

## Laboratory 2

### Car-Sharing (2 / 2)

#### *Agenda*

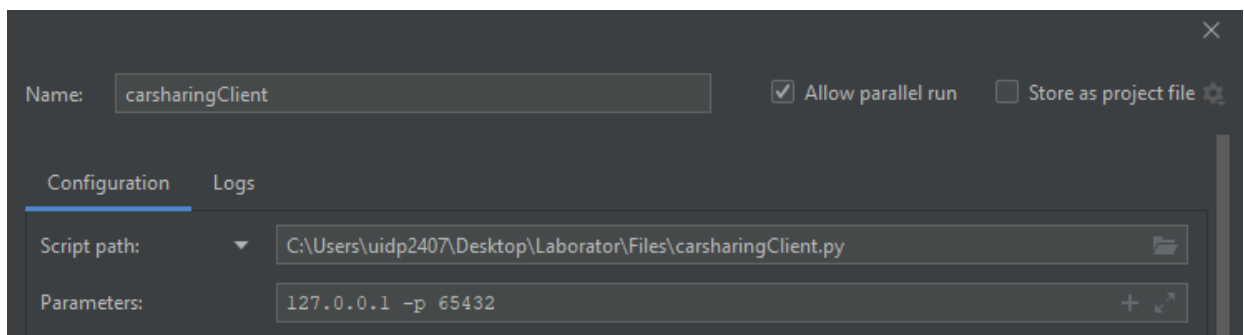
Introduction .....	2
Preconditions .....	5
Tasks .....	6
Evaluation .....	7

## Introduction

This laboratory aims to implement more of the features from car-sharing application. In the first laboratory we implemented a carsharing client and server which were able to exchange start-rental and end-rental messages. The server validated the request from the client by checking if the car requested for rental is in the list of authorized cars and that it is available for rent. We also added a graphical interface in the element to signal if the client is authorized and the rental procedure began successfully.

Important! The starting point for our application today is the complete `carsharingClient.py`, `carsharingServer.py` and `clients.json` from moodle. The latest sources contain the complete solution from Laboratory 1 and some extra functionality added to resolve the tasks from Laboratory 2.

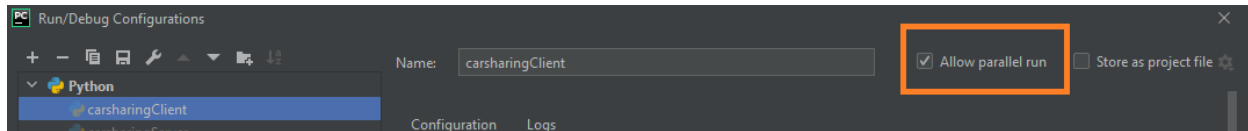
Once again please configure PyCharm to run your scripts with these parameters in order to have a TCP address/port setup for our applications:



Our server is not yet complete. It is currently able to receive start-rental and end-rental requests but it is not yet able to perform any validation of clients sending the request. Knowing the identity of the client can be useful for various reasons such as checking if the client has a valid driver's license, legal age to rent a car. In this laboratory, we would like to extend our server and client to have the following extra capabilities:

- The server will be able to accept connections from multiple clients and differentiate between them
- The clients should provide authentication data when connecting to server
- The server will restrict a client from renting more than a car
- The server may also check if the doors are locked, engine stopped and fuel level is proper before ending the rental process.

In order to be able to work multiple clients, we should first configure our PyCharm to allow starting multiple carsharing clients at once. In order to do this edit the client configuration to allow parallel run.



In order to be able to use the car-sharing service, a person must create an account using a phone application. We will simulate this by adding a new form window in our client application

The mandatory requirements are usually:

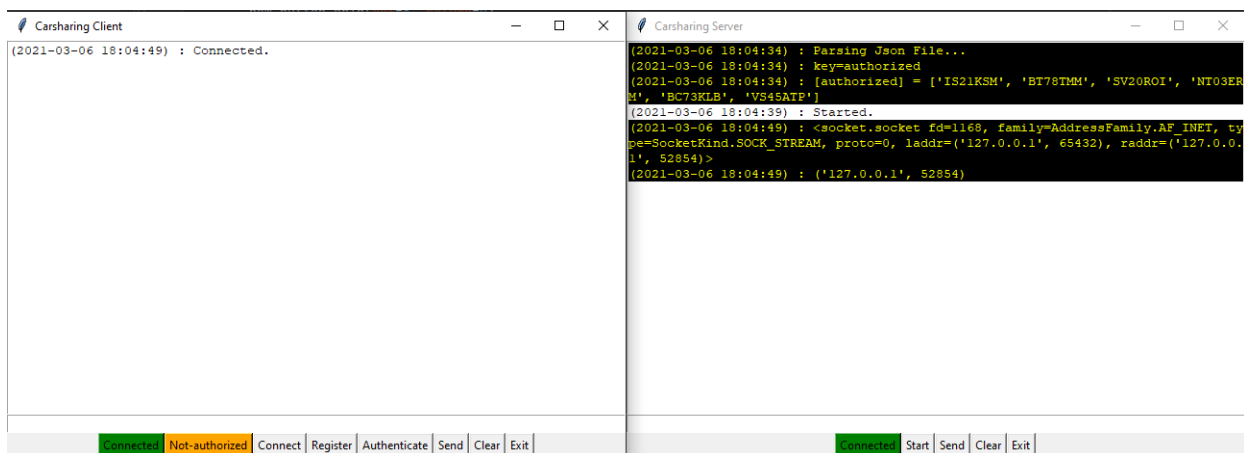
- Name
- Identification details
- Valid driver license – photograph is usually required to be uploaded in the application, but in our application we will only use an ID
- Payment method / Credit card
- Email address
- Mobile Phone Number
- Agreement of the 'Terms and Conditions' provided by the company

The account is validated by the backend by performing various checks on the data received. Based on this validation, the backend will authorize or not the driver for renting the car.

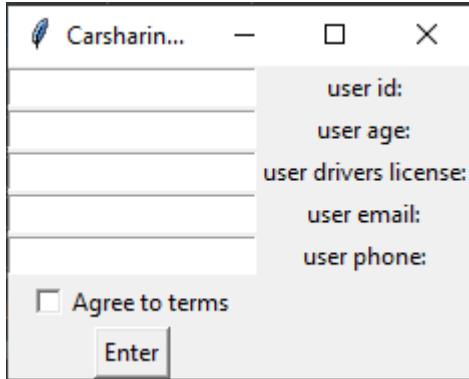
Please be aware that the authorization may also be revoked in various cases specified in the 'Terms and Conditions'. One example would be severe damage to the car during a car-drive.

The backend may also keep a history of the driver's car rides.

**Important!** The updated client from moodle already contains changes in order to collect user data from customer. If you run the client you will notice that a two new buttons, register and authenticate are present



Register button should be used in order to collect the user input data. A new window will appear.



Carsharin...

user id:

user age:

user drivers license:

user email:

user phone:

☐ Agree to terms

Enter

The data you complete in this field will be saved in these variable of class Client:

```
self.userId = ""  
self.userAge = ""  
self.userDriverLicense = ""  
self.userEmail = ""  
self.userPhone = ""
```

### *Preconditions*

You will need to have the updated code from moodle.

## Tasks

Add the following functionalities:

- a) Registration / Authentication of client to server part 1. The following requirements should be implemented:
  - Client should not be able to initiate a connection to server before client data is registered ( Register button is pressed and form is completed and validated). **Hint!** a flag should be set to true when user input is validated ( else condition below). That flag should be used to restrict/admit communication.

```

if user_id_entry.get() == "" or user_age_entry.get() == "" or user_drivers_license_entry.get() == "" or user_email_entry.get() == "" or user_phone_entry.get() == "":
    print_system_notification('Data is incomplete. Please complete all fields')
else:
    self.userId = user_id_entry.get()
    self.userAge = user_age_entry.get()
    self.userDriverLicense = user_drivers_license_entry.get()
    self.userEmail = user_email_entry.get()
    self.userPhone = user_phone_entry.get()
    print_system_notification('Successfully registered client data: user id:' + self.userId + ' user age:' + self.userAge + ' user drivers license:' + self.userDriverLicense + ' user email:' + self.userEmail + ' user phone:' + self.userPhone)
    popup.destroy()

```

- When Authenticate button is pressed, client should send an authentication request to server. The request should have the form: ***"authenticate ID Age DriversLicense Email Phone"***. **Hint!** The slot connected to Authenticate button is the method authenticate in class client. All logic for sending the authentication message should be written there. In order to concatenate data obtained from user input you can use format functionality: result = "{} {} {} {}".format(str1, str2, str3, str4)

```

def authenticate(self):
    print_system_notification('authentication ongoing')

```

- b) Registration / Authentication of client to server part 2. The following requirements should be implemented
  - When server receive the authenticate message, it should send a confirmation to the client if the customer id is not in the list of restricted customers - the list is parsed at server startup from a json file and it is saved in Server class variable restricted\_customers. If the client is restricted, server should send an error message and not accept further commands from the client. The server should not accept any commands ( eg. Start-rental, End-rental ) from client until authentication is complete. If such commands are issued by client before authentication is done, error messages should be sent to client.
- c) Server should contain a database of all clients. The database should contain the following data for each active rental ***ID Age DriversLicense Email Phone***. Requirements:
  - Create a new button named Show Customer. When a user id is completed in the entry of the server ( same entry where we write messages), all data about customer should be printed in server chat. If there is no active rental for the required id a error message should be printed

*Evaluation*

Score:

- a) 1.5 point(s)
- b) 1.5 point(s)
- c) 1 point(s)