AALBORG UNIVERSITY
COPENHAGEN

Semester: ICTE4

Title: Electric car fleet prediction by means of machine learning

Project Period: 01.09.2018 – 06.06.2019

Semester Theme:
Master thesis

Supervisor(s):
Niels Koefoed
Knud Erik Skouby

Project group no.: 34SER L1

Members
(do not write CPR.nr.):

Laurentiu Narcis Zait
(20147152)

Pages: 54
Finished: 05.05.2019

Aalborg University Copenhagen
A.C. Meyers Vænge 15
2450 København SV

Secretary: Maiken Keller

Abstract:

Electric car rental companies print out every morning a list of cars that need to be put to recharge.
As a starting point, the prototype will present the user with the shortest route between the cars. This opens up the discussion about graph theory, or to rely on 3rd party services such as Google Maps.
On top of this route, the geographical location of each electric charger will be added to the route generating algorithm. This might change the order of the cars the person has to visit.
By adding additional data and using machine learning on statistical data, the prototype will try to predict where there is a need for a rental car (which geographical area would make more sense to place the cars). Indirectly this could help the company increase its revenue.
The project's prototype will be not a consumer facing product, but a product to be used inside a company.

ELECTRIC CAR FLEET PREDICTION BY MEANS OF MACHINE LEARNING

Laurentiu Narcis Zait
Student nr. 20147152

Vapor server live link: https://driveg.vapor.cloud
Vapor server repo: https://github.com/narciszait/driveGreen
Phase 1 repo: https://github.com/narciszait/driveGreen_iOS_phase1
Phase 2 repo: https://github.com/narciszait/driveGreen_iOS_phase2
Phase 3 repo: https://github.com/narciszait/driveGreen_iOS_phase3

1. Introduction

This master thesis project has the intention of creating a prediction system for electric car rental companies. The predictions will be focused around where the rental company should place their cars to charge for maximum rental opportunity.

The whole project starts from the premise, that the cars transmit to a central location their location, battery status and maybe other sensor readings (e.g. windshield cleaning liquid). Taken this into consideration, the cars can be considered to belong to an Internet of Things scenario. The way the cars report these readings is out of scope in this project. The cars could have their own mobile data connection and communicate with the "home server" or the communication could be done through the user application, in the background without the user knowing this.

The project starts from the premise that electric car rental companies, such as DriveNow/Green Mobility, print out every morning a list of cars that need to be put to charging. The company has an employee (called jockey) assigned to a specific zone of the city, who will go to each car on his list and drive them to the nearest electric charger. It is up to him to find the optimal route for him, in order not to waste much time. Besides putting them to charge, the question that arises is whether the charging location has enough user demand or does the jockey have to find a new charging location with more user demand.

The aim of the project is to create a prototype of a service/platform offered to the above-mentioned companies. It will not be a consumer facing product, but a product to be used inside the company, enterprise application.
It will consist of a server, a machine learning service and a mobile application. Each night or at a certain amount of time, the server would ask the DriveNow (as an example) server for the list of addresses of the cars to be put to charge. The server retrieves that list and it will send it to the machine learning service to have it sorted. The Machine Learning service will send it back to the server, which will forward it to the mobile application, to be used by the employee.

The employee has nothing else to do but follow the ordered list of cars and chargers presented to him in the application.

To make things manageable and provide a structured approach to the problem, the prototype has been split into 3 phases. This makes it much easier to tackle the problem at hand, and each subsequent phase after phase I, builds on top of the previous one.

Phase I will contain an iOS application, where the employee has to log in to get the list of cars that he needs to put to charge. After logging in, he will see the cars marked on a map and he will be shown the shortest path for going through each car.

Phase II builds on top of Phase I and will add the location of the electric chargers to the map. The route will contain the car locations and the charger locations. This will most likely change the order the employee goes through each car.

Phase III will see the introduction of an extra property (an additional parameter) to the electric chargers. This property will represent the number of users that have rented a car charging at that charger. The property can be thought of representing how popular a charger is, better yet, the popularity can be extended to a 2-3-4 hundred meter radius zone around the charger.

With this new popularity property, the aim is to create a prediction system, either powered by manual calculations or by machine learning, to help by indicating to the jockey on where to put the car to charger.

The underpinnings of the prediction system will be discussed throughout the report and what it means to be powered by manual calculations or by machine learning.

2. Problem formulation

In order to better define the project's goals, a problem formulation is to be made.

"How could an electric charger prediction system based on machine learning be a solution for maximizing rental opportunities for an electric rental car fleet?"

Several sub questions have been devised to help answer the problem formulation:

- Why choose machine learning?
- What can be used instead of machine learning?
- What does "solution" refer to?
- What does "prediction system" mean?

The sub questions are meant to divide the main problem formulation into smaller chunks and will be answered in subsequent chapters. By answering them, it will help create a better understanding of the project and what it wants to achieve.

Delimitations

To better answer the problem formulation, project delimitations should be presented.

Since the prototype is intended to be used inside a company, the State of the Art chapter will not contain any competitor application(s) analysis, that try to resolve the same problem. This is due to the enterprise nature of the prototype, and because it is not a consumer facing product. Most likely, similar applications are developed in-house, or under heavy NDA terms if made by outside parties.

As it will be explained in the Generating data chapter, acquiring a valid dataset has turned out to be a challenge that affected the development of the prototype. The outcome of this challenge is discussed in the Development chapter.

3. Methodology

The project started out with defining the idea and coming up with the problem formulation.

The next step taken was to start research. The research went into different directions: if there are similar solutions in existence; research about machine learning service and machine learning algorithms that could prove to be useful; research about datasets that could be used.

After the research was over, time was spent in two directions.
One direction was taken into writing drafts for the different chapters of the reports and then finalizing them.
The other direction was taken into the development of the prototype. Time was spent sequentially in generating the dataset, developing the iOS apps and the Vapor server.

For the project, there has been no user testing, focus groups conducted. Also no surveys were conducted, due to the enterprise nature of the projects.

In the initial stage of the project, contact with an electric car rental company has been established. An offer for collaboration has been given, but the company did not answer it. Upon subsequent tries to reestablish dialogue, the company remained silent. If the collaboration succeeded, then project would have had a different outcome for the prototype and a different conclusion of the report.

The project adopted an agile approach throughout its evolution. Weekly or every two weeks meetings have been arranged with the project supervisor, where chapter drafts and prototype progress were presented. Feedback was given and taken notice of and the next chapter draft was decided upon for the next meeting.

4. State of the Art

As stated in the Delimitations subchapter, the project is an enterprise application and it has proven hard to find equivalent projects out in the public. Therefore, the aim of the State of the Art chapter is to present all the theories, algorithms and services that are going to be used in developing this chapter.

Since Phases I and II deal with finding the shortest path possible for the employee to go through each car and the pair of car-charger, graph theory, shortest path and Dijkstra's algorithm are going to be used.

Phase III has to do with predicting where to put the car to charge. This can be done with a hand calculated linear regression, but it can also be optimized/automated by using machine learning. The case for when to use manual calculation or when to use Machine Learning will be in a future chapter. Machine learning is a very board subject and so, only a subpart of the algorithms used in the field of machine learning will be discussed. Also, different machine learning services will be brought into discussion, as they offer different options that could be used.

Graph Theory

In Mathematics, graphs are discrete structures that contain vertices and edges. Vertices can be considered points and vertices are the lines that connect the points. Graph have many utilities in everyday life, without people realizing it. Some examples of graph usage are: model acquaintanceships between people, model telephone calls and telephone numbers, model roadmaps. Graph can also be used to show if it is possible to walk down every street in a city, without passing on the same street twice.
The mathematical definition of a graph is: "A graph G = (V,E) consists of V, a nonempty set of vertices (or nodes) and E, a set of edges. Each edge has either one or two vertices associated with

it, called endpoints. An edge is said to connect its endpoitns." (Discrete Mathematics and Its Applications, Kenneth H. Rosen, 2013)

The following picture shows a representation of a simple graph. A simple graph has its edges connect to two different vertices. At a closer look, one can see that no two different edges connect to the same pair of vertices.

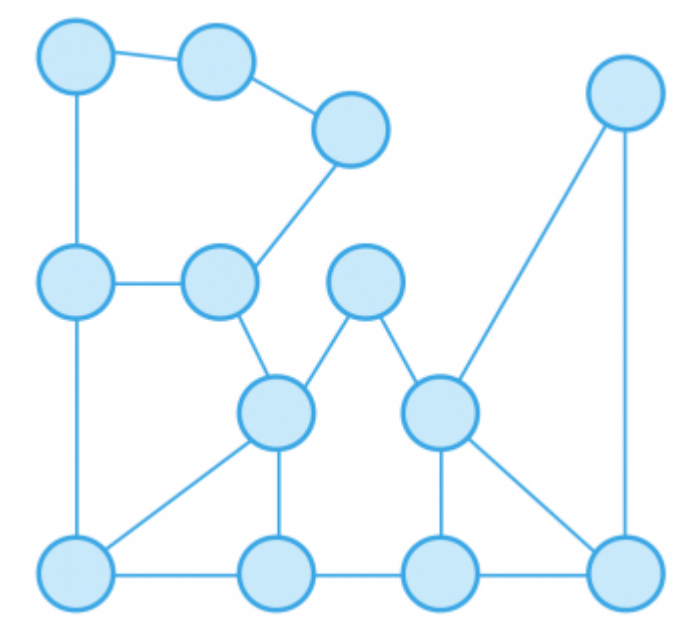


Figure 1 a simple graph[1]

Besides the graphical representation of a graph, there are 2 other possibilities to represent a graph: an adjacency list and an adjacency matrix.

In an adjacency list, each edge is listed and next to it, each vertice that connect to it.

[1] https://www.raywenderlich.com/773-swift-algorithm-club-graphs-with-adjacency-list

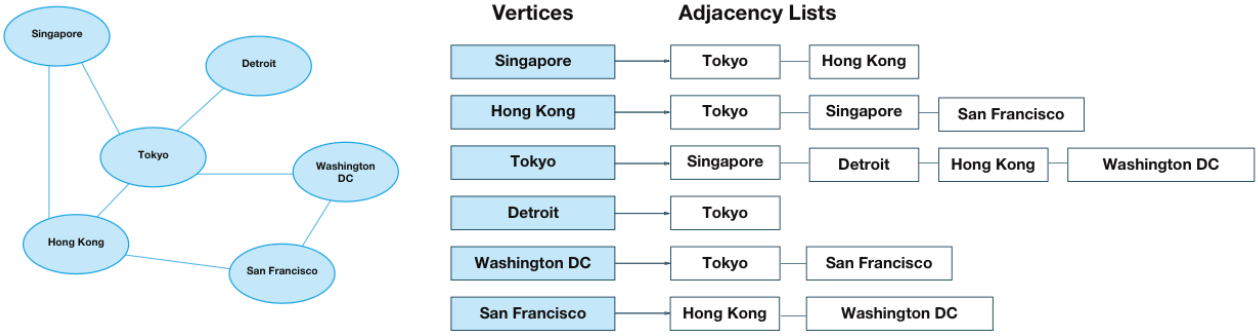
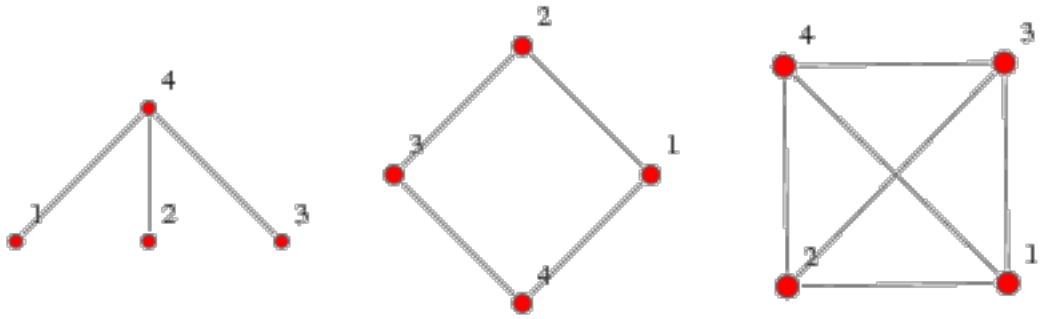


Figure 2 Adjacency list[2]

The adjacency matrix is a simple matrix, labeled with rows and columns corresponding to the graph's vertices, with a 1 or a 0 if the vertices are adjacent (they are neighbors in an edge).



$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Figure 3 Adjancecy Matrix[3]

In Computer Science, graphs are represented as a data structure that consists of a finite set of vertices and a set of edges, which are references/links between the vertices. The edges are

[2] https://www.raywenderlich.com/773-swift-algorithm-club-graphs-with-adjacency-list
[3] http://mathworld.wolfram.com/AdjacencyMatrix.html

represented as ordered or unordered pairs, depending on whether the graph is directed or undirected.

The following graph will be used as an example for how graph are represented in Computer science:

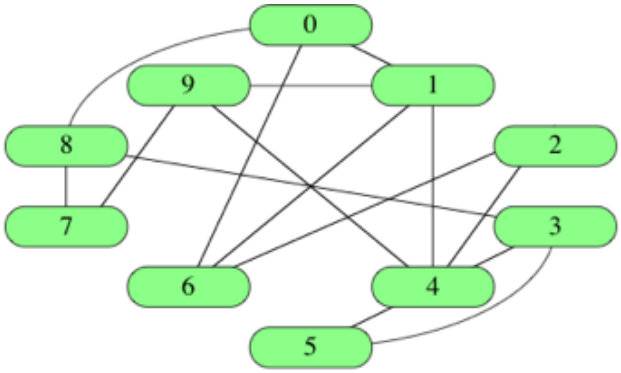


Figure 4. A graph example[4]

One way to represent the graph, as a data structure, according to the definition is by using an edge list. For the above graph, the edge list would look like this:

```
[ [0,1], [0,6], [0,8], [1,4], [1,6], [1,9], [2,4], [2,6], [3,4], [3,5],
[3,8], [4,5], [4,9], [7,8], [7,9] ]
```

The edge list is made up of one big array that contains smaller arrays which denote the connection between the edges: e.g. [0,1], [0,6], [0,8] - as it can be seen from the image, the vertex 0 is connect to vertices 1, 6 and 8.

Another way to represent the above graph is by using an adjacency matrix. This looks like the following:

```
[ [0, 1, 0, 0, 0, 0, 1, 0, 1, 0],
  [1, 0, 0, 0, 1, 0, 1, 0, 0, 1],
  [0, 0, 0, 0, 1, 0, 1, 0, 0, 0],
  [0, 0, 0, 0, 1, 1, 0, 0, 1, 0],
  [0, 1, 1, 1, 0, 1, 0, 0, 0, 1],
```

[4] https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs

```
  [0, 0, 0, 1, 1, 0, 0, 0, 0, 0],
  [1, 1, 1, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 1, 1],
  [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
  [0, 1, 0, 0, 1, 0, 0, 1, 0, 0] ]
```

The adjacency matrix is made up of a 2D matrix, containing one big array that contains smaller arrays. The smaller arrays are for each of the vertices and inside, they contain 0 for no edge between the vertex and 1 if there is an edge.

Another way to represent that graph is to use an adjacency list:

```
[ [1, 6, 8],
  [0, 4, 6, 9],
  [4, 6],
  [4, 5, 8],
  [1, 2, 3, 5, 9],
  [3, 4],
  [0, 1, 2],
  [8, 9],
  [0, 3, 7],
  [1, 4, 7] ]
```

The adjacency list is also represented as a 2D matrix, containing one big array made up by smaller arrays. The smaller arrays contain the adjacent/neighboring vertices.
The following is an example of how a graph can be represented, and initialized in the Swift programming language.

```
public class Vertex {
    var neighbors: Array<Edge>
        init() {
        self.neighbors = Array<Edge>()
    }
}
```

```
public class Edge {
    var neighbor: Vertex
    var weight: Int
    init() {
        weight = 0
        self.neighbor = Vertex()
    }
}
```

```
public class SwiftGraph {
    //declare the graph
    private var graph: Array<Vertex>
    public var isDirected: Bool

    init() {
        graph = Array<Vertex>()
        isDirected = true
    }

    //create a new vertex
    func addVertex() -> Vertex {
        //set the key
        let childVertex: Vertex = Vertex()

        //add the vertex to the graph canvas
        canvas.append(childVertex)

        return childVertex
    }
}
```

```
//add edge to source vertex
func addEdge(source: Vertex, neighbor: Vertex, weight: Int) {
        //new edge
        let newEdge = Edge()

        //establish default properties
```

```
        newEdge.neighbor = neighbor
        newEdge.weight = weight
        source.neighbors.append(newEdge)

                //check condition for an undirected graph
        if isDirected == false {
                    //create a new reversed edge
           let reverseEdge = Edge()

           //establish the reversed properties
           reverseEdge.neighbor = source
           reverseEdge.weight = weight
           neighbor.neighbors.append(reverseEdge)
       }
    }
```

Shortest Path

Graphs can be used to generate the shortest path between multiple nodes. One example could be an airline route system, or a computer network. In order to use the shortest path, a new concept has to be introduced: weighted graphs.

Weighted graphs have a number, be it positive or negative, assigned to each of their edges.

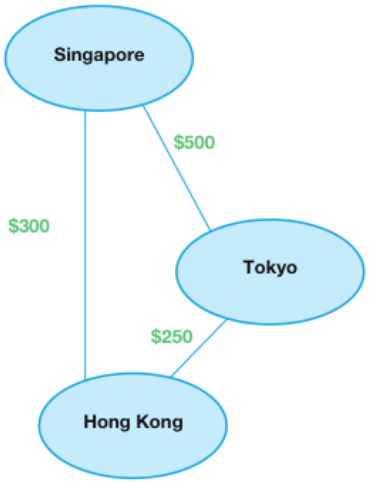


Figure 5 A weighted graph[5]

[5] https://www.raywenderlich.com/773-swift-algorithm-club-graphs-with-adjacency-list

To find the shortest path, one would have to find all the vertices between the two edges that he would want to connect by the path, and from that selection, select the vertices that have the smallest weight. Things can get complicated if there is a request to make a circuit (a smaller subset of vertices from a graph, that have a common start and finish edge), where each vertex is visited only once. This is known as the “travelling salesperson problem”.

Dijkstra’s Algorithm

Dutch mathematician Edsger Dijsktra came up with an algorithm on how to calculate the shortest path in an undirected weighted graph, in 1959. The only condition that the graph needs to fulfill is to have the weights as positive numbers.

Following is a pseudocode representation of how Djikstra’s algorithm works:

```
1.  procedure Djikstra(G: weigthed connected simple graph, with all weights
positive)
2.  /* G has vertices a = v1, v2, …., vn = z and lengths w(vi,vj) where
w(vi,vj)= ∞ if (vi,vj) is not an edge on in G */
3.  for i := 1 to n
4.     L(vi) := ∞
5.  L(a) := 0
6.  S := ∅
7.  /* the labels are now initialized so that the label of a is 0 and all other
labels are ∞ and S is the empty set */
8.  while z ∉ S
9.    u := a vertex not in S with L(u) minimal
10.  S := S ∪ |u|
11.  for all vertices v not in S
12.     if L(u) + w(u,v) < L(v) then L(v) := L(u) + w(u,v)
13.     //this adds a vertex to S with minimal label and updated the labels of
vertices not in S
14.return L(z) // L(z) = length of a shortest path from a to z
```

Djikstra's algorithm finds the length of a shortest path between two vertices in a connected simple undirected weighted graph. As it can be seen from the above pseudocode representation, Djisktra's algorithm uses O(n2) operations (additions and comparisons) to find the shortest path.

Machine Learning Algorithms

Linear regression

The Linear regression algorithm assumes a linear relationship between the input variables (x) and the single output variable (y). Put in simple terms, (y) can be calculated from a linear combination of the input (x). If (x) is a single variable, then we are dealing with a simple linear regression. The (y) equation and a simple graph representing the equations are shown in the figure below:

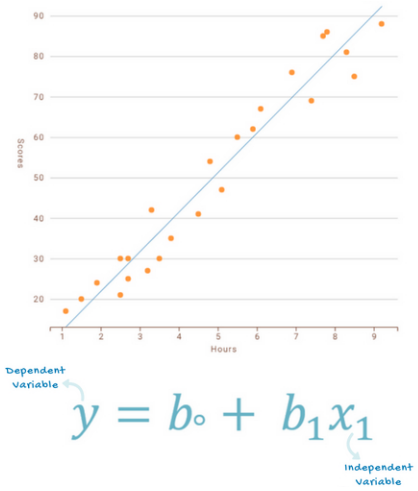


Figure 6 Simple Linear regression[6]

When having multiple input variables, the method is called multiple linear regression. The (y) equation looks like this:

```
f(x,y) = w1 * x + w2 * y
```

x, y are input variables; w1, w2 are the coefficient/weighs used.

They can have random values or predetermined ones at the beginning and then they can be fine-tuned to better place the function values on the desired axis.

Linear regression is actually an algorithm borrowed from statistics. Its main use is in predictions.

[6] https://github.com/Avik-Jain/100-Days-Of-ML-Code

Logistic regression

Logistic regression is used for a different class of problems known as classification problems. The aim is to predict the group to which the current object under observation belongs to. A very simple example would be to guess the height of a person based on the person's sex.

The algorithm measures the relationship between the dependent variable (what it being tried to predict) and one or more independent variables (the features), by estimating probabilities. The probabilities must have values of 0-no or 1-yes. To achieve the binary values is the task of logistic function, called also sigmoid.

The sigmoid function is an S-shaped curve that can take any real-valued number and map it into a range of values between 0 and 1, but never exactly at those limits. An example of a sigmoid function's graph can be seen below

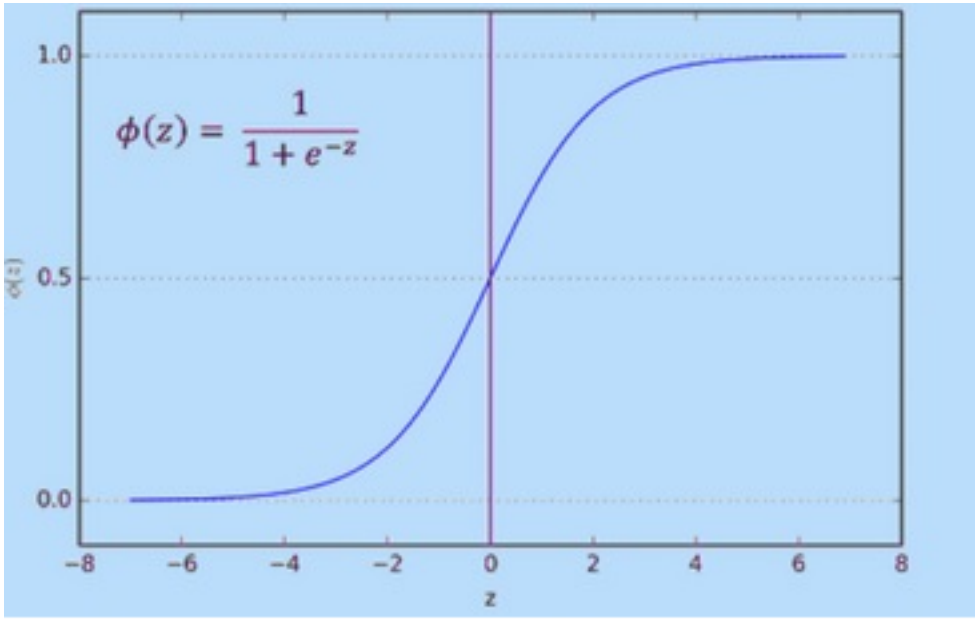


Figure 7 A sigmoid function[7]

Taking a look at the above figure, it should be noted that the graph of the function will never reach +/- 1. The above example can be a little misleading as it shows that when the z value is 8, the y

[7] https://github.com/Avik-Jain/100-Days-Of-ML-Code

value is 1. It is not 1, but a very close number to 1 and continues as such to infinity, while never reaching a value of 1. The same is true for the revers (-8 as a value for z and -1 for y)

Neural networks

"Neural networks are a class of machine learning algorithms used to model complex patterns in datasets using multiple hidden layers and non-linear activation functions. A neural network takes an input, passes it through multiple layers of hidden neurons (mini-functions with unique coefficients that must be learned), and outputs a prediction representing the combined input of all the neurons."[8]

The following image showcases how a neuron can be imagined:

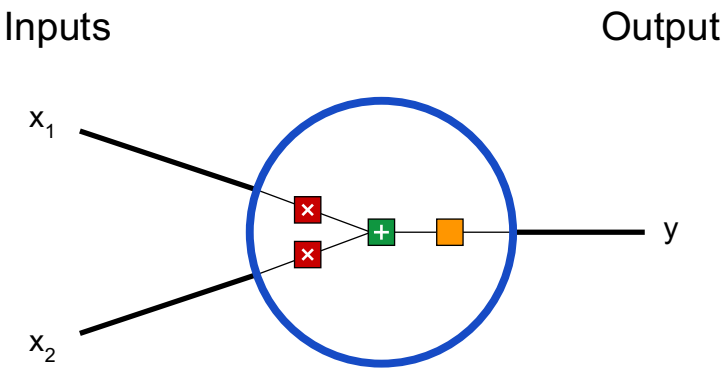


Figure 8. Representation of a neuron[9]

3 things are happening here. First, each input is multiplied by a weight ■:

$x_1 \rightarrow x_1 * w_1$

$x_2 \rightarrow x_2 * w_2$

Next, all the weighted inputs are added together with a bias b ■:

$(x_1 * w_1) + (x_2 * w_2) + b$

Finally, the sum is passed through an activation function ■:

$y = f(x_1 * w_1 + x_2 * w_2 + b)$

[8] https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html
[9] https://victorzhou.com/blog/intro-to-neural-networks/

The activation function is used to turn an unbounded input into an output that has a nice, predictable form. A commonly used activation function is the sigmoid function, the one describes above.

A neural network is a collection of neurons connected together. The following image will provide an overview of how a neural network likes like:

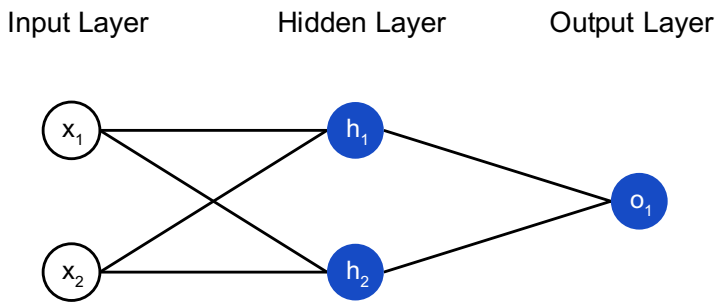


Figure 9. Representation of a neural network[10]

As it can be seen above, the network has 2 inputs, one output and 2 hidden layers. A hidden layer is any layer between the input and the output. There is no rule specifying how many hidden layers can be in a neural network.

It has the name of a network because the inputs for the output o_1 are the outputs from h_1 and h_2.

Neural networks are used in prediction systems, time series predictions, anomaly detection and natural language processing.

Machine learning software

Machine learning has its roots in statistics. A couple of software products that are today being used for machine learning, were initially used for statistical data, e.g. numPy, SciPy, pandas for Python.

In recent years the interest in machine learning has exploded and companies started offering LaaS (Learning-as-a-Service). LaaS makes all the calculations needed with the power of the cloud, freeing companies by using local machines to do the calculations. All the user/company needs is to have a valid set of data and an idea of what they want to achieve.

[10] https://victorzhou.com/blog/intro-to-neural-networks/

The following is a list of companies that offer LaaS:

- Microsoft Azure Machine Learning Studio[11]
- AWS Machine Learning[12]
- IBM Watson Machine Learning[13]
- Google Cloud Machine Learning Machine[14] – although Google offers Tensorflow[15] as well, which can be run on a local machine
- BigML[16]
- Etc.

All of the above-mentioned LaaSs support the above-mentioned machine learning algorithms. One way to differentiate between them, would to be to look at the price the service requires.

5. Tools used

To build the prototype, the following software tools and products were used:

- Server: the prototype server is done using Vapor[17]. Vapor is a web server architecture written in Swift, the same programming language used to program iOS, macOS, watchOS, tvOS applications. Vapor is open-sourced and has built a vibrant community of users and contributors around itself. Vapor is not the only server-side Swift framework and alternatives to server-side Swift altogether exist: Ruby on Rails, node.js, PHP, ASP.Net
- Server database: the Vapor server uses a SQLite database[18], that is written to a file. There was the possibility to have a SQLite database residing in memory, but this meant that as long as the server is running, the database would exist. When the server shuts down, the database would cease to exist. For the needs of the prototype, the SQLite database does the job very well. Vapor can also interact with MySQL, PostgresQL, MongoDB and support for other databases is constantly added.

[11] https://azure.microsoft.com/en-us/services/machine-learning-studio/
[12] https://aws.amazon.com/sagemaker/
[13] https://www.ibm.com/cloud/machine-learning
[14] https://cloud.google.com/ml-engine/
[15] https://www.tensorflow.org
[16] https://bigml.com/
[17] https://vapor.codes
[18] https://www.sqlite.org

- Server dependency manager: Swift Package Manager[19]. “The Swift Package Manager is a tool for managing distribution of source code, aimed at making it easy to share code and reuse others’ code. The tool directly addresses the challenges of compiling and linking Swift packages, managing dependencies, versioning, and supporting flexible distribution and collaboration models.”[20]

- iOS mobile application programming language: Swift . “Swift is a general-purpose programming language built using a modern approach to safety, performance, and software design patterns”[21] It was announced in 2014 by Apple, open-sourced at the end of 2015 and it was meant to replace Objective-C.
- IDE (Integrated Development Environment): Xcode[22]. Xcode is developed by Apple and it is the official IDE that includes the iOS, tvOS, watchOS, macOS SDKs so developers can build applications for the desired platforms. Xcode can also be used for Vapor projects, since they are also written in Swift.
- iOS dependency manager: Cocoapods[23]. Very similar in purpose and use to Swift Package Manager. Cocoapods existed before Swift Package Manager and it has build a strong and sharing community around it.
- Google Maps for iOS[24]. MapKit, Apple’s own Map framework could have been used, but it does not support multi stop routes.
- IQKeyboardManager[25] is a handy iOS library that takes care of the iOS keyboard not to cover text fields or text views, so the user could see what it is written in there.
- Alamorefire[26] is a HTTP networking library, written in Swift. It makes it easier for developers to make API calls in their applications.

[19] https://github.com/apple/swift-package-manager
[20] https://github.com/apple/swift-package-manager
[21] https://swift.org/about/
[22] https://developer.apple.com/xcode/
[23] https://cocoapods.org
[24] https://cocoapods.org/pods/GoogleMaps
[25] https://github.com/hackiftekhar/IQKeyboardManager
[26] https://github.com/Alamofire/Alamofire

6. Generating data

The idea of the project was to be done in collaboration with an electric car rental company. The company has been approached, the problem and the solution were presented but no answer has been given from the company regarding the collaboration.

Because of this the real dataset has become out of the picture. This left 2 choices:

1. To find a public dataset that could be used for this project
2. Generate an own dataset

Going with option number one, a series of concerns started to appear: Before using a public dataset, one would have to investigate and make sure what license agreement covers the dataset and if it can be use in university or company projects.

There exists the possibility that a specific dataset, for a specific geographical location might not exist, but it can have another geographical location. The question that arises is how relevant the dataset can be for the project.

One dataset that comes very close to the need for this project is available here: https://www.kaggle.com/doit-intl/autotel-shared-car-locations. It contains geographical locations of shared cars in Tel Aviv, Israel. The dataset has for each record the following properties: timestamp, latitude, longitude, total_cars and carsList. The last two properties and the timestamp would not be needed. Another obstacle in using this dataset, is the geographical location contained inside of it: Tel Aviv, Israel, and not Copenhagen, Denmark.

Option number two: It needs a thorough planning of what the dataset should contain and how many entries should the dataset contain initially. More data can be added later on, without a problem. The problem is represented by the cold start and trying to figure out how the ideal dataset for the project would look like.
Option number was the chosen path for the path – generating an own dataset, fit for the project. An electric car would be represented by a geographical location (latitude and longitude), battery

level (indicated in percentage) and interest in the car (indicated in percentage). It started out with generating 50 random geographical locations, on a radius of 20 km around Copenhagen. For this, a website called "Random Point Generator"[27] has been used. At the time of writing, the geographical coordinates random generator does not work anymore. Each of those 50 locations had to be manually checked on Google Maps, to be sure that none of them are in locations inaccessible (such as Amagerfælled or in Øresund). If that was the case, a new location would be chosen to replace the inaccessible one. Next up, the website called random.org[28] was used to generate 50 integers, with values between 1 and 100. This batch of 50 integers would be attached to each pair of the geographical locations previously generated. The integers would denote how much battery a car, situated at the geographical location would have remaining. A second batch of 50 random integers was generated and attached to each car. The second integer would denote the interest in the car, an indication of how popular/in demand that car is.
The geographical coordinates, the battery indicator and the demand for the car will be put together into a .csv file for easier arrangement, which will be transformed into a .json file, that will be updated to server. The explanation why this file will be residing on the server will be revealed in a later chapter.

After having a small data collection of 50 cars around Copenhagen, the next point to tackle would be finding the location of chargers for electric cars. The chargers exist, the municipality of Copenhagen has been installing them for several years now and kept on expanding this network. One solution could have been to contact the municipality and see if it is possible to get access to a list of the locations for electric chargers. Another option would be to investigate if there are any online websites that have a such a list.

One such website is Chargemap[29]. Upon opening the website, it defaults to map view over the United Kingdom of Great Britain and Northern Ireland. It does show electric charger that are in Denmark. Chargermap does not offer an open API that could be queried for charger locations. They offer a subscription service and a mobile companion app that can be used.

[27] http://www.geomidpoint.com/random/
[28] https://www.random.org/integers/
[29] https://chargemap.com/map

After several searches, the Open Charge Map[30] website has been found. It offers an open API, that developers can use to query for charger locations, but it also allows to have new electric charger locations added to the database.

The Open Charger Map API[31] has different parameters that can be used when making an API request. Of major importance for this project, are the output, country code, maximum results, latitude, longitude, distance and distance unit parameters. Putting the parameters together and making the call to the API[32] will return a number of 87 chargers. The parameters for the API call are:

- output=json the returned response will in a JSON format;
- countrycode=DK the country code is DK, for Denmark;
- maxresultdefaults=200 the maximum number of returned results will be 200. If left by default it will be 100.
- latitude=55.6761 & longitude=12.5683 represent the geographical coordinate for the center of Copenhagen
- distance=20 & distanceunit=KM used together, they denote a radius of 20 km around the center of Copenhagen

The returned data has much more information than needed:

```
1.      [
2.          {
3.              "ID": 114935,
4.              "UUID": "2769C600-1486-446D-872A-B610DE6D392F",
5.              "ParentChargePointID": null,
6.              "DataProviderID": 1,
7.              "DataProvider": {
8.                  "WebsiteURL": "http://openchargemap.org",
```

[30] https://openchargemap.org/site
[31] https://openchargemap.org/site/develop/api
[32] https://api.openchargemap.io/v3/poi/?output=json&countrycode=DK&maxresults=200&latitude=55.6761&longitude=12.5683&distance=20&distanceunit=KM

```
                    "Comments": null,
                    "DataProviderStatusType": {
                        "IsProviderEnabled": true,
                        "ID": 1,
                        "Title": "Manual Data Entry"
                    },
                    "IsRestrictedEdit": false,
                    "IsOpenDataLicensed": true,
                    "IsApprovedImport": true,
                    "License": "Licensed under Creative Commons
Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)",
                    "DateLastImported": null,
                    "ID": 1,
                    "Title": "Open Charge Map Contributors"
                },
                "DataProvidersReference": null,
                "OperatorID": 1,
                "OperatorInfo": {
                    "WebsiteURL": null,
                    "Comments": null,
                    "PhonePrimaryContact": null,
                    "PhoneSecondaryContact": null,
                    "IsPrivateIndividual": null,
                    "AddressInfo": null,
                    "BookingURL": null,
                    "ContactEmail": null,
                    "FaultReportEmail": null,
                    "IsRestrictedEdit": null,
                    "ID": 1,
                    "Title": "(Unknown Operator)"
                },
                "OperatorsReference": null,
                "UsageTypeID": 5,
                "UsageType": {
                    "IsPayAtLocation": true,
                    "IsMembershipRequired": false,
                    "IsAccessKeyRequired": false,
                    "ID": 5,
                    "Title": "Public - Pay At Location"
```

```
47.             },
48.             "UsageCost": "Free",
49.             "AddressInfo": {
50.                 "ID": 115281,
51.                 "Title": " Q-Park Industriens Hus",
52.                 "AddressLine1": "H. C. Andersens Blvd. 18",
53.                 "AddressLine2": "",
54.                 "Town": "København ",
55.                 "StateOrProvince": "",
56.                 "Postcode": "1787",
57.                 "CountryID": 65,
58.                 "Country": {
59.                     "ISOCode": "DK",
60.                     "ContinentCode": "EU",
61.                     "ID": 65,
62.                     "Title": "Denmark"
63.                 },
64.                 "Latitude": 55.675114,
65.                 "Longitude": 12.568064,
66.                 "ContactTelephone1": "",
67.                 "ContactTelephone2": "",
68.                 "ContactEmail": "",
69.                 "AccessComments": "First to the right in parking
house ",
70.                 "RelatedURL": "",
71.                 "Distance": 0.11063224614500103,
72.                 "DistanceUnit": 1
73.             },
74.             "NumberOfPoints": 2,
75.             "GeneralComments": "",
76.             "DatePlanned": null,
77.             "DateLastConfirmed": null,
78.             "StatusTypeID": 50,
79.             "StatusType": {
80.                 "IsOperational": true,
81.                 "IsUserSelectable": true,
82.                 "ID": 50,
83.                 "Title": "Operational"
84.             },
```

```
            "DateLastStatusUpdate": "2019-01-20T11:15:00Z",
            "DataQualityLevel": 1,
            "DateCreated": "2019-01-19T16:30:00Z",
            "SubmissionStatusTypeID": 200,
            "SubmissionStatus": {
                "IsLive": true,
                "ID": 200,
                "Title": "Submission Published"
            },
            "UserComments": null,
            "PercentageSimilarity": null,
            "Connections": [
                {
                    "ID": 161722,
                    "ConnectionTypeID": 17,
                    "ConnectionType": {
                        "FormalName": null,
                        "IsDiscontinued": false,
                        "IsObsolete": false,
                        "ID": 17,
                        "Title": "CEE 5 Pin"
                    },
                    "Reference": null,
                    "StatusTypeID": 50,
                    "StatusType": {
                        "IsOperational": true,
                        "IsUserSelectable": true,
                        "ID": 50,
                        "Title": "Operational"
                    },
                    "LevelID": 2,
                    "Level": {
                        "Comments": "Over 2 kW, usually non-domestic
socket type",
                        "IsFastChargeCapable": false,
                        "ID": 2,
                        "Title": "Level 2 : Medium (Over 2kW)"
                    },
                    "Amps": null,
```

```
123.                     "Voltage": 230,
124.                     "PowerKW": 3.7,
125.                     "CurrentTypeID": 20,
126.                     "CurrentType": {
127.                         "Description": "Alternating Current - Three
Phase",
128.                         "ID": 20,
129.                         "Title": "AC (Three-Phase)"
130.                     },
131.                     "Quantity": 2,
132.                     "Comments": null
133.                 }
134.             ],
135.             "MediaItems": null,
136.             "MetadataValues": null,
137.             "IsRecentlyVerified": false,
138.             "DateLastVerified": null
139.         }
140.     ]
```


Figure 10. Example of data returned for one electric charger

Most of the information returned for one charger is not needed and it can be discarded, except for the “AddressInfo” dictionary, which contains “Latitude” and “Longitude” keys.

The next step after having the list of chargers returned, would be to extract the desired information. To achieve this, the returned json was transferred into a .csv file (comma separated values). This made it much easier to manipulate the data, as the csv file would be displayed as table with 87 entries, and 121 columns - each property of the charger object was transformed into a column. All columns would be discarded as the next step, exact for the Latidute and Longitude columns, that depicted the geographical location of the charger.

As used previously, Random.org, with its random integer generator was needed again to generate two batches of 87 random integers. One batch contains integers with values between 51 and 100 and it meant to represent the rental user interest in the area near the charger. The second batch

of integers contains values between 100 and 1000 and are meant to represent how many times a car was rented in the past from that charger's location. Putting together the chargers' geographical coordinates, the "rental interest" and the "history" into a .json file and uploading them to the server, was the next step.

As stated previously, an explanation on how the 2 .json files are used, will be provided in a future chapter.

7. User Scenarios

User scenarios are a narrative way of describing how a user can interact with a piece of software, be it either a website, a mobile or desktop application.
The following user scenarios follow John, an employee at a fictional electric car rental company called DriveGreen.
The scenarios follow each phase of how the project is divided and each scenario builds on top of the other, adding complexity and new functionality.
Scenarios 4 and 5 are going to be discussed in more detail in the Future Improvements Chapter.

Scenario 1 - Phase 1

John works as a jockey at DriveGreen, an electric car rental company, where people can take the car from anywhere in the city and drop it off anywhere in the city, when they are done with it.

John's job is to put car that have very little battery left to charge.
In the morning, when his shift starts, John opens the application on his mobile phone, logs in with his company email and a password and after he successfully logs in, he is presented with a list of cars that need to be put to charge. In the settings part of the application, he can enable biometric authentication (fingerprint or facial recognition) so can spend less time typing in his email and password.

The list of cars is ordered based on the distance from where he is and then the distance between each car.
Seeing the list, John can now start his work day.

Scenario 2 - Phase 2

An update to the application brings some changes to the application:
After successfully logging in, John can see that the list of cars is a little different.

Now the list is showing a car location and an electric charger location:
`John's location -> car -> charger -> car -> charger -> etc.`
The charger shown on the list is the closest charger to the car.

Scenario 3 - Phase 3

After using the updated version of the application for several months, John receives a new update to the application.
Since he started to use the application with the previous update, John started to know approximately where he should put the car to charge.

The new update still shows the list of cars and the subsequent chargers where he should put the cars to charge, but the charger locations are not the ones he got to know.
Sometimes the list would tell John to put a car to charge at a charger that is situated 12 minutes away, instead of the one he got to know, that is only 3 minutes away.
John is curious and starts wondering why this new order in the car/charger list is. Talking with his colleagues and superiors, he finds out that the suggested chargers have a higher interest rate from the rental users and the company is trying to come to the needs/demands of the customers.

This new order in the car/charger list does affect John work day, but there is nothing he cannot handle.

Scenario 4 - Extras

The new update has been released for some time now. John got used to follow the suggestions of the application and place the cars at the recommended chargers.

The company is pushing out a new update for the application.
The intent is to make John's and his colleagues lives much easier.
The new update brings with it navigation instructions. From the moment John logs in, the application gets his location using the mobile phones GPS, the locations of the cars he needs to put to charge and the locations of the chargers where he should put the cars to charge, from a server.

From his initial location, John will get directions on how to get to the first car, by using public transport. Upon entering the car, John will get driving directions to the recommended charger. From the charger to the next car he will be travelling using public transport.
`John's location -> public transport -> car -> driving -> charger -> public transport -> car -> driving -> charger -> etc.`

Scenario 5 - Other types of rentals

This scenario refers to the shared electric scooters companies, such as Lime, Voi, Tier.
The companies use independent contractors to pick up discharged scooters, charge them at home and then place them again on the street.

In this scenario, the prototype would show discharged scooters belonging to a specific company. The contractor would have to log in to see where the scooters are placed on the map. The app would generate the shortest path between the contractor's position and all the scooters.
Upon arriving at the scooter's location, the contractor can confirm in the application that he took possession of the scooter. When the contractor picks up the last scooter, the app will display a route from his location to his house.

Upon arriving home, he sets each scooter to charge. When all the scooters are charged, he should confirm this in the application.

After confirming, he is supposed to head out and place the scooters on the street.

One possibility would be to place them at random places in the city.

Another possibility would be to have the application suggest where to place the scooters, based on how many users have rented scooters from that area before. This idea ties in with the electric charger prediction, where the number of how many users rented a car previously, is needed.

8. Requirements

To come up with the requirements for this project, the FURPS+ approach has been selected. This was created by Robert Grady from HP[33]. FURPS comes from the initials of Functionality, Usability, Reliability, Performance and Supportability. The + comes from a series of additional requirements: Design, Implementation, Interface and Physical requirements.
The following table will give an overview on the requirements, to which part of the prototype they apply to and how they can be tested. The requirements are extracted from the User scenarios chapter.

Area	Functionality	Testing	Server/Mobile app
Functionality	Provide secure login process	Check for a secure connection to the server that does not leak	Server / Mobile app
Functionality	Support biometric login	Implement the LocalAuthentication framework from iOS	Mobile app
Functionality	Be able to retrieve car and charger location	Make API calls to the server	Mobile app
Functionality	Provide an overview of the cars	Show the cars on a map	Mobile app
Functionality	Have the possibility to update car locations	A refresh mechanism (e.g. pull-to-refresh, etc.)	Mobile app
Usability	The application must be easy to use	User testing, focus group	Server / Mobile app

[33] Peter Eeles, Capturing Architectural Requirements, 2005
https://www.ibm.com/developerworks/rational/library/4706.html

Usability	Not make it hard for the user to retrieve data	User testing, focus group, app architecture	Mobile app
Reliability	The prototype should have an uptime of 99.9%	Server monitoring	Server
Reliability	The prototype should be tolerant to user mistakes	Provide UI feedback	Server / Mobile app
Reliability	Provide appropriate feedback to user in case of mistake/error	Provide UI feedback	Server / Mobile app
	There will be no offline mode	Test in case of no connectivity	Mobile app
Performace	Must be available 24/7	Server monitoring	Server
Performace	Connection should be kept alive even in bad mobile network situations	Strangle the internet connection	Server / Mobile app
Supportability	The app would be localised in both English and Danish	Switch the phone's language between English and Danish	Mobile app
Supportability	The prototype should be built in a way that future improvements could be added in a manageable fashion	Design patterns, Software Architecture	Server / Mobile app
Implementation	The app will be written in Swift	Inspect the source code	Mobile app
Implementation	The server will be written in Swift, using Vapor	Inspect the source code	Server
Implementation	The app will be made available initially for iOS	Source code	Mobile app
Interface	The prototype will rely on APIs to ensure communication between different parts of the system	Listen for traffic between the mobile app and server, using Charles[34]	Server / Mobile app

[34] https://www.charlesproxy.com

9. Development

Initial idea

The prototype was envisioned to be composed of 4 components:

- The electric car rental company server
- A Vapor server
- A machine learning service
- An iOS app

Every night, a CRON job would run on the DriveGreen (a fictional electrical car rental company) server and it will forward the locations of the almost discharged cars to the Vapor server. The Vapor would forward the list of locations to either a Learning-as-a-Service product, or it could trigger locally a machine learning model generation.
The LaaS, or the local ML implementation would have available beforehand the list with the electric charger locations and it would use that together with the car locations to generate the machine learning model. After the model is generated, it will output a list, consisting of cars and to which charger to put the car to charge. This list would be send back to the Vapor server.
When the employee logs in in the morning, the iOS app will do an API call to the Vapor server to retrieve the above mentioned list and then he start his work day.
The following diagram gives an overview of how the system was imagined to work:

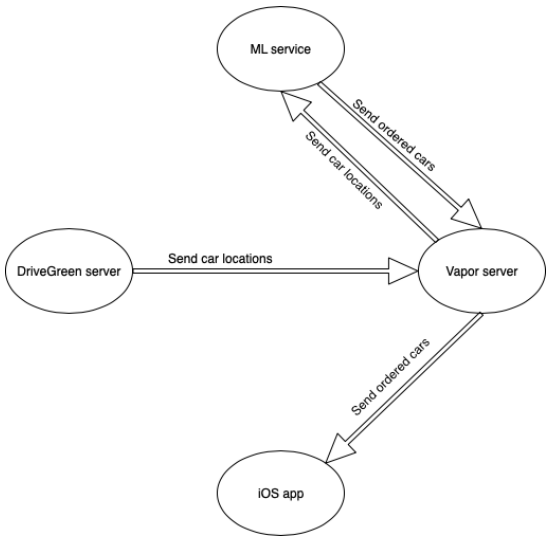


Figure 11. Prototype diagram

As discussed in the Generating data chapter, acquiring a viable dataset proved to be challenge. The option to generate a dataset from zero was chosen. This resulted in having 2 JSON files, one for a series of 50 cars and another one with locations of the electric chargers around Copenhagen. The initial plan was to store and retrieve both JSON files from the Vapor server, but in the end just the cars JSON is stored on the server, while the chargers JSON is stored in the iOS app.

Server development

Server development started with a slow tempo, as it was a new, uncharted territory. It started from the basic template that is offered when a new Vapor project is created. This gave the opportunity to get acquainted with how things work in the Vapor world.
In initial phase, the server was developed to provide CRUD (create, read, update, delete) operations on users. For this to work, a User model has to be defined, with all the properties needed for a User object.
The UserController is where the CRUD operations are defined and where the routes for the CRUD operations are defined. It also includes the "login" route:

- Login: POST request, with the email and password in the header, encoded in base 64.
- Create User: POST request, with the email and password in the body of the request
- Read all Users: GET request, with an authorization header
- Read Specific User: GET request, with an authorization header
- Update User: PUT request, with an authorization header
- Delete User: DELETE request, with an authorization header

The following table offers a better overview of the User routes. The Security chapter is discussing about the authorization header and the base 64 encoded pair of email and password.

API Call / Route	Operation
POST vaporServer/api/users/login	User login
GET vaporServer/api/users	Get a list of all the registered users
GET vaporServer/api/users/:id	Get a specific user, by providing the user id
PUT vaporServer/api/users/:id	Update a specific user, by providing the user id
POST vaporServer/api/users	Create a new user

DELETE vaporServer/api/users/:id	Delete a specific user, by providing the user id

The list of cars is stored in a JSON file, that the server would read from and then forward it to the iOS application. The Car model file defines the properties of the car object and offers a class function that reads the JSON file and it returns an array containing all the cars from the JSON file. The CarController defines 3 routes, but only one is being used from the iOS app. The routes are covered in the following table.

API Call / Route	Operation
GET vaporServer/api/cars	Returns all the cars from the JSON file, in the order they are stored in the file
GET vaporServer/api/cars/random	Randomizes the order of the cars in the array returned from the JSON file
GET vaporServer/api/cars/:int	Randomizes the order of the cars in the array returned from the JSON file and returns the :int (an integer) first elements of the array

The Vapor server does not have any GUI. All the request are to be made using tools like Postman, Paw or the old-but-trusty cURL command line tool.

Mobile application development

The iOS part of the prototype consists of 3 iOS applications. It could have been done as one iOS app with 3 targets, but to differentiate between the codebase, compiler #if statements would had to be used and that would make the code very ugly and not easy to read.
The 3 applications are named Phase 1, Phase 2 and Phase 3. They correspond to each of the phases described in the Introduction chapter.

Phase 1

The Phase 1 application consists of 3 screens, as it can be seen in the Storyboard screenshot below, although the screenshot shows 5 screens. This will be explained shortly.

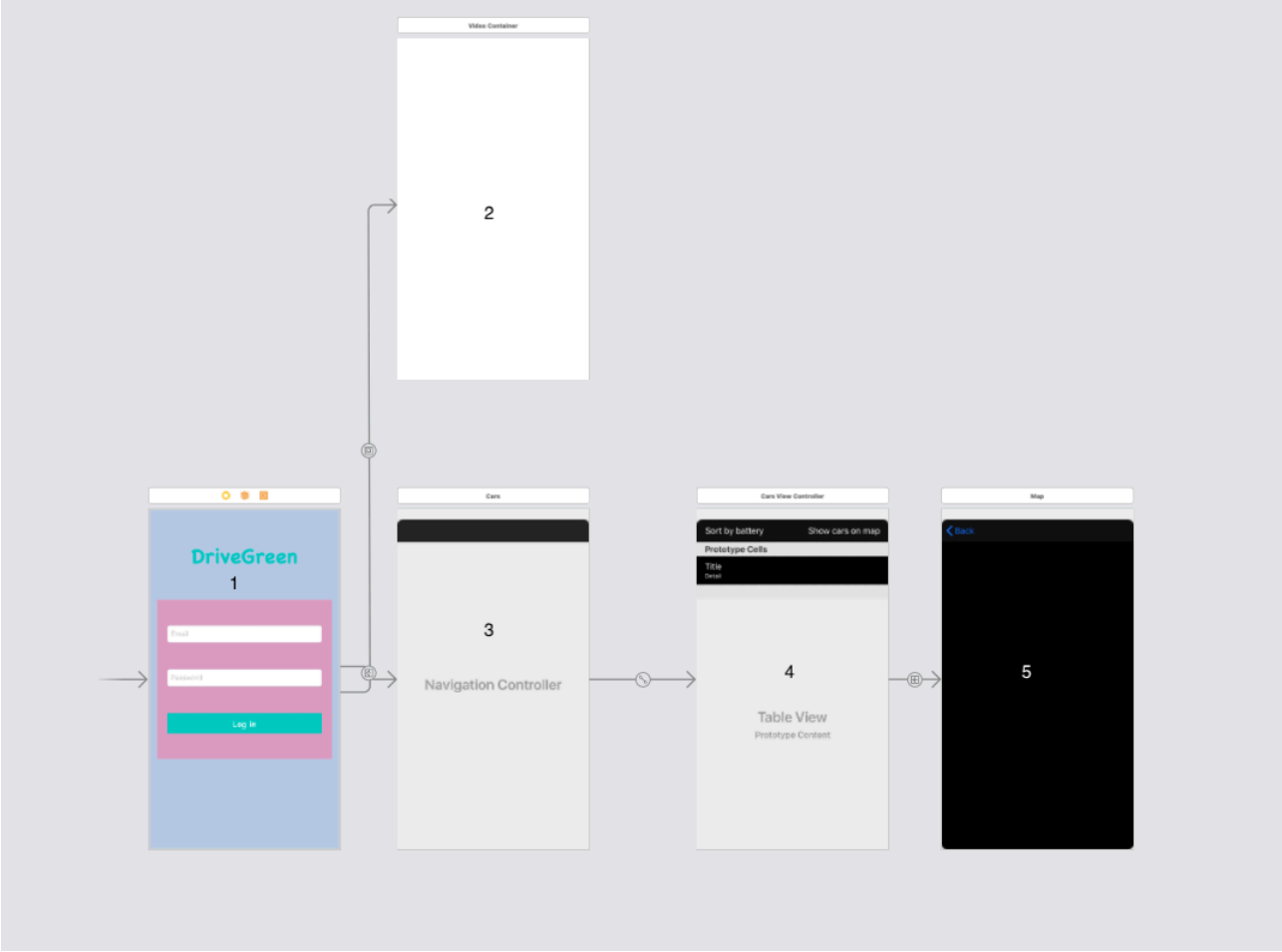


Figure 12. Phase 1 application layout

The first screen is the Login Screen, numbered 1 in the above screenshot. This consists of 2 text fields and a button. The 2 text fields are used for the user to write his work email and his password. The button is used for the login action. As it can be seen from the screenshot, there is screen that connects to the Login Screen. This is a container view, numbered 2, that plays a video on the background of the Login screen.

Upon tapping on the login button, the app makes an API call to the Vapor server to log in the user. The app takes the email and password values provided, concatenates them (divided by a double colon, e.g. “email”:”password”) and encodes the concatenated string to base 64. The encoded string is added to the Login request header. The header is further explained in the Security chapter.

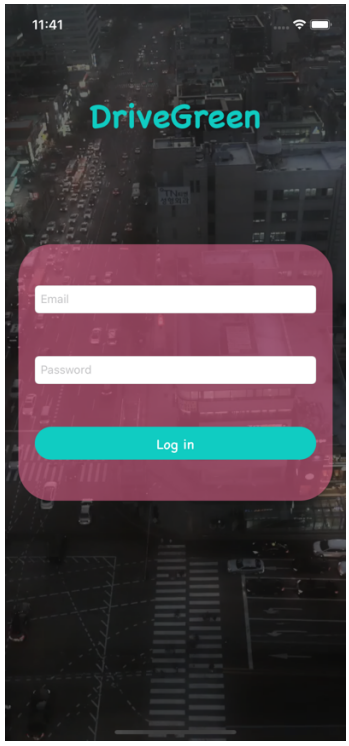


Figure 13. The Login Screen

After successfully logging in, the Vapor server returns a 200 HTTP status code and among other values, a token. This token is saved in the UserDefaults, which is "an interface to the user's defaults database, where you store key-value pairs persistently across launches of your app."[35]. The token value will persist until the user deletes the app, or resets the phone, or the application rewrites the token value.

After having the token value available, the app makes a second API call to the server to retrieve the list of cars. When the list is retrieved, screen number 4 is loaded.

[35] https://developer.apple.com/documentation/foundation/userdefaults

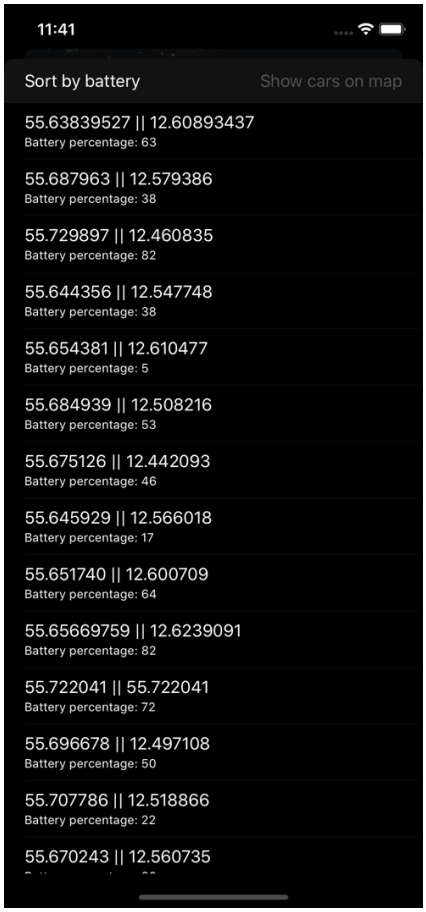


Figure 14. The Car list screen

It will display a list, containing the array of cars, sent by the server. Here, the user can select 3 cars and upon tapping on the upper-right corner button (“Show Map”), it will redirect him to screen number 5.

Screen number 5 contains a Google Maps View, where the selected cars are shown as pinpoints on the map. The map also shows the user’s location, but since using the simulator, the user’s position is simulated as being close to Vesterport Station, in Copenhagen.

Figure 15. Car locations on a map

The “Show Routes” button in the upper-right corner, upon tapping, will make an API call to the Google Maps Directions API, where it send the geographical coordinates of the cars plus those of the user. This request will ask the Maps Directions to provide a route that will pass through all the coordinates provided and to calculate the shortest path possible.

When the request is returned to the app, the map will display a route between the selected coordinates. Sometimes the route will resemble a circle, but other times it would show that the user would have to travel twice on the same road.

The shortest route suggested by Google Maps does not show an order for the user to travel.

When the user taps on one of the car markers, a small popup will appear showing how much battery the car has remaining and also the general population’s interest in the car.

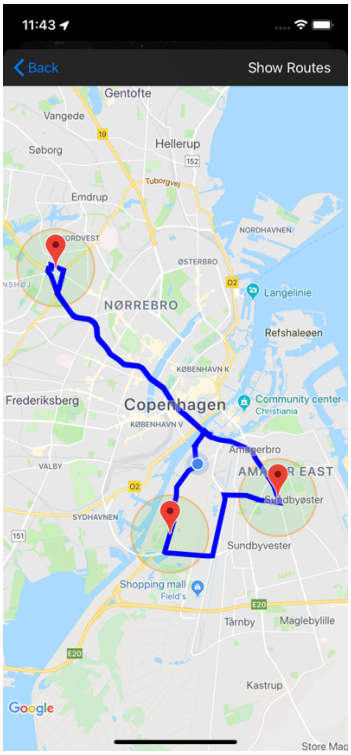


Figure 16. Shortest route between the selected cars and the user's location

Upon tapping on the “Back” button, the user is taken to the previous screen, the list of cars. Here, he can select other 3 cars and have them shown on the map.

From the application layout screenshot, one screen was not mentioned. That is screen number 3, which can be seen it is called a Navigation Controller. This is not a screen directly seen by the user, except for the navigation bar where the “Back” button is, present in screen number 5, the Map screen. The Navigation Controller makes it possible to have a hierarchy of screens and in this case it implements the List screen (number 4) and the Map screen (number 5) and makes it possible for the user to come back to the previous screen when he is looking at the Map Screen.

Phase 2

The Phase 2 application is a copy of the Phase 1 application. It offers the same functionality, except it also loads the location of the electric chargers and shows them on the map screen.
As it can be seen in the next screenshot, loading all 87 electric chargers clutters the map and the selected cars are barely visible on the map

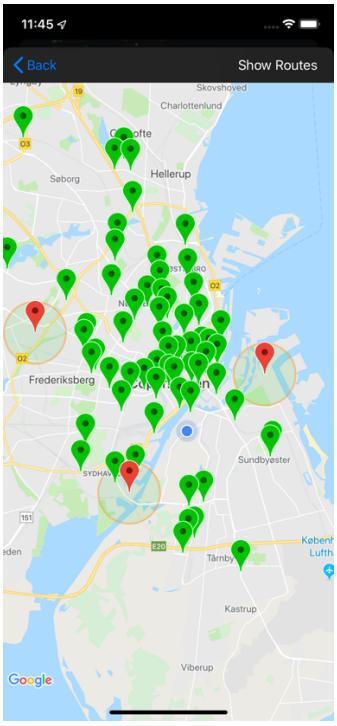


Figure 17. Cars, electric chargers and the user's locations

To clean up the map screen, 2 simple conditions were implemented:

When the chargers are loaded, the distance between each car and each charger is calculated and if the distance is between 700 m and 1.000 m, then the charger will be shown on the map.

By having these 2 conditions, the map screen decluttered and it became cleaner.

Figure 18. Electric chargers that fulfill the 2 conditions

The chargers list, as mentioned previously, is not stored on the server, but inside the iOS application. Ideally, it should be stored on the server. Also, the calculations for the distance between each charger and each car should be ideally done on the server and on device, but Vapor does not include Core Location, Apple's framework for geographical services.

Phase 3

The Phase 3 app builds on top of Phase 2.
The only addition that Phase 3 has is to add another 2 conditions to the already existing distance conditions and those are if the charger has a rental history bigger than 500 and a general population interest bigger than 50.
The numbers, 500 and 50, can be tweaked and have other values. The same is available for the distance numbers.
This is not a solution generated by machine learning, but it is a very simple, manual calculation, which have as a result one or several electric chargers that meet the desired criteria.

10. Security

By security, the intention is to refer to identity and access management. The following chapter will look at the individual pieces of the what the prototype is composed of: the iOS, the server.
Before starting, a series of explanations is needed.

Identity and access management consists of a collection of policies, process and systems which ensure that an individual or a system is given a set of permissions within another system. Identity and access management contains the actions of authentication and authorisation among other things.

Authentication is process in which someone confirms its identity, he proves to be who he claims to be. In the physical world, this can be done by providing government issued documents such as national id card, driving license or passports. In the online world, a common way of authentication is through the user - password combo, but it is not limited only to that. In recent years users could

use biometrics - either by using fingerprint or facial recognition - to log in to different application or online services. In some cases, the system might require multi-factor authentication, where a user has to provide an additional set of credentials, e.g. the user provides a username/password combo, and the next step is to provide a pin number received through an SMS message on the user's mobile phone.

Authorization, in the context of cybersecurity, refers to the process of giving the user the right to access specific functions or resources. Instead of the word authorization, it can be exchanged for client privilege. The access to specific resources or functions can be information, files, emails, databases, funds, etc.

Access control comes in between authentication and authorization. When the identity of the user is confirmed, he is granted access to the system. The problem that arises is how much access can the user have. This is solved by limiting the access through access control. "Authorization defines the set of actions that the identity can perform after gaining access to a specific part of the infrastructure, protecting from threats that access controls alone are ineffective against."[36]

The following table gives a better overview of the 3 definitions presented above.

Category	What	Protects from
Authentication	Confirms the identity is who it says it is	The whole world
Access Controls	Provides additional authentication / validation to access specific resources	Compromised (stolen) Credentials
Authorization	Determines what actions the identity can perform on specific resources	Accidents, Compromised Credentials, Malicious Insiders

[36] https://cloudknox.io/authentication-vs-access-controls-vs-authorization/

This project's prototype is intended to be used in the enterprise medium. Therefore the data available on the mobile application and the server must be kept private from the outside world. In order to do so, a subset of the actions belonging authentication and authorization are included with the prototype. Authentication is available on the iOS app and on the server, making sure that just employees of the company can have access to car locations and battery levels, charger locations and charger popularity. Authorization makes sure the employees who have been described in this project have access to only the needed and desired information and nothing more.

The data contained in the prototype as a whole (looking at the iOS app, the server and the ML service) can be considered anonymous data. It does not contain personal identifiable information where a real life person can be identified. As stated in the **Generating data**, the data refers to cars, their location, their battery level, electric chargers, and the historical/statistical data the company has about these chargers.

The iOS application

In order for the employee to use the application, he will have to log in with his username/work email and a password. Further more, the application could take advantage of the biometric log in features, the device offers. This can be TouchID – for finger print authentication or FaceID – facial authentication. But in order for the biometric log in to function, the employee should log in with his username/email and password at least ones.

When exiting and coming back to the app or going to the multitasking screen on the phone, the application will hide its content, by covering the application screenshot with a generic screen. This way, an external user could not see the content of the application or have a glimpse of the information offered in the application. This practice, of obfuscating the application content, is done by banking applications (e.g. Danske Bank, e-Boks).

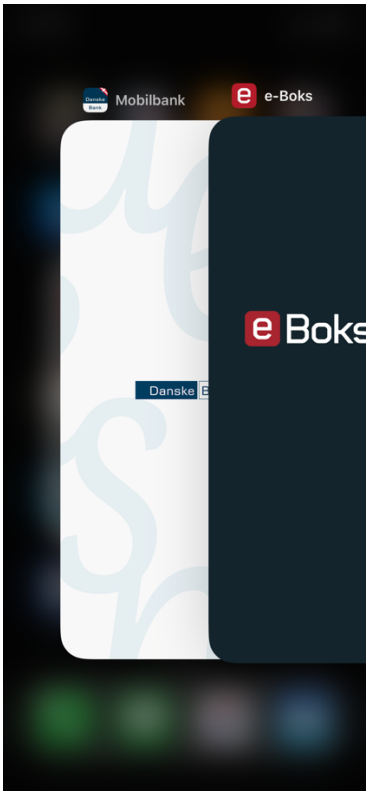


Figure 19. iOS Apps hiding their content when in the multitasking switcher screen

Since iOS 9, Apple has forced developers to use App Transport Security[37]. This forces applications to use an HTTPS connection, while returning an error for non-HTTPS connections. By having an SSL encrypted connection between the application and the server, it avoids having a Man-in-the-Middle type of attack.

"The Secure Enclave is a coprocessor fabricated in the Apple A7 or later A-series processor. It uses encrypted memory and includes a hardware random number generator. The Secure Enclave provides all cryptographic operations for Data Protection key management and maintains the integrity of Data Protection even if the kernel has been compromised."[38]

"Communication between the processor and the Touch ID sensor takes place over a serial peripheral interface bus. The processor forwards the data to the Secure Enclave but cannot read it. It's encrypted and authenticated with a session key that is negotiated using the device's shared

[37] https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW33

[38] iOS Security – White Paper November 2018 - https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf

key that is provisioned for the Touch ID sensor and the Secure Enclave. The session key exchange uses AES key wrapping with both sides providing a random key that establishes the session key and uses AES-CCM transport encryption."

Server security

SSL

SSL stands for secure socket layer, and it is a basic web security certificate, which also encrypts the channel between User (Point A) and Web application server (Point B), regardless which direction data is sent. Moreover, these certificates also help to authenticate the server's identity. HTTPS, HTTP Secure or HTTP over SSL takes advantage of SSL and it encrypts the data exchange between the web client and the server. This way, if someone were to sniff the packet exchange, they would not be able to see the contents. HTTPS is used to prevent Man-in-the-Middle attacks. The MITM attack is usually available when the exchange of raw or not encrypted data being done via the network. Attackers might monitor the network for example with packet sniffing application and see all packets going from point A to point B. Also, attackers might force the traffic through them and try to alter the messages on the way. MITM attack allows attackers to gain access to the transferred data (not encrypted or damaged), and infect the data.

At the time, Vapor 3 does not support SSL certificates. According to one issue[39] raised on GitHub, the only way to do this would be by adding NGINX in front of the Vapor sever, where TLS will be used. NGINX will act as an SSL/TLS enabled proxy.

Vapor 4, which is the yet-to-be-released next version of Vapor will add support for SSL/TLS without requiring an SSL/TLS enabled proxy. The Vapor 4 update requires the release of Swift 5 (which is already out) and the next version of Swift NIO (which is as well yet-to-be-released).

[39] https://github.com/vapor/documentation/issues/395

Password Hashing

According to the OWASP Password storing cheat sheet[40], there are several ways to store a password, but the document outlines never to store password in plain-text.

One way to store a password would be by hashing the password. “Hashing performs a one-way transformation on a password, turning the password into another String, called the hashed password. “One-way” means that it is practically impossible to go the other way - to turn the hashed password back into the original password. There are several mathematically complex hashing algorithms that fulfill these needs.”[41]

When the password is provided, the plain text version of the password is hashed using the same algorithm and the resulting hashed text is compared to the actual hashed valued of the password that is stored in the database. If the two hashed strings are a match, then the provided password is good, and the next step can commence. If they do not match, a retry of the password is asked. This would be a typical flow when using hashed passwords, since hashing is a one-way transformation function, as stated above.

Vapor has a package for the popular BCrypt hashing algorithm[42]. It is written is Swift. It offers basic functionality to hash but also to verify.

Token authentication

To protect the Vapor server even further than hashing the user’s password, token authentication was implemented.

For the Login API call, a basic authentication scheme was implemented. The “Basic” HTTP-authentication scheme[43] transmits the username and password, in the header of the HTTP request, and the username/password pair is encoded using base64.

[40] https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md
[41] https://docs.oracle.com/cd/E26180_01/Platform.94/ATGPersProgGuide/html/s0506passwordhashing01.html
[42] https://github.com/vapor-community/bcrypt
[43] https://tools.ietf.org/html/rfc7617

After a successful login, an authorization token is sent back to the client. The token is a randomly generated string, using the Crypto package[44].

In order to access any other resources from the server by making API calls, the client has to add the token to the HTTP header, under “Authorization” and append the word “Bearer” before the token string. This technique is inspired by RFC 6750[45], although that request for comments refers to accessing OAuth 2.0 protected resources. This is not the case.
The HTTP header looks like this: “Authorization”:” Bearer HquOpAzlOaBGYGB0pxrYew==”
The following table depicts the API calls available on the Vapor server and how they are protected:

API Call	Protection
GET vaporServer/	None
GET vaporServer/hello	None
POST vaporServer/api/users/login	“Basic” HTTP-authentication scheme
GET vaporServer/api/users	Authorization Bearer token
GET vaporServer/api/users/:id	Authorization Bearer token
PUT vaporServer/api/users/:id	Authorization Bearer token
POST vaporServer/api/users	Authorization Bearer token
DELETE vaporServer/api/users/:id	Authorization Bearer token
GET vaporServer/api/cars	Authorization Bearer token
GET vaporServer/api/cars/random	Authorization Bearer token
GET vaporServer/api/cars/:int	Authorization Bearer token

[44] https://github.com/vapor/crypto-kit
[45] https://tools.ietf.org/html/rfc6750

11. Business model

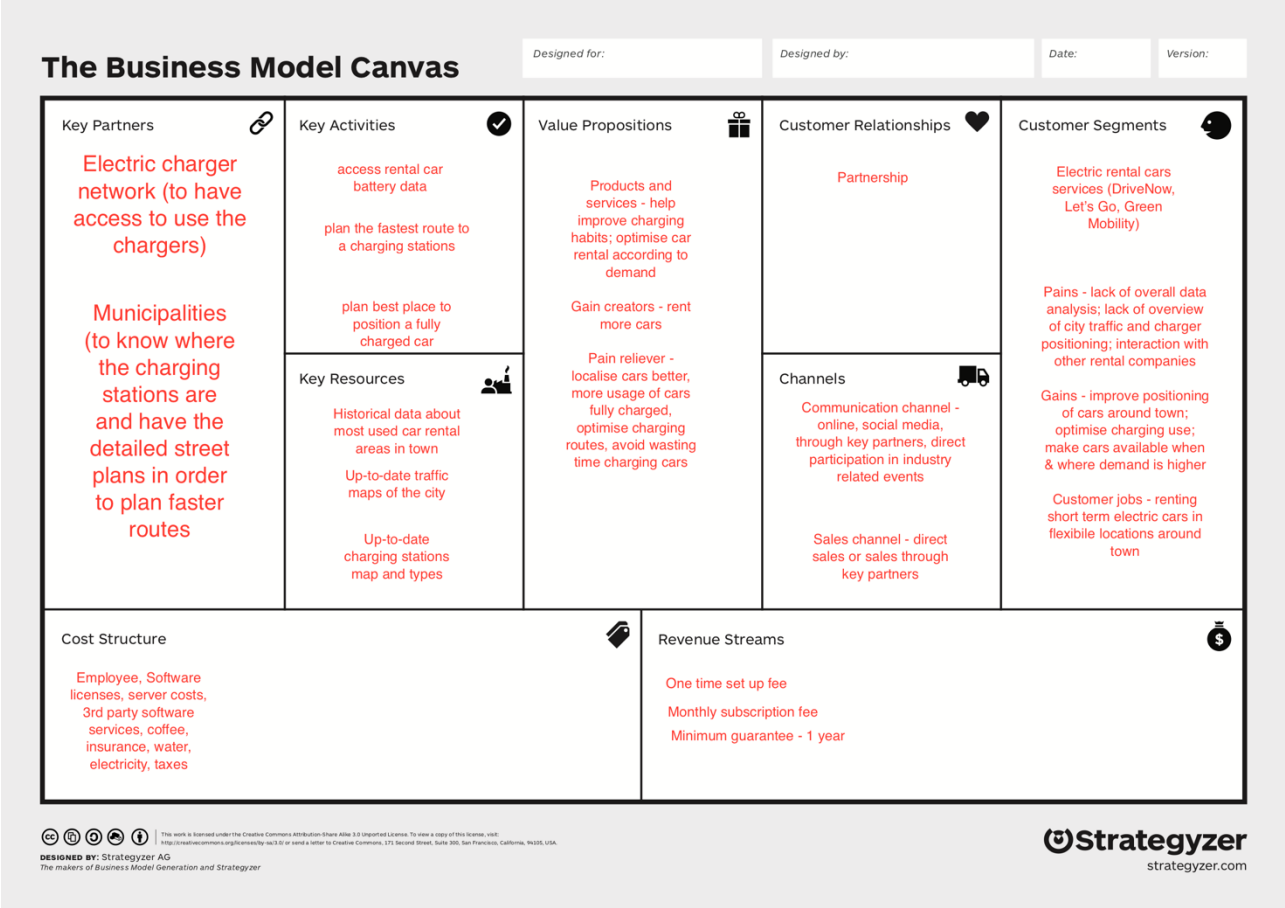
In terms of proposed business model, the project aims to answer the following question:

How could a system that will predict where to have the cars placed to charge, for maximum rental opportunity increment the company revenue?

The whole project is seen as service/platform offered to the above-mentioned companies. It will not be a consumer facing product, but a product to be used inside the company, enterprise application.
It will consist of a server, a ML service and a mobile application. Each night or at a certain amount of time, the server would ask the DriveNow server for the list of addresses of the cars to be put to charge. The server retrieves that list and it will send it to the Machine Learning service to have it sorted. The Machine Learning service will send it back to the server, which will forward it to the mobile application, to be used by the employee.

A native mobile application makes the most sense due to the fact that the employee is on the move all day. Access to the device's GPS is much easier from a native application than from a web application. Also, another point for native application is battery management. If the application is made using native SDK, the operating system can better manage the battery than a web application.

On the next page, it can be seen how this service is envisioned to fit in with the Business Model Canvas. After the Business Model Canvas overview, each of the 9 key points are explained.



Key Partners:

- the electric charger networks (as currently there are different distributors offering electric car charging stations) to have access to use their chargers
- municipalities – even if the electric charger networks have their own locations, the municipality should have a 3rd party, unbiased list of these chargers

Key Activities:

- access car battery status – as stated in the explanation of the project
- plan the fastest route to a charger – driving a car to a charger is a cost and driving to the closest charger can reduce that cost
- plan at which charger to leave the car – this is what the service wants to achieve

Key Resources:

- historical data about most user car rental areas in town – the whole service would not function without this dataset
- up-to-date traffic maps of the city – this can be accessed from the municipalities and it is going to help plan the fast route to the best suited charger

- up-to-date charging stations map and types – not all electric cars have the same electric socket; the map can be accessed from both the municipalities and the electric charging networks

Value Proposition:

- Products and services - help improve charging habits; optimize car rental according to demand
- Gain creators - rent more cars
- Pain reliever - localize cars better, more usage of cars fully charged, optimize charging routes, avoid wasting time charging cars

Customer Relationship:

- Partnership – by partnering up with the different electric car rental companies

Channels:

- Communication channel - online, social media, through key partners, direct participation in industry related events
- Sales channel - direct sales or sales through key partners

Customer segments:

- Electric rental cars services (DriveNow, Let's Go, Green Mobility)
- Pains - lack of overall data analysis; lack of overview of city traffic and charger positioning; interaction with other rental companies
- Gains - improve positioning of cars around town; optimize charging use; make cars available when & where demand is higher
- Customer jobs - renting short term electric cars in flexible locations around town

Cost structure:

- Employee, Software licenses, server costs, 3rd party software services, coffee, insurance, water, electricity, taxes

Revenue Streams:

- One time set up fee
- Monthly subscription fee
- Minimum guarantee - 1 year

At the end of the day, the rental company has revenue coming in when users rent/use their cars. By trying to predict the placement of charging cars in areas where the user demand is high, it will increase the company revenue, due to having the cars rented/used.

12. Discussion & future improvements

The first thing that should be improved, or included, would be the machine learning service.

The Phase 3 app predicts the electric charger where a car should be put to charge, but it does not do it using machine learning. It does it by using a simple "if statement" followed by 4 conditions. This gets the job done, but it does not scale and it is not a solution where a pattern can be observed and that pattern used in the coming future. The "if statement" solution does not scale due to the fact that the conditions are manually specified in code. What would happen if no charger fulfills the conditions? There will be nothing shown on the map. In that case, where would the car go charge?

This proves the importance of the dataset. If a more convincing dialogue would had happened when approaching one of the existing electric car rental companies, a viable dataset would have been an important assets in this project.
Looking at the AutoTel Shared Cars Availability dataset from Kaggle[46], it can be observed that AutoTel project relies heavily on statistics to try to make the predictions. The project also uses 2 datasets to make the predictions, one for the location of the cars, containing historical locations of the car up to several months back in time, and the other dataset contains the neighborhoods of Tel Aviv. These 2 datasets combined help make the prediction.

At the beginning of the project, Phase 1 seemed very similar to the Travelling Salesman Problem. Because of this similarity, the concept of graphs was introduced in the State of the Art chapter. In the actual prototype, graphs are completely unused and instead are replaced by the Google Maps Directions API.
As a future improvement, the Google Maps Directions API should be taken out and replaced by graphs. The node of the graph would hold the car information (battery, interest in the car), but also the charger information, and the edges would be the distance between the cars and the chargers.

[46] https://www.kaggle.com/gidutz/starter-autotel-shared-car-locations/notebook

The graph calculations can be done on device, but this affect the user experience and can render the mobile application unresponsive. The recommended place to perform the calculations for the graphs is the server. The problem at the moment with the Vapor server is that it does not support the distance calculations.
One solution to make this it work on the Vapor server would be to gather all the geographical locations and have them forward to the Google Maps API to create a distance matrix, that will be sent back to the server. Just like in the case of the Google Maps dependency in the iOS app, having the distance matrix created by Google is an extra dependency.

One noticeable improvement that can be implemented in the Vapor server, would be to create a GUI for it. At the moment, the only interactions that happens with the server is through HTTP request using tools like Postman, Paw or cURL. A web browser can be used to make the API calls, but the browser is not that easy to use in the case of other HTTP requests than GET.
It can be argued that by having no GUI for the Vapor server, it will be used by a set of trained employees only, those that had a training into using the Vapor server. The absence of the GUI will make the server inaccessible to all the company employees.

Regarding the token authentication request, this can pose a problem when combine with biometric login on the mobile phone application. The flow follows this path: the user provides the email and password in the application, a login API call is sent to the server, the server verifies that the email and password are present in the user database and send back a token. The token is then stored in the app and is subsequently used in all the following API calls.
By implementing biometric login, after the first successful login, the user is asked if the wants to take advantage of using this login method. This entitles the user to skip providing the email and password every time he logs in in the app. By skipping login screen, the app goes directly to the next API call, the one where to get the list of cars.
The problem arises because the app has not received a token from the server to be used in the header of the get list of cars request. As it is currently implemented, the token does not expire and previous token can be reused, which solves the problem, but it is not an ideal solution.

Ideally, a real authentication token should be implemented. An authentication token consists of 2 token: a bearer token that has a sooner expiration date and a refresh token that has a later expiration date. When the user logs in, he is given both token. For all the following API requests he uses the bearer token until it expires. When it expires, the server will refuse the connection. When this happens, the user sends the refresh token. Upon receiving the refresh token, the server will send a new bearer and refresh token back to the user.

13. Conclusion

The project started from the willingness to solve a problem a company is facing during day-to-day operations.
In order to tackle it, a problem formulation has been stated:

"How could an electric charger prediction system based on machine learning be a solution for maximizing rental opportunities for an electric rental car fleet?"

Several sub questions have been devised to help answer the problem formulation. They are shown below but they are accompanied by a short answer:

- Why choose machine learning?

  A pattern of usage can be identifies, by choosing machine learning, in the case of the rental cars. The only condition for this to work is the availability of a viable dataset. The pattern of usage can help the company identify unknown trends and better act on them.

- What can be used instead of machine learning?

  As it can be seen in the Phase 3 app and in the Discussion and Future improvements chapter, machine learning can be replaced by a simple if statement containing 4 conditions. The if statement does work and gives results, but it is not a scalable

solution and it requires human intervention when the results are not the desired one.

- What does “solution” refer to?

  Solution refers to the prototype. As it can be seen in the Development chapter, the ideal solution consists of 4 elements, while the prototype consists of only 2.

- What does “prediction system” mean?

  Prediction is one of the use cases of machine learning. Classical examples of machine learning prediction refer to the sales price of a house, given that certain properties of the house are known, and the same information is available for the houses from the same neighborhood/area/city.

To answer the problem formulation, the project started out with just an idea and tried to come up with a solution that can solve the problem captures by that idea. The presented solution failed to include machine learning, but instead relies on manual calculations. The failure can be attributed to the lack of a viable dataset, which could not be acquired.

14. Bibliography

1. https://www.raywenderlich.com/773-swift-algorithm-club-graphs-with-adjacency-list - Ray Wenderlich website, Swift Algorithm Club – Graphs, last accessed on 5.06.2019
2. Idem as number 1
3. http://mathworld.wolfram.com/AdjacencyMatrix.html - Wolfram Mathworld, last accessed 5.06.2019
4. https://www.khanacademy.org/computing/computer-science/algorithms/graph-representation/a/representing-graphs – Khan Academy, Representing Graph, last accessed 5.06.2019
5. Idem as number 1
6. https://github.com/Avik-Jain/100-Days-Of-ML-Code – Github, 100 days of ML Code, last accessed 5.06.2019
7. Idem as number 6
8. https://ml-cheatsheet.readthedocs.io/en/latest/nn_concepts.html – Neural Networks Concepts, last accessed 5.06.2019
9. https://victorzhou.com/blog/intro-to-neural-networks/ - Introduction to Neural Networks, last accessed 5.06.2019
10. Idem as number 9
11. https://azure.microsoft.com/en-us/services/machine-learning-studio/ - Microsoft Azure Machine Learning Studio, last accessed 5.06.2019
12. https://aws.amazon.com/sagemaker/ - Amazon Web Service Sage Maker, last accessed 5.06.2019
13. https://www.ibm.com/cloud/machine-learning – IBM Watson Machine Learning, last accessed 5.06.2019
14. https://cloud.google.com/ml-engine/ - Google Cloud Machine Learning Engine, last accessed 5.06.2019
15. https://www.tensorflow.org – Tensorflow, last accessed 5.06.2019
16. https://bigml.com/ - BigML, last accessed 5.06.2019
17. https://vapor.codes – Official website for Vapor, Server-side Swift, last accessed 5.06.2019
18. https://www.sqlite.org – Official website for SQLite, last accessed 5.06.2019

19. https://github.com/apple/swift-package-manager – Github, Swift Package Manager, last accessed 5.06.2019
20. Idem as number 19
21. https://swift.org/about/ - Official Swift website, last accessed 5.06.2019
22. https://developer.apple.com/xcode/ - Xcode official website, last accessed 5.06.2019
23. https://cocoapods.org – Official Cocoapods website, last accessed 5.06.2019
24. https://cocoapods.org/pods/GoogleMaps – Google Maps on Cocoapods, last accessed 5.06.2019
25. https://github.com/hackiftekhar/IQKeyboardManager - Github, IQKeyboardManager, last accessed 5.06.2019
26. https://github.com/Alamofire/Alamofire - Github, Alamofire, last accessed 5.06.2019
27. http://www.geomidpoint.com/random/ - Generate random geographical points, last accessed 5.06.2019
28. https://www.random.org/integers/ - Generate random integers, last accessed 5.06.2019
29. https://chargemap.com/map – Chargermap.org, last accessed 5.06.2019
30. https://openchargemap.org/site – Openchargemap.org, last accessed 5.06.2019
31. https://openchargemap.org/site/develop/api – Openchargermap.org API, last accessed 5.06.2019
32. https://api.openchargemap.io/v3/poi/?output=json&countrycode=DK&maxresults=200&latitude=55.6761&longitude=12.5683&distance=20&distanceunit=KM – API call for getting all the electric chargers around Copenhagen
33. https://www.ibm.com/developerworks/rational/library/4706.html - Peter Eeles, Capturing Architectural Requirements, 2005, last accessed 5.06.2019
34. https://www.charlesproxy.com – Charles Proxy, last accessed 5.06.2019
35. https://developer.apple.com/documentation/foundation/userdefaults – UserDefaults documentation, last accessed 5.06.2019
36. https://cloudknox.io/authentication-vs-access-controls-vs-authorization/ - Authentication vs access controls vs authorization, last accessed 5.06.2019

37. https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CocoaKeys.html#//apple_ref/doc/uid/TP40009251-SW33 - App Transport Security, last accessed 5.06.2019
38. https://www.apple.com/business/site/docs/iOS_Security_Guide.pdf - iOS Security – White Paper November 2018, last accessed 5.06.2019
39. https://github.com/vapor/documentation/issues/395 – Github, Vapor SSL support issue, last accessed 5.06.2019
40. https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Password_Storage_Cheat_Sheet.md – OWASP, Password Storage cheat sheet, last accessed 5.06.2019
41. https://docs.oracle.com/cd/E26180_01/Platform.94/ATGPersProgGuide/html/s0506passwordhashing01.html – Oracle Documentation, Password hashing, last accessed 5.06.2019
42. https://github.com/vapor-community/bcrypt – Github, Vapor Bcrypt, last accessed 5.06.2019
43. https://tools.ietf.org/html/rfc7617 - The 'Basic' HTTP Authentication Scheme, last accessed 5.06.2019
44. https://github.com/vapor/crypto-kit – Github Vapor Crypto-kit, last accessed 5.06.2019
45. https://tools.ietf.org/html/rfc6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage, last accessed 5.06.2019
46. https://www.kaggle.com/gidutz/starter-autotel-shared-car-locations/notebook – Kaggle, Autotel shared car locations, last accessed 5.06.2019