Voortgangsverslag

B1. Parkeergaragebeheer



**Gemaakt door**: Bob Thomas, Zahyr Seferina, Sean Lieuw, Leroy Wallenburg, Robbie Valkenburg

**Klas**: ICT-V1S

**Project-groep**: 2S

**Datum**: 14-10-2015

**Versie**: 0.1

Inleiding

Het project is tot stand gekomen omdat het sinds 2015 is verboden om met auto’s die als hoofdbrandstof diesel en een datum van eerste afgifte van vóór 2001 hebben het centrum van Utrecht in te rijden.

Om dit beter te handhaven, zal er software geschreven worden. De software zal code beschikken om kentekens te herkennen vanuit foto materiaal. Eenmaal uitgelezen, zal er de bijbehorende informatie van het kenteken worden opgezocht. De software zal vanuit de gegeven informatie de conclusie trekken of de auto in overtreding is.

In het rapport zal duidelijk worden hoe het project gerealiseerd gaat worden. In het rapport zullen o.a. contactgegevens, taakverdelingen en technische uitwerkingen zijn beschreven voor het project.

Inhoud

[Contactgegevens 4](#_Toc432726984)

[Rolverdeling 4](#_Toc432726985)

[Taakverdeling 5](#_Toc432726986)

[Technische uitwerkingen 6](#_Toc432726987)

[Algemeen 6](#_Toc432726988)

[Flask 6](#_Toc432726989)

[Openalpr 6](#_Toc432726990)

[RDW 6](#_Toc432726991)

[Crypto.Cipher <module algorithm > AES 6](#_Toc432726992)

[Sqlite3 6](#_Toc432726993)

[Web User Interface 7](#_Toc432726994)

[Ontwerp 7](#_Toc432726995)

[UserInterface 7](#_Toc432726996)

[Back**-**end 7](#_Toc432726997)

[Route.py Code 8](#_Toc432726998)

[Applicatie workflow 10](#_Toc432726999)

[Plate\_recognition(): 11](#_Toc432727000)

[Introductie 11](#_Toc432727001)

[Toepassing 11](#_Toc432727002)

[Uitwerking 11](#_Toc432727003)

[RdwAPI(): 13](#_Toc432727004)

[Introductie 13](#_Toc432727005)

[Toepassing 13](#_Toc432727006)

[Uitwerking 14](#_Toc432727007)

[Encryptor(): 17](#_Toc432727008)

[Introductie 17](#_Toc432727009)

[Toepassing 17](#_Toc432727010)

[Uitwerking 18](#_Toc432727011)

[Parking\_lot\_database(): 20](#_Toc432727012)

[Introductie 20](#_Toc432727013)

[Toepassing 20](#_Toc432727014)

[Uitwerking 21](#_Toc432727015)

[ERD uitwerking 22](#_Toc432727016)

[PSD Uitwerkingen 22](#_Toc432727017)

Teamgegevens

## Contactgegevens

|  |  |  |
| --- | --- | --- |
| **Naam** | **Emailadres (@student.hu.nl)** | **Telefoonnummer** |
| Teamlid 1 | [robbievalkenburg.valkenburg@student.hu.nl](mailto:robbievalkenburg.valkenburg@student.hu.nl) | 0631159083 |
|  |  |  |
| Teamlid 2 | [zahyr.seferina@student.hu.nl](mailto:zahyr.seferina@student.hu.nl) | 0623466780 |
| Teamlid 3 | [sean.lieuw@student.hu.nl](mailto:sean.lieuw@student.hu.nl) | 0640288082 |
| Teamlid 4 | [leroy.wallenburg@student.hu.nl](mailto:leroy.wallenburg@student.hu.nl) | 0641460283 |
| Teamlid 5 | [bob.thomas@student.hu.nl](mailto:bob.thomas@student.hu.nl) | 0629440082 |

## Rolverdeling

|  |  |  |
| --- | --- | --- |
| **Rol** | **Beschrijving** | **Naam / Namen** |
| Teamleider | Leidt de vergaderingen. Houdt bij wie welke taak doet, en of alles op tijd wordt opgeleverd. | Bob |
| Analist | Analyseert het probleem/ de taak, en maakt vervolgens een globaal overzicht van de applicatieworkflow. | Sean |
| Tester | Test of de applicatie daadwerkelijk doet wat deze moet doen: Komt het overeen met de opdracht en de applicatieworkflow? | Zahyr |
| Teamlid | Programmeert delen van de applicatie | Leroy, Robbie |

## Taakverdeling

|  |  |  |
| --- | --- | --- |
| **Taak** | **Beschrijving** | **Naam / Namen** |
| GUI | Programmeren van een gebruiksvriendelijke interface voor de gebruiker. | Zahyr |
| Kenteken uitlezen | Programmeren van kenteken herkenning in foto’s. Volgens kenteken in foto resulteren in tekst. | Leroy |
| Verkrijgen van voertuiggegevens | Het ophalen van de voertuiggegevens die overeen komen met het kenteken. | Bob |
| Encryptie voor gegevens | Een encryptie maken voor de gegevens die opgeslagen zullen worden. | Sean |
| Opslag | Het realiseren van een opslag voor de gegevens. | Robbie |

# Technische uitwerkingen

## Algemeen

De software zal grotendeels geschreven worden met Python versie 3.4. Hiervoor zullen de volgende libraries, api’s, modules en database voor gebruikt worden:

### Flask

Met deze module voor Python is het mogelijk om een web interface te maken voor je Python software.

### Openalpr

Dankzij Openalpr kan vanuit een foto een kenteken worden herkend en dit als *plain tekst* geresulteerd worden.

### RDW

Via de RDW-api kunnen voertuiggegevens opgevraagd worden die overeen komen met het gegeven kenteken. Hiervoor is een api-key nodig.

### Crypto.Cipher <module algorithm > AES

Voor het encrypten van onze gegevens zal Crypto.Cipher gebruikt worden. Hiermee waarborgen wij onze veiligheid op gegevens.

### Sqlite3

Zoveel mogelijk gegevens zullen worden opgeslagen in een database. Hiervoor wordt een sqlite3 database gebruikt. Verder in het rapport zal er een ERD geïllustreerd zijn over de structuur van de database.

## Web User Interface

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

OntwerpEen 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.

UserInterfaceVoor 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**-**endHet 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 CodeHet 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))

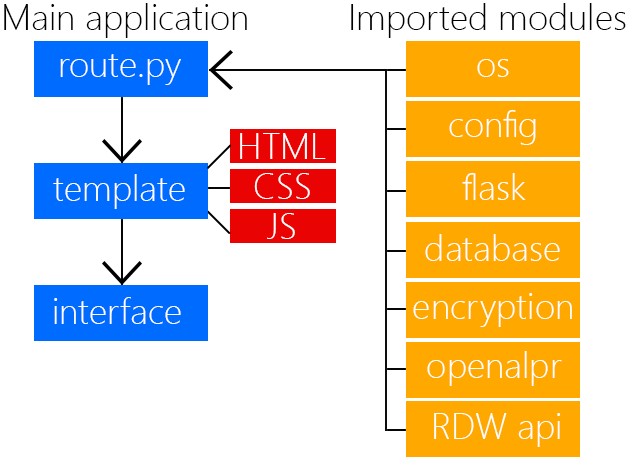
**if** \_\_name\_\_ == "\_\_main\_\_":  
 app.run(debug=**True**, port=666)

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



## Plate\_recognition():

### Introductie

De module plate\_recognition.py bevat een klasse AutomaticPlateRecognition().

Deze klasse heeft één functie: get\_plate(self, file).

De werking van de klasse in grove lijnen is in de vorm van een flowchart te vinden aan het einde van dit document.

### Toepassing

Om de klasse te gebruiken moet er eerst een instantie gemaakt worden.

kenteken = AutomaticPlateRecognition()  
  
Vervolgens moet de functie get\_plate(self, file) worden aangeroepen en voor file moet het pad naar de in te lezen afbeelding worden gegeven.

kenteken.get\_plate(“locatie”)

### Uitwerking

Als de functie get\_plate() wordt aangeroepen dan wordt er binnen die functie eerst een variabele ‘plate’ gemaakt: plate = ''

Vervolgens wordt alpr.exe gedraaid en krijgt het imagepad mee. De uitvoer van alpr.exe wordt dan in de variabele plate gezet.

De uitvoer van alpr.exe, als er een kenteken wordt gevonden, ziet er als volgt uit:  
  
output.png

Een aantal resultaten met een “confidence rating” erachter. Dit betekend hoe zeker alpr er van is dat het betreffende kenteken de juiste is.

Als er geen kenteken wordt gevonden is de output “No license plates found”.

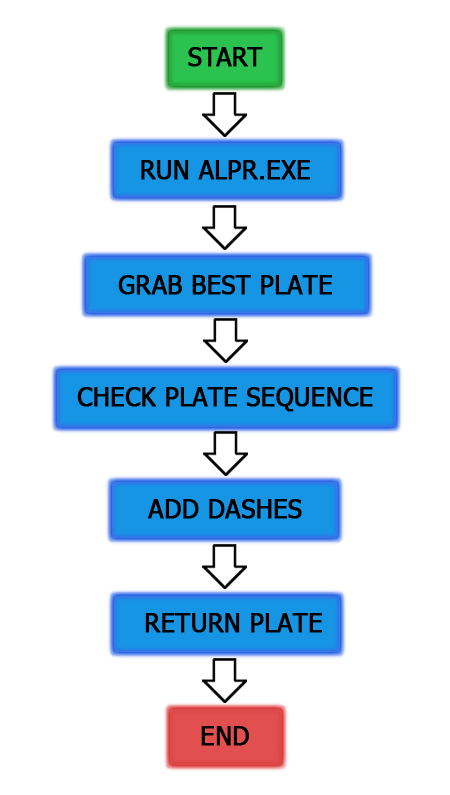
Daarom de volgende if-statement:

Als het woord “found” in plate wordt gevonden dan betekend dat dat er geen kenteken is gevonden. In dat geval geeft de functie “error” terug.

Als het woord “found” **niet** in plate wordt gevonden dan betekend dat, dat er wel een kenteken is gevonden en dan wordt de else uitgevoerd.  
In else wordt een nieuwe variabele gemaakt: number\_plate.

Uit plate wordt het kenteken met de hoogste confidence rating gehaald en in number\_plate gezet. Dan wordt er in nog een if-statement gekeken wat de volgorde van de plaat moet zijn (bijvoorbeeld xx-xx-xx of xx-xxx-x) en worden er op de juiste plaatsen streepjes gezet.

Als laatste returned de functie het juiste kenteken.



*figuur 1: de klasse AutomaticPlateRecognition() in flowchart vorm.*

## 

## RdwAPI():

### Introductie

De module genaamd rdw\_api.py bevat een classe met de naam RdwApi.

Deze class word gebruikt om met behulp van de nummerplaat informatie op te vragen bij welke auto die hoort.

### Toepassing

Om de class te kunnen gebruiken moet je een API key aanvragen bij <http://overheid.io>.

Als de API key verkregen is moet je die toevoegen aan de config.py zoals volgt:

API\_KEY = “Voeg de API key binnen de aanhalingstekens .”

Eenmaal dat gedaan is mag er gebruik gemaakt worden van de volgende functie:

api = RdwApi(config.API\_KEY)

Om informatie te kunnen opvragen van een auto en zijn bijbehorende nummerplaat gebruik je:

api.request\_information(‘nummer\_plaat’)

Als er informatie beschikbaar is van de nummerplaat in de database van overheid.io word er een dictionary teruggegeven. De informatie verschilt per auto. Bij afbeelding 1.1 is er een voorbeeld gegeven (zie afbeelding 1.1).

### Uitwerking

De volgende uitwerking word op een simpel manier uitgelegd hoe de code werkt en in elkaar zit.

Het moment dat je api = RdwApi(config.API\_KEY) gebruikt word de \_\_init\_\_ functie in de class aangeroepen. Die zorgt ervoor dat de api key aan de class gebonden wordt en kan er overal gebruik worden gemaakt van het “self”-keyword.

De class bestaat uit 2 functies waarvan 1 een helper functie is en gebruikt word om de api aan te sturen.

#### vehicle\_allowed(auto informatie)

De helper functie genaamd “vehicle\_allowed” word gebruikt om te kijken als de auto binnen de reglementen valt of die toegestaan is om hier te parkeren.

De regel geldt dat als er een auto is die als een datum van eerste afgifte voor 2001 heeft en op diesel brandstof rijdt dan is hij niet toegestaan om hier te parkeren en word dit aangegeven in de database.

In de comments van de code word dit ook uitlegt.

#### request\_information(nummer plaat)

De 2e functie is genaamd “request\_information” en deze word gebruikt om via een nummerplaat informatie op te vragen via de API.

De url die word aangeroepen is: <http://overheid.io/api/voertuiggegevens/+nummerplaat>.

Als de auto’s gegevens in de database zijn opgeslagen dan geeft die dat terug met een JSON (zie afbeelding 1.1).

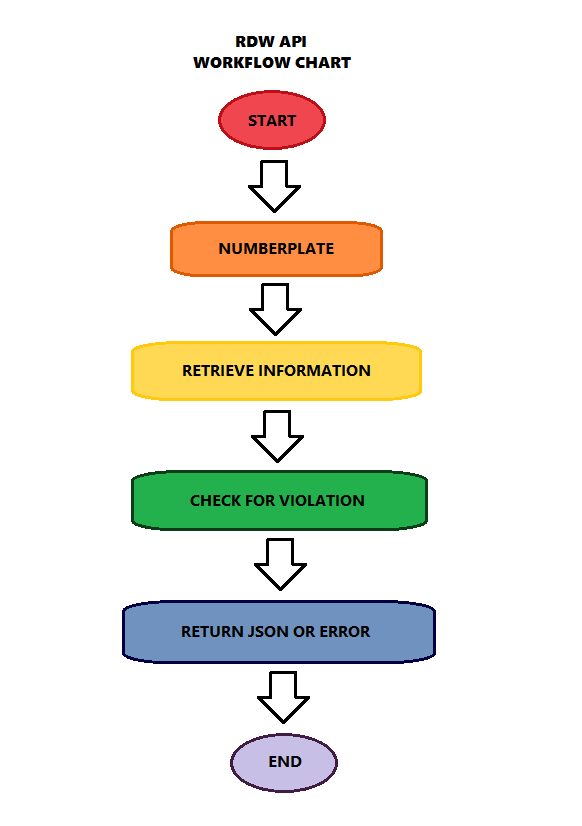
Met deze gegevens kan “vehicle\_allowed” zijn functie uitvoeren en krijgt die een extra key value pair met:

Parkerentoegestaan: “Ja” of “Nee”.

Als er geen data is gevonden geeft die een string “No data found” terug en word de functie “vehicle\_allowed” ook niet uitgevoerd.



Afbeelding 1.1



Afbeelding 1.2

## Encryptor():

### Introductie

De module genaamd encryptor.py bevat een class met de naam Encryptor.

Deze class word gebruikt om met behulp van een computerversleutelingstechniek die AES (Advanced Encryption Standard) heet de nummer plaat informatie te encrypten en decrypten.

### Toepassing

Er zijn twee verschillende soorten functies beschikbaar in de class Encryptor(). De eerste functie is genoemd “encrypt()”en die zorgt voor de encryptie.

Om de functie te kunnen gebruiken heb je eerst de nummer plaat informatie van een auto nodig. Natuurlijk is het ook mogelijk om je eigen gegevens in de functie te gebruiken, maar het uiteindelijke doel zal zijn voor het encrypten van de nummer plaat gegevens.

Eenmaal de nummerplaat gegevens bekend zijn zal hij moeten worden toevoegd bij de module route.py, in de home()-functie, binnen de laatste if-statement waar er staat:

encoded = encryptor.encrypt(str(car\_info))

De informatie moet worden geplaatst waar er “car\_info” staat. Daarna worden de gegevens verwerkt.

De tweede functie is genoemd “decrypt()” en die zorgt voor de decryptie.

Om deze functie te gebruiken heb je eerst de secret key nodig. Er is gebruikt gemaakt van een eigen secret key in plaats van een randomly generated key, omdat het uiteindelijk handiger was. De secret key hoeft dan niet constant te worden verandert per encrypted data. De secret key is in de module route.py en als volgt uit: SECRET = bytes(b’\*\xd0c\xaeIz\xa3<Y\xf7O\x14\x9el\xc4\xfa’).

### Uitwerking

Het moment dat je encryptor = Encryptor() aanroept word er ook gebruik gemaakt van de \_\_init\_\_ functie. Die zorgt ervoor dat de volgende functies binnen de class kunnen worden gebruikt met behulp van het “self”-keyword:

- encoded  
- decoded  
- secret  
- block\_size  
- padding

Zoals eerder vermeld bestaat de class uit twee verschillende functies: een encrypt-functie en een decrypt-functie.

#### encrypt(self, private\_info)

De encryptie AES die gebruikt word binnen deze functie is geimporteerd via Crypto.Cipher en om die te kunnen krijgen moet je “pycrypto” downloaden in je Python. AES maakt gebruik van een precies hoeveelheid aantal bytes. Dit betekent dat als er één bit te weinig is tijdens het encrypten dan lukt het hele proces niet. Daarom word er gebruik gemaakt van padding. Die zorgt ervoor dat de bytes precies genoeg zijn om te kunnen encrypten door de afwijkende bits erin te voegen.

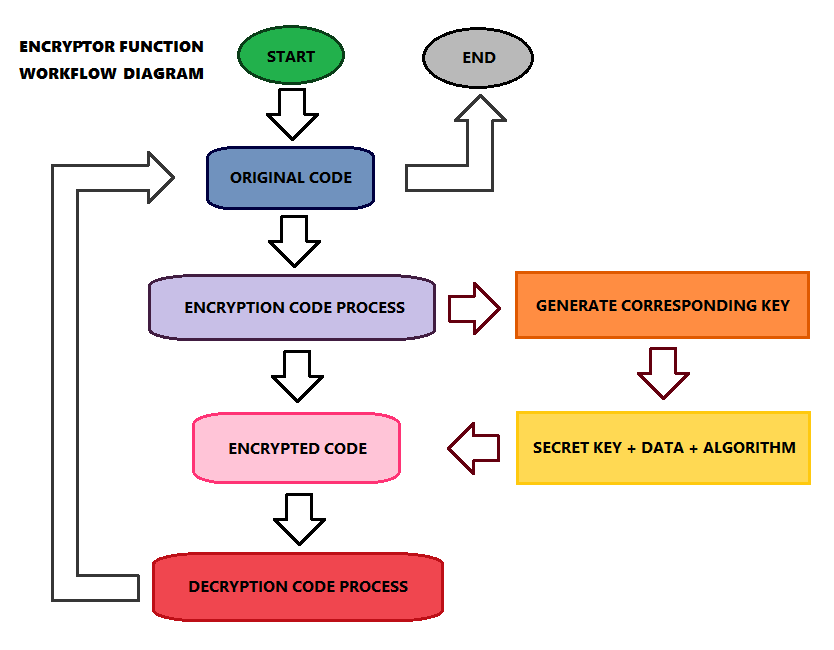
Er is ook gebruik gemaakt van de geïmporteerde base64 die helpt met het proces van “encoding.” Dat is niet te vergelijken met encrypten. Informatie kan niet zomaar worden encrypted en moet eerst in een bepaalde manier worden verandert en opgeslagen voordat die proces begint. Dat heet encoding en met behulp van base64 is dat mogelijk om te doen.

De algoritme die gebruikt word voor het encrypten is opgeslagen in de functie “cipher”. Binnen die functie word de zelf gekozen secret key geïmplementeerd, waardoor er een algoritme word gemaakt die samenhangt met diezelfde key.

#### decrypt(self, encrypted\_string)

De decrypt()-functie gebruikt veel van dezelfde processen als de encrypt()-functie en is er hierdoor weinig te vertellen, behalve de verschillen tussen de twee functies.  
Tijdens het decode-proces moet eerst de padding worden weggehaald of populairder gezegd “weggestript”. Daardoor heet de functie ook .rstrip. Python kan de functie .rstrip niet herkennen en is er daarom gebruik gemaakt van .decode(UTF-8) om dat wel mogelijk te maken. De padding moet natuurlijk weggehaald worden omdat de ingevoerde bits anders namelijk de decryptie kunnen verstoren en een foute uitvoer afgeven.   
De key die gebruikt word is natuurlijk ook de zelf-gekozen secret key, afkomstig uit de route.py module.

Op de volgende pagina is een workflow-chart gemaakt die op een simpel manier de proces aanduidt (zie afbeelding 1.2).



Afbeelding 1.2

## Parking\_lot\_database():

### Introductie

De module genaamd parking\_lot\_database.py bevat een class met de naam ParkingLotDatabase.

Deze class dient gebruikt te worden om gegevens over de auto of over geparkeerde auto’s op te slaan in een database. Met de class kan je zowel gegevens opslaan als gegevens ophalen.

### Toepassing

Bij het aanroepen van de class dienen er geen argumenten te worden meegegeven aan de constructor. In de constructor zal er met de database ‘parking\_lot.db’ een connectie gemaakt worden. Ook de benodigde tabellen zullen aangemaakt worden als deze nog niet bestaan.



Afbeelding 1.1

### Uitwerking

#### create\_car(self)

Maakt de tabellen in de database indien ze nog niet gemaakt zijn. Deze functie wordt altijd in de constructor aangeroepen.

#### register\_parking(self, license\_plate, info, image\_path, status)

Update in de database tabel ‘car’ de nieuwe informatie (license\_plate en info) waar de foto locatie gelijk is aan image\_path . Ook wordt in de tabel ‘car\_parked’ een nieuwe rij toegevoegd met : kenteken plaat (license\_plate), status(status) en het aantal huidige seconde toe.

#### current\_cars\_parked(self)

Resulteert een lijst met de auto’s die op het moment geparkeerd staan. Dit doormiddel van een query aan de ‘car\_poarked’ tabel waar alles wordt geselecteerd waar geen eind tijd is gezet (end\_time).

#### get\_all\_cars(self)

Resulteert een lijst van alle rijen die in de ‘car\_parked’ staan.

#### get\_all\_cars\_history(self)

Resulteerd een lijst met alle auto’s die in de tabel ‘car’ staan.

#### finish\_parking(self, license\_plate)

Update de rij in de tabel ‘car\_parked’ en zet het veld `end\_time` op het huidige aantal seconden waar het veld `license\_plate` gelijk is aan variable license\_plate.

#### get\_car\_by\_plate(self, license\_plate)

Resulteert een dictonary met gegevens over de auto passend bij het kenteken (license\_plate).

#### create\_array\_dict(self, cars, parked=False)

Resulteert een voledige dictonary lijst met alle bijhorende auto informatie. In de variabel parked dient meegegeven te worden of het om auto’s gaat die al geparkeerd staan. Dit is nodig voor verdere opbouw in de code.

#### create\_parked\_car\_dict(self, car)

De variable car bevat een de zelfde gegevens die een rij kan hebben in de tabel ‘car\_parked’. Met die gegevens wordt volgens vanuit de ‘car’ tabel de bijhorende gegevens toegevoegd. Uiteraard wordt in deze functie een decryptie gebruikt om de variabel info weer leesbaar te krijgen.

#### create\_car\_dict(self, car)

In de variable car dienen al gegevens te zitten die gegeven zijn in de ‘car’ tabel. De functie zal verder de benodigde gegevens opzoeken in de tabel ‘car\_parked’ die passend zijn bij het kenteken. Vervolgens wordt deze data van ‘car’ en ‘car\_parked’ samengevoegd in een dictonary en terug geresulteerd.

## 

## ERD uitwerking

In de onderstaande **ERD** (Entity-relationshipmodel) zal een overzicht geven van de database structuur die gebruikt gaat worden voor het project.

C:\Users\robbie\Dropbox\b1 programming\code\database\erd optie 2.png

## PSD Uitwerkingen

In de onderstaande **PSD** (programma structuur diagram) zal globaal het proces beschreven worden van onze software als een auto wilt parkeren in de parkeergarage.

