own vehicles and their guest's vehicles. This can be seen on **Figure 3**.

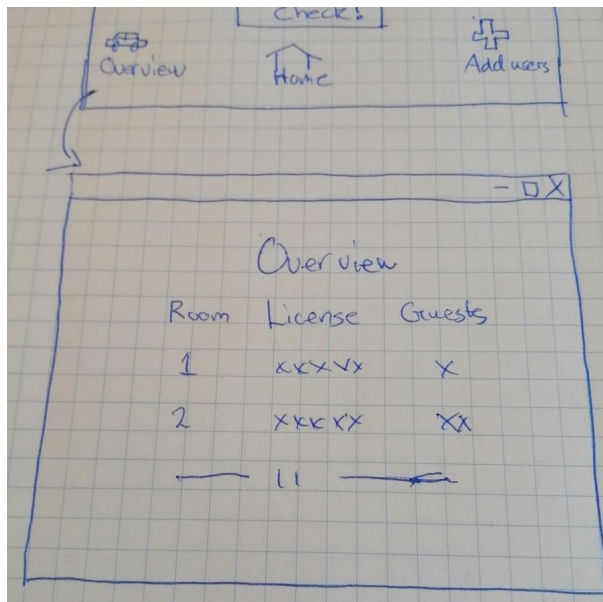


Figure 3: Prototype No. 3 of the App.

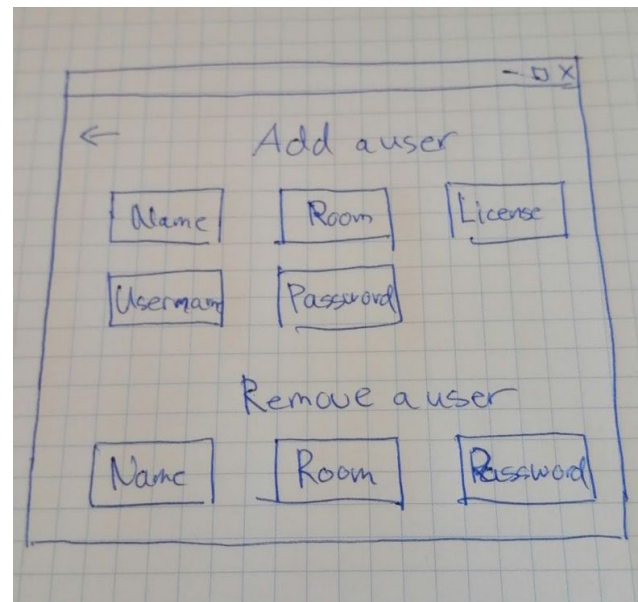


Figure 4: Prototype No. 4 of the App.

Moreover we designed a prototype window of how it would look if the supervisor wanted to add new residents to the program (**Figure 4**).

We also wanted the supervisor to be able to remove a user - incase the resident moved out of the dorm *or* their information was typed in incorrectly. And finally, on the supervisor's home-screen (**Figure 2**), they would be able to type in a license plate to find information about who the car belongs to.

Now all we needed was to start implementing the prototypes into Visual Studio 2019.

The final login window

After implementing the design from the prototypes (**Figure 1**) in Visual Studio 2019, we ended up with results shown on the Figures below (**Fig. 5, Fig 6. and Fig 7**)

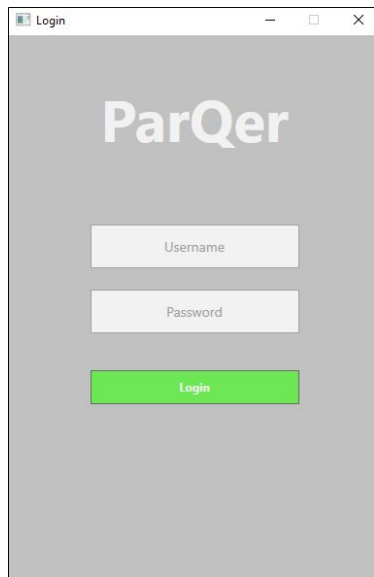


Figure 5: Login window.

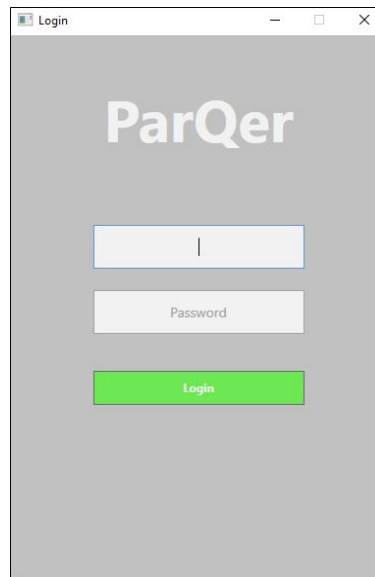


Figure 6: Placeholders.

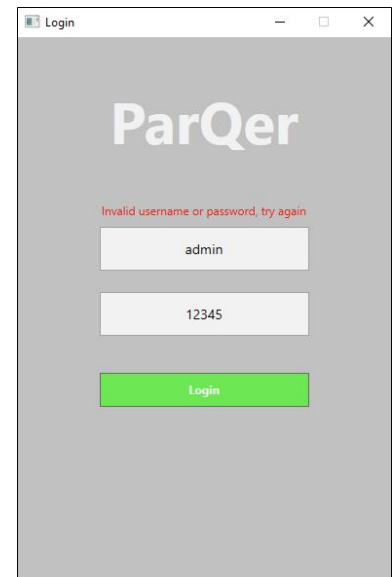


Figure 7: Incorrect login.

The final login window has our 'logo', 2 'textboxes' for user input and a 'login-button'.

To make the user-experience more smooth we have placed 'placeholders' on the 'textboxes'. These placeholders give the user a hint, and can be seen on **Fig. 5** and **Fig. 6**.

If the user manages to fill in the wrong login, we have a 'message' prompting them - this message tells them that they have to give it another try with a different login (see **Figure 7**).

Another thing we have added to make the user-experience better is, that when a user has typed in the login-information, they won't have to press the 'login-button', but can get away with just using the 'enter' key on the keyboard.

The final 'graphical user interface' for a resident

After a resident logs in they are shown the page on **Figure 8**.

There are two subsections on the page - one showing their own information and the other showing information about their guest (the guest(s)'s license plate).

The resident is now able to add up to 4 more license plates (total of 5). We have created this limit, because if every resident has 1 car each, there wouldn't be enough space for all of them to be parked at the dorm. Now since everyone doesn't have a car, there's enough space for cars at the moment.

But to allow visitors for small events the resident might be hosting, we have allowed 5 - this can of course be changed if the 'ParQer app' is going to be used somewhere else with more space for parking.

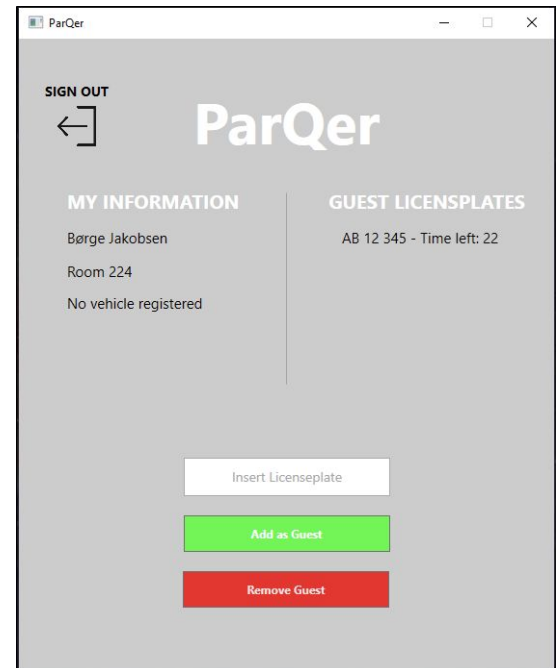


Figure 8: Window for residents.

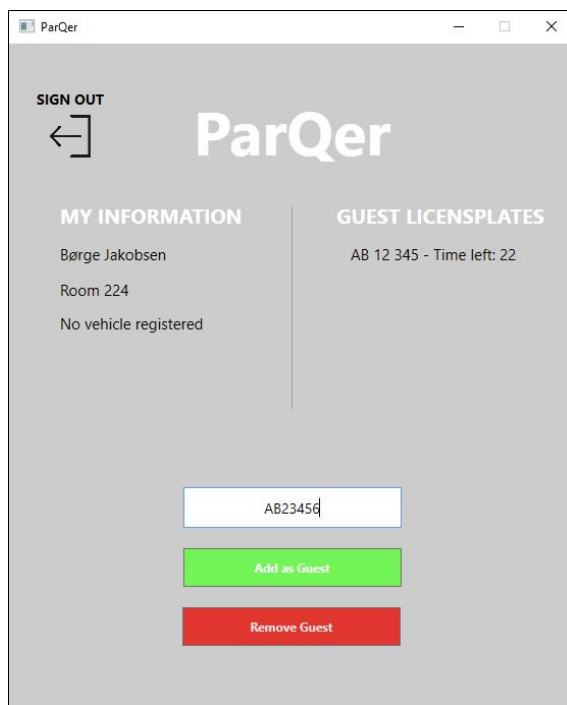


Figure 9: User adding a new guest.

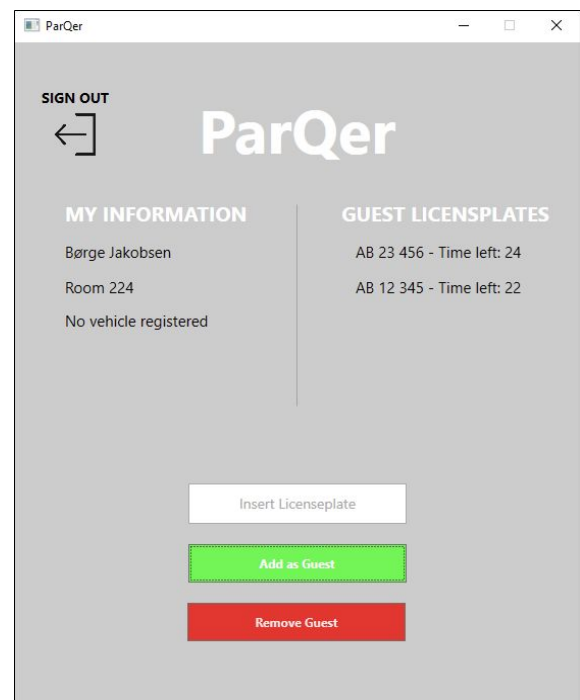


Figure 10: New guest has been added.

The final 'graphical user interface' for a inspector

When the 'admin' *or also called* 'inspector' logs into the system, he is shown the window on **Figure 11**. The inspector is shown a very minimalistic window where he can type in a license plate to search for more information on it. The information fetched from our database is then shown as on **Figure 12**.

On **Figure 12**, we see the result for 'HOURS LEFT' come back with 'R' - this letter 'R' represents a 'resident', whereas if the license plate didn't belong to a resident, it would show that there is '0 hrs' left, and that the 'ROOM NUMBER' was 'Not Found'.

Now depending on where the 'ParQer' app is being used, different actions will be made possible if the vehicle doesn't show up in the system.

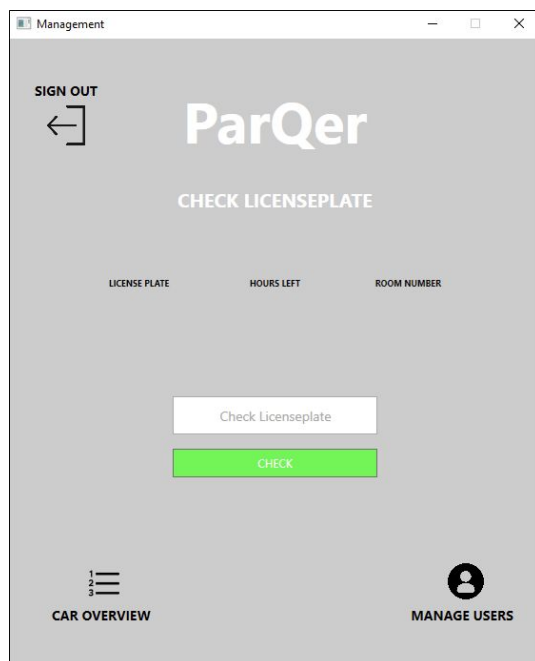


Figure 11: Management 'home' window

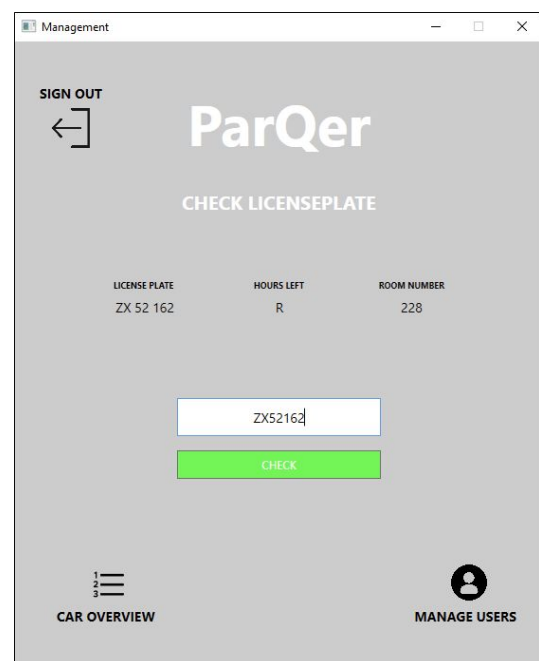


Figure 12: Checking a license plate

Now we have also implemented the prototype design shown in **Figure 4**, this includes the ability to register a new resident in the system, as well as to search up a resident by their room number. The implementation of **Fig. 4** can be seen on **Fig. 13 and Fig. 14**. The window will be shown when the button 'MANAGE USERS' is pressed on the 'home window'.

Comparing these implementations of the GUI shown on **Fig. 11 and Fig. 12** to the sketches we drew (**Figure 2**) you can observe that we no longer have the 'home' button - we removed this because we found it to be redundant.

The window on **Fig. 13** prompts the inspector with a form to be filled out, if they would like to create a new resident. This form is supposed to be filled out by the resident, and then be confirmed by the inspector. The password field will be consisting of '*' when the resident types in their password - this way only the resident will know their password.

The screenshot shows a window titled 'Manage Users'. It has a 'BACK' button with a left arrow. The 'FIND A RESIDENT' section has a 'Room Number' input field, a green 'SEARCH' button, and a red 'REMOVE' button. Below this are labels for 'FIRST NAME', 'LAST NAME', 'USERNAME', 'ROOM NUMBER', and 'LICENSE PLATE'. The 'ADD A RESIDENT' section has input fields for 'FIRST NAME', 'LAST NAME', 'USERNAME', 'PASSWORD', 'ROOM NUMBER', and 'LICENSE PLATE', with a green 'CREATE' button at the bottom.

Figure 13: 'MANAGE USERS' window.

This screenshot shows the same 'Manage Users' window after a search. The 'Room Number' field now contains '224'. The search results are displayed below the 'FIND A RESIDENT' section: 'FIRST NAME' is 'Børge', 'LAST NAME' is 'Jakobsen', 'USERNAME' is 'sup1', 'ROOM NUMBER' is '224', and 'LICENSE PLATE' is 'No vehicle registered'. The 'ADD A RESIDENT' section remains unchanged.

Figure 14: A search made for a resident.

To make the user-experience easier one more time, we have added 'tab-indexes' to the 'textboxes'. This makes it possible to tab between the input-fields instead of moving the mouse. And once again, we have added text 'placeholders' which are removed once the user clicks on the 'textboxes'.


If a resident's information has been input to the system with a typo or some other error, it can be deleted again if the room number is searched up and the 'REMOVE' button is pressed. This can be seen on **Figure 14**.

The Code Implementation

MySQL implementation

As this is a program where a user must have the ability to check if a car is allowed to be parked at a given parking lot while also having different users have the ability to add guest vehicles that should be allowed to be parked at the parking lot, we must have the data stored remotely. Our solution for this was to use a MySQL database to store all residents of a building and the cars they own, on top of this we've also added ability to store guest cars as this was also required for our program to function correctly.

When we use MySQL we have the ability for multiple users to run the same program at different locations and different user accounts at the same time, that way we ensure that residents of a building can add guest vehicles on their own devices while the apartment manager can use his own device to check if cars are allowed to park at a parking lot.

As MySQL is not built into C# we are required to find a external library to connect to our database, we choose to use the  [mysql-net/MySqlConnection](https://www.nuget.org/packages/mysql-net/MySqlConnection) library to aid us in connecting to our database.

Once we've referenced the external library we can use the built in functions to interact with our database, we wrote a couple of functions to make this task a bit easier for ourselves. Namely, we've made an Execute function for all inserts, removals and updates we have to do in our program, we've also made a Select function that will select some data from the database and return the result as a DataTable.

Global Functions

In order to fight redundancy and make the code easier to read we decided to add a static class called global functions, the purpose of this class is to collect all the common functions we use throughout the program at the same place. A good example of this could be the encryption function, we've made a encryption method that applies SHA256 encryption on a string and return the given string, instead of having the same many lines of code to perform this action we made a function so we only use one line to encrypt strings and we also make the program easier to read that way.

We've also added a function to check if a given license plate is in the correct format, a correct format being two letters followed by 5 numbers like the Danish license plates. We've also allowed some form of user error in typing the license plates, we allow any of the following formats: AA XX XXX, AAXX XXX, AA XXXXX and AAXXXXX.

To check if a license plate is valid we are using regex matching, we wrote a simple regex to match license plate types:

```
string licensePattern = "[A-Za-z][A-Za-z]\\s?[0-9][0-9]\\s?[0-9][0-9][0-9]";  
if (!Regex.Match(licensePlate, licensePattern).Success)
```

User class

The whole app builds upon the idea we have residents who are able to add their guests vehicles to the list of allowed vehicles on the parking lot of the building. In order to keep track of the information about the residents we created a User class that will hold all the information about any given user and their guests, that way it is way easier for us to keep track of the information about a user. When doing this it is also easy to display the information on the users own homepage, currently we allow the users to see their name, license plate, room number and a list of active guest passes they have given out. In addition to just storing information about the given user, we've also implemented functions in the class that allows the user to remove and add guests to themselves, these are made as functions of the user class to make it easy to add and remove the guests from a specific user.

Further work

Mobile App

To make the work for the supervisor/inspector, a mobile app version of the desktop app would be of help.

The plan would be to implement a License plate recognising (from a picture). This would mean that the inspector only would need to take a picture of the license plate for it to search up information about the vehicle (and it's owner).

This mobile app would have to be implemented with Xamarin.

Overview of all vehicles

A view of a type List of all the cars belonging to the residents would be a nice-to-have for the supervisor/inspector. This feature (see **Figure 3**) will be implemented in future updates to the software.

Adding ownership of multiple cars

Later down the road, adding the possibility to have more than one car as a resident would be possible.

This feature would not be a big deal at our Student Dorm, but would be a nice-to-have for other, and possibly bigger residencies.

Conclusion

We can conclude this project by reflecting over the development we have made during the making of this application.

We have developed a desktop-application for managing parked vehicles outside a residence. This app makes it easier for the inspectors by giving an overview of all the vehicles parked outside the dorm.

We've managed to create a somewhat minimalistic design - by minimalistic we are implying that there are no unnecessary features implemented in the GUI, and that the user-experience is made better by the small details (i.e. the placeholders for the textboxes and the possibility to tab around the forms).

Of course there are some improvements to be made, which are mentioned in the section above, but again, this can be implemented later down the road.