**Web User Interface**

Als gebruikersinterface is er voor een web-interface gekozen met Flask, een Python module voor web-development.

**Ontwerp**Een gebruikersvriendelijk en simpel ontwerp is gekozen voor de beste resultaten. De eerste stap was om dit op papier te krijgen, dus een prototype werd gemaakt. Toen het klaar was en iedereen akkoord ging met het ontwerp was het onderzocht wat het beste manier was om het user interface te maken.

**User Interface**Voor het gebruikersinterface was er eerst voor Tkinter gekozen, maar kort daarna werd er gerealiseerd dat het niet de beste optie was voor onze ontwerp. Het Tkinter module is niet geschikt voor een simpel, gebruikersvriendelijk- en mooi ontwerp want het had een “out-dated” GUI. Na het onderzoeken van andere mogelijkheden heeft groepslid Bob Thomas gesuggereerd om een web-based interface te maken met de Flask module.

**Back-end**Het Flask module werkt door “route.py” te maken die als het centrale bestand werkt. Alle custom-modules worden in dit bestand geïmporteerd en met elkaar gekoppeld om later in het gebruikersinterface weer te geven. De Flask “route.py” genereerd ook het HTML, CSS en Javascript structuur. Alle “transacties” tussen de bestanden gebeurt in het “route.py” bestand.

Het python script in het “route.py” bestand kijkt of het knop “IN-GARAGE” of “UIT-GARAGE” ingedrukt is. Als er een auto geselecteerd is en het “IN-GARAGE” knop ingedrukt, dan wordt de afbeelding van het kenteken gelezen met het OpenAlpr module, dan naar het RWD module database gestuurd en het resultaat word opgevangen en beveiligt met PyCrypto, waar het dan in de database wordt bewaard.

Als het “UIT-GARAGE” knop ingedrukt is dan wordt het geselecteerde auto met behulp van Javascript verwijderd van de garage.

Ook worden HTML, CSS en Javascript gebruikt. HTML wordt gebruikt voor het structureren van de webpagina, CSS voor de positie en stijl van de elementen in de pagina en Javascript voor het selecteren, weergeven en transacties met de database in de “route.py” bestand.

**Route.py Code**Het Flask module is in een variabel gezet met de naam app en wordt gebruikt door:

app = Flask(\_\_name\_\_)

Daarna worden er twee functies gemaakt een voor het gebruikers interface pagina en een voor het JSON bestand die er wordt gehaald door het RDW API.

@app.route('/', methods=['GET', 'POST'])  
**def home**():

De webpagina locatie wordt gespecificeerd met the methoden ‘GET’ en ‘POST’ om het halen van data op de webpagina.

**if** request.method == 'POST':

Kijkt of er een knop werd ingedrukt of niet, met de methode “POST”.

**if** request.form['action'] == "IN-GARAGE":

Kijkt of het ingedrukte knop “IN-GARAGE” is.

**if** request.form['car-action'] == "park":  
 plate = automatic\_plate\_recognition.get\_plate(os.path.join(os.getcwd(), request.form['id']))

Kijk of het auto moet geparkeerd worden, dan haalt het kenteken van de auto van het afbeelding.

**if** plate != 'error':  
 car\_info = api.request\_information(plate)

Als het kenteken goed door het API gelezen was dan wordt het naar het RDW API gestuurd om het informatie over die auto te halen.

**if** car\_info != "No data found":  
 encoded = encryptor.encrypt(str(car\_info))  
 status = "PARKED" **if** car\_info['parkerentoegestaan'] == "Ja" **else** 'VIOLATION'  
 database.register\_parking(plate, encoded.decode("UTF-8").rstrip('{'), request.form['id'],  
 status)  
**else**:  
 error = "Car not found in government api"

Kijkt of er geen error gevonden is, daarna word het informatie doorgestuurd naar de encryptie module om het te beveiligen en geeft de auto een status van “PARKED’ als het toegestaan is.

**elif** request.form['action'] == "UIT-GARAGE":  
 **if** request.form['car-action'] == "remove":  
 plate = request.form['id']  
 database.finish\_parking(plate)  
 **else**:  
 error = "Car not in parking"

Als de knop “UIT-GARAGE” ingedrukt is dan wordt de geselecteerde auto van de garage verwijderd en weer in de wachtlijst gezet.

cars = {  
 'all': database.get\_all\_cars(),  
 'parked': database.current\_cars\_parked()  
}  
car\_parked = {  
 'all': database.get\_all\_cars\_history()  
}

Dictionaires gemaakt van de database module om het makkelijker te maken voor het importeren van de informatie.

JSON RDW API bestand:

@app.route('/car/<id>')  
**def car**(id=**None**):  
 **return** jsonify(database.get\_car\_by\_plate(id))

**if** \_\_name\_\_ == "\_\_main\_\_":  
 app.run(debug=**True**, port=666)

Begint de applicatie met debugging mode en gespecificeerd op port 666

**if** \_\_name\_\_ == "\_\_main\_\_":  
 app.run(debug=**True**, port=666)

Begint de applicatie met debugging mode en gespecificeerd op port 666

**Applicatie workflow**

