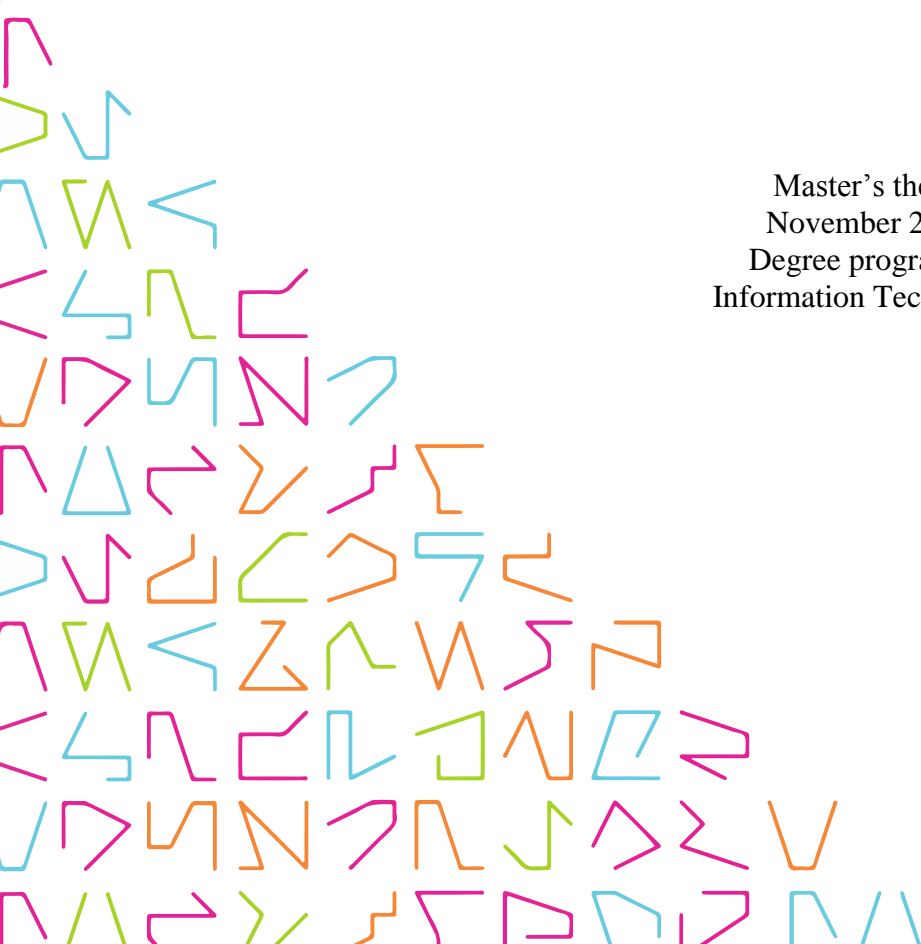


PARK HERE

A Web Based Application for Finding Available Parking Spaces
in Tampere, Finland

Omokaro Richardson Bob

Master's thesis
November 2018
Degree program in
Information Technology



ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Masters in Information Technology

Omokaro Richardson Bob

A web based application for finding available parking spaces in Tampere,
Finland

Master's thesis 41 pages, appendices 3 pages
November 2018

This master's thesis focuses on a real time information collection and monitoring system. The aim is to show and provide information on available car parking slots in Tampere region. This study proposes that real time information and monitoring system can be achieved by utilizing Finnpark API.

Contemporary urban rising is witnessing huge traffic as more vehicles are on the road on a daily basis. According to Statistics Finland, over 5,045,365 cars were in traffic use in 2017. The primary purpose is to ease movement of persons, goods and services from one point to another. These movements often create significant challenges such as road congestion, for all road users. In addition to road congestion, finding parking spaces in public car parks is a challenge for car owners.

Oftentimes, vehicle owners are fined for parking in unauthorized places or even on the road side thereby causing traffic gridlock for other road users.

By utilizing a web interface powered by Finnpark API, this study provides real time update of available parking slots in Tampere region. After inputting their desired destination in the web interface, car owners are provided with available parking spaces. This study proposes that with the use of this web interface, car owners will save valuable time and money involved in finding suitable parking lots.

Key words: web application, real time, car parking slot, Finnpark API

CONTENTS

1	INTRODUCTION	8
2	COMPONENT REQUIREMENT AND TECHNOLOGIES	11
2.1	CLIENT-SIDE PROGRAMMING.....	11
2.1.1	AngularJS framework	11
2.1.2	Bootstrap framework.....	12
2.1.3	JQuery library.....	12
2.1.4	Google maps	12
2.1.5	Google Maps API.....	13
2.1.6	Embed Google map.....	13
2.1.7	REST API	15
2.1.8	How RESTful APIs work	16
2.2	SERVE-SIDE PROGRAMING.....	17
2.2.1	DATEX2	17
2.2.2	XML.....	18
2.2.3	Finnpark API.....	19
2.3	IDE.....	20
2.3.1	Visual studio Code	20
3	SYSTEM ARCHITECTURE AND USE CASES	21
3.1	System Architecture.....	21
3.1.1	HttpClient.....	22
3.2	Use Case Diagram	22
3.3	Class Diagram.....	23
3.4	ER Diagram	25
4	DESCRIPTION OF SYSTEM AND API	27
4.1	Data Contents.....	28
4.1.1	Parking facilities basic information.	28
4.1.2	Parking facility table publication	29
4.1.3	Parking facility status information	29
5	SYSTEM DEVELOPMENT AND TESTING	32
5.1	System Development	32
5.1.1	Adding bootstraps and JQuery	33
5.2	Testing	35
6	USER INTERFACE IMPLEMENTATION.....	36
6.1	Front Page	36
6.2	Input Destination.....	37
6.2.1	Parking Facilities.....	37

6.2.2 Availability.....	38
6.3 Get Direction.....	39
7 DISCUSSION	40
7.1 Future Development.	40
8 REFERENCES	41
9 APPENDICES.....	42
Appendix 1	42
Finnpark DATEX 11 parking extension table.....	42
https://www.dropbox.com/sh/ujs2hoim8z9py2y/AABIplU25NWZktVokYbZ MWk_a?dl=0	42
Appendix 2	42
Finnpark parking data API source code	42
http://parkingdata.finnpark.fi:8080/Datex2/OpenData	42
Appendix 3	42
Front end code	42
Appendix 4	44
Implementing map component code	44

ABBREVIATIONS AND TERMS

XML	Extensible Markup Language
SGML	Standard Generalized Markup Language
UML	Unified Modeling Language
IDE	Integrated Development Environment
REST	Representational State Transfer
MVC	Model, View, Controller software architecture style
CSS	Cascading Style Sheets
HTML	Hypertext Mark-up Language
JS	JavaScript commonly used programming language for Web programming
GUI	Graphical User Interface allows users to interact with electronic devices
API	Application Programming Interface
FINNPARK	Tampere Parking agency
ER	Entity Relationship

LIST OF FIGURES

Figure 1: Standard carpark scenario.....	8
Figure 2: System overview.....	10
Figure 3: Snap shot of embedded google map.....	15
Figure 4: REST API architecture diagram.....	17
Figure 5: System Architecture diagram.....	21
Figure 6: Vehicle driver use case diagram.....	23
Figure 7: Parking system class diagram (hansOnUML, 2018).....	25
Figure 8: ER Diagram for Smart Parking (Super intelligence parking system (SIPS))	26
Figure 9: Shows Google search parking pin points in Tampere.....	27
Figure 10: Shows the parking information in XML format.....	28
Figure 11: Node.js setup for windows.....	32
Figure 12: Creating angular project, parkhere test.....	33
Figure 13: Adding bootstrap and JQuery to Angular cli.....	34
Figure 14: Bootstrap scripts needed in scripts and Bootstrap css file in styles in angular- cli.json file.....	34
Figure 15: Node and NPM installation testing.....	35
Figure 16: Parkhere interface.....	36
Figure 17: Parkhere inputing destination.....	37
Figure 18: Google maps destination results.....	38
Figure 19: Parkhere results using google maps.....	39
Figure 20: Integration of google maps in Parkhere.....	39

LIST OF TABLES

Table 1: Shows the UML packages.	28
Table 2: DATEXII Parking Extension: parkingFacilityTablePublication	29
Table 3:DATEXII Parking Extension: parkingFacilityTableStatusPublication	30

1 INTRODUCTION

With a steady increase in urbanization, towns and cities have experienced an exponential growth in the daily use of cars. According to Statistics Finland, over 5,045,365 cars was in traffic use in 2017. The primary purpose was to ease movement of persons, goods and services from one point to another.

These movements often create a significant level of traffic congestion for all road users. With the rising increase in car usage, public and private parking slot have also become a challenging utility. This includes finding parking slots close to the desired destination for activities such as transacting business, shopping, social activities, etc.

Also, where parking slots are available, it is usually a significant distance away from the desired destination. Thus, car owners often drive slowly around their desired parking slots until an opening becomes available. Hence the phenomenon of cruising for parking (Shoup, 2006) in congested places. Cruising for parking refers to car owners continuously driving in a vicinity until they find an available parking slot. Besides the cost of time, money and gas, cruise parking has a negative impact on the overall economic progress of the affected area. Thus, prior knowledge of where to park your car when you arrive at your destination will save you time and gas if you already have an idea of the closet available parking spot.



Figure 1: Standard carpark scenario

In addition, vehicle owners are often fined for parking in unauthorized places or even on the road side thereby increasing the chances of traffic gridlock for other road users.

This thesis proposes that to avoid cruise parking and alleviate car congestion, car owners can be provided a platform equipped with real time update of available parking slots are provided on request. To achieve this, the API from Finnpark is utilized. Tampere region is used as a case study to show the effectiveness and profitability of this study.

Tampere houses a population of 231,853 inhabitants (Tampere-city, 2018). It is the second largest city in Finland with an active urban environment. Although transportation system in Tampere is very effective, the region still experiences cruising for parking; hence the emergence of smart parking apps such as Easypark, Finnpark, and Parkman.

These apps are paid services offering available parking slots to users for a fee. In addition to preventing cruising parking, users get to pay for only what they use which eliminates fines for overparking or underutilizing paid parking time.

A common limitation of these apps is that they do not provide information on available free parking slots, hence the web application service analyzed in this thesis.

The web application uses the FinnPark API to search for available free parking slots for users.

The park here system entails the following:

- Destination: the user inputs desired destination in to the search browser button
- Available Parking facilities: park here display all available parking facilities nearby.
- Get direction: the get direction links the user to Google map for fastest route to the desired location.

This is shown in the figure 1.

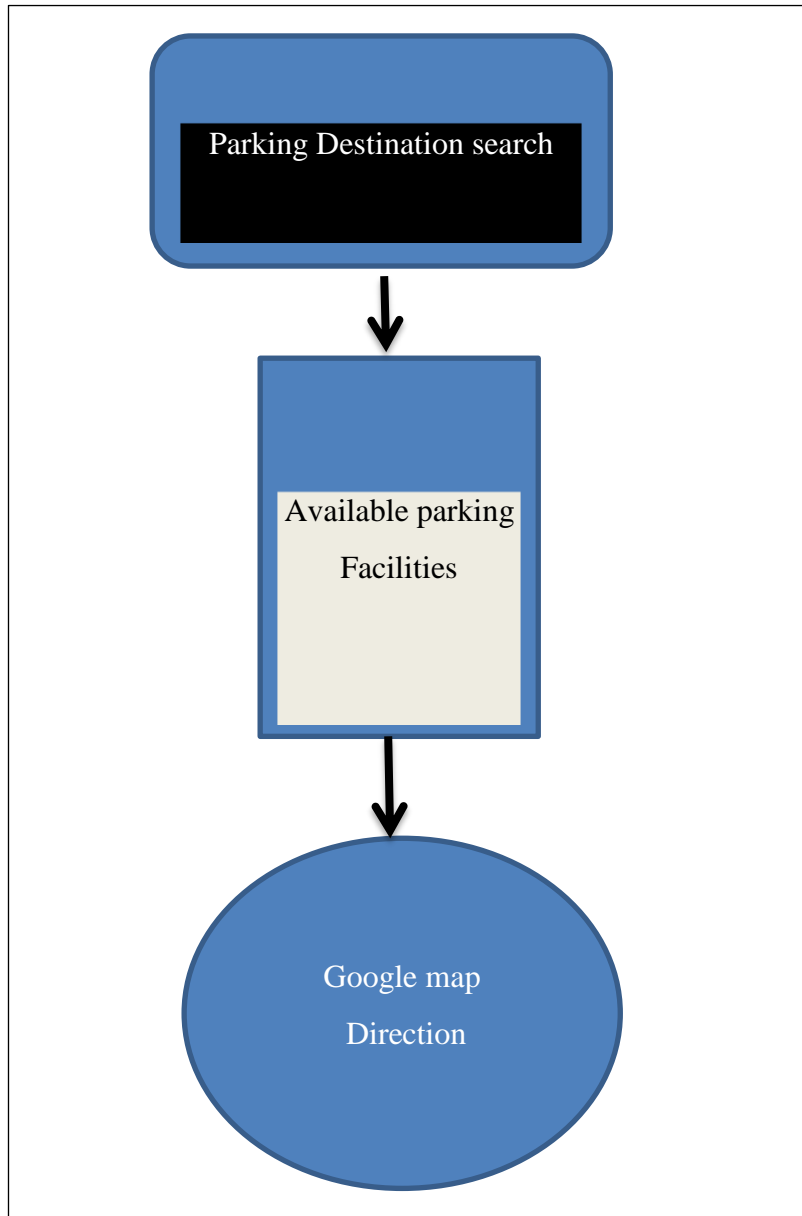


Figure 2: System overview

2 COMPONENT REQUIREMENT AND TECHNOLOGIES

The purpose of this thesis is to deploy a web based parking application that shows available parking spaces in the Tampere region of Finland.

This thesis is been deployed and implemented with already existing technologies. For this section, we will be explaining the components and technology used in this thesis.

2.1 CLIENT-SIDE PROGRAMMING

Client side programming has mostly to do with the user interface, with which the user interacts. In web development it's the browser, in the user's machine, that runs the code, and it's mainly done in JavaScript, flash, HTML, CSS and any language running on a client device that interacts with a remote service is a client-side language.

2.1.1 AngularJS framework

Angular JS (AngularJS, 2018) is a JavaScript-based open-source front-end web application framework mainly maintained by Google and by a community of individuals and corporations to address many of the challenges encountered in developing single-page applications.

The JavaScript components complement Apache Cordova, a framework used for developing cross-platform mobile apps. It aims to simplify both the development and the testing of such applications by providing a framework for client-side model–view–controller (MVC) and model–view–view model (MVVM) architectures, along with components commonly used in rich Internet applications. In 2014, the original AngularJS team began working on the Angular application platform (Fluin, 2018).

The AngularJS framework works by first reading the HTML page, which has additional custom tag attributes embedded into it. Angular interprets those attributes as directives to bind input or output parts of the page to a model that is represented by standard JavaScript variables. The values of those JavaScript variables can be manually set within the code, or retrieved from static or dynamic JSON resources (Fluin, 2018).

2.1.2 Bootstrap framework

Bootstrap is a free and open-source front-end framework (library) for designing websites and web applications. It contains HTML and CSS-based design templates for typography, forms, buttons, navigation and other interface components, as well as optional JavaScript extensions. Unlike many web frameworks, it concerns itself with front-end development only.

Advantages of bootstrap are:

- **Ease of Use:** Bootstrap provides less files for those who are accustomed to CSS pre-processing. However, for users who do not know how to use it, they have the option of having the traditional plain CSS files instead.
- **Highly Flexible:** Bootstrap gives the developers the flexibility to develop. It is a CSS framework with predefined classes for layout using its grid system, various CSS components and JavaScript functions.
- **Responsive Grid:** This is the strongest part of the bootstrap framework. Bootstrap offers a 12 column grid system. The grid system is responsive, that is it adjusts itself depending on the device resolution of the client.
- **Big List of Components:** Bootstrap has all the components you would require for your website. It includes Drop down Menus, Navigation Bar, Progress Bar, Alerts, Labels, and Badges etc.

2.1.3 JQuery library

Bootstrap comes with several JavaScript components in the form of JQuery plugins. They provide additional user interface elements such as dialog boxes, tooltips, and carousels. They also extend the functionality of some existing interface elements, including for example an auto-complete function for input fields.

2.1.4 Google maps

Google Maps is a web mapping service developed by Google. It offers satellite imagery, street maps, 360° panoramic views of streets (Street View), real-time traffic conditions

(Google Traffic), and route planning for traveling by foot, car, bicycle (in beta), or public transportation.

Google Maps uses JavaScript extensively. As the user drags the map, the grid squares are downloaded from the server and inserted into the page.

For the purpose of this thesis I will only make use of the route planner component.

2.1.5 Google Maps API

Google Maps API allows developers to integrate Google Maps into their websites. It was a free service that requires an API key. Web sites use the Google Maps API, making it the most heavily used web application development API.

The Google Maps API is free for commercial use, provided that the site on which it is being used is publicly accessible and does not charge for access.

2.1.6 Embed Google map

Google map's interactive features which includes accurate directions, location details, 360 degree street level panoramas zooming, panning and satellite views makes it the most popularly used online mapping service. In addition, the friendly user interface which is regularly updated with new improved features facilitates its seamless operational processes.

Part of the seamless operational process of google maps includes the ability to embed it easily on a website.

Google map on a website provides the site with authority, trustworthiness, professionalism and confidence for end users. A characteristic feature of google map is the use of a pin to accurately pinpoint a location. This is particularly useful for businesses because of its specificity in providing not only the exact location of an address but also its surrounding neighborhood.

Benefits of embedding google map on a website includes:

1. Especially for the less tech savvy, google map provides accurate directions to a specified address for users. This saves time and prevents inaccurate copying of address into a native map application.
2. Google map also presents the users with easy access to other data such as phone number, website address, and locality.
3. Other areas of local interest are also presented to users when accessing google map on a website. This can include popular location on interest close to the specific address.
4. Availability of 360 street view panorama ensures that users can find the specified address thus reducing the chance of going the wrong way.
5. To boost confidence and trustworthiness, google map also shows verified reviews from other customers.
6. Importantly, interacting with google map keeps users on a website. This is valuable for search engine optimization issues.

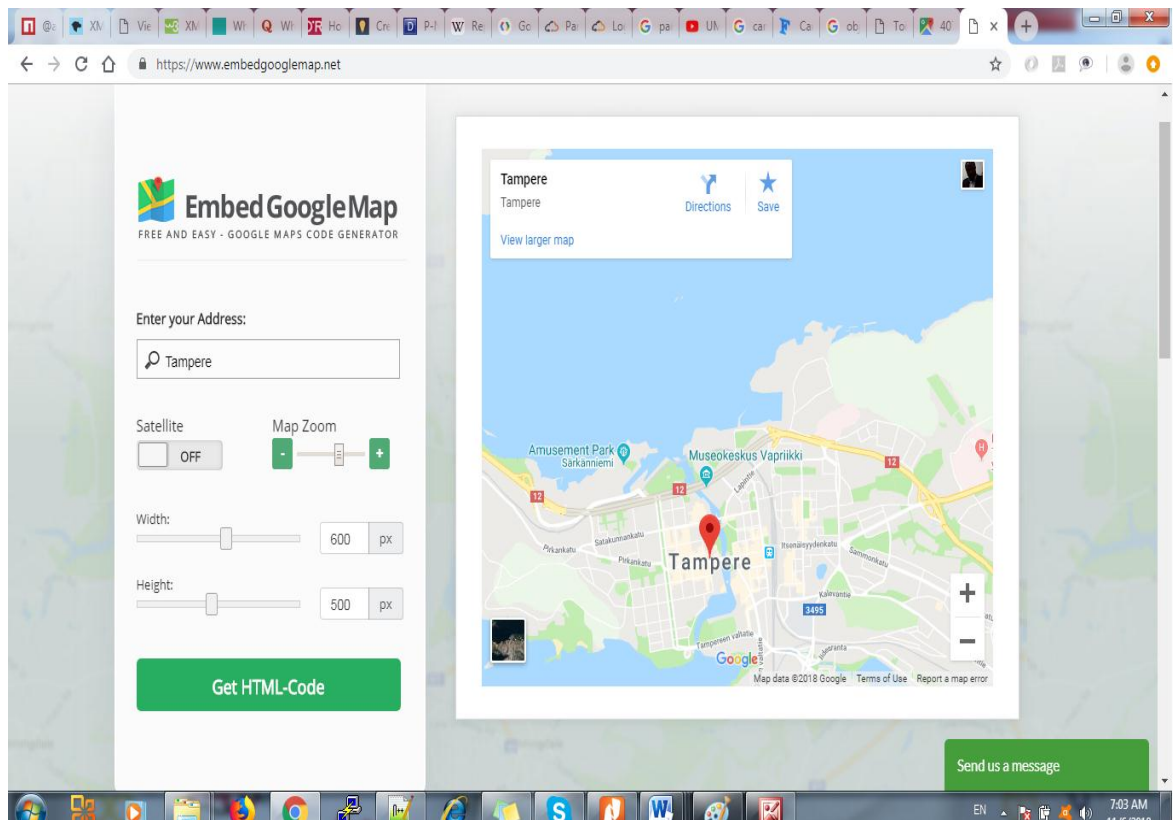


Figure 3: Snap shot of embedded google map

2.1.7 REST API

A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.

A RESTful API also referred to as a RESTful web service is based on representational state transfer (REST) technology, an architectural style and approach to communications often used in web services development.

REST technology is generally preferred to the more robust Simple Object Access Protocol (SOAP) technology because REST leverages less bandwidth, making it more suitable for internet usage. An API for a website is code that allows two software programs to communicate with each another. The API spells out the proper way for a developer to write a program requesting services from an operating system or other application.

The REST used by browsers can be thought of as the language of the internet. With cloud use on the rise, APIs are emerging to expose web services. REST is a logical

choice for building APIs that allow users to connect and interact with cloud services. RESTful APIs are used by such sites as Amazon, Google, LinkedIn and Twitter

2.1.8 How RESTful APIs work

A RESTful API breaks down a transaction to create a series of small modules. Each module addresses a particular underlying part of the transaction. This modularity provides developers with a lot of flexibility, but it can be challenging for developers to design from scratch. Currently, the models provided by Amazon Simple Storage Service, Cloud Data Management Interface and OpenStack Swift are the most popular.

A RESTful API explicitly takes advantage of HTTP methodologies defined by the RFC 2616 protocol. They use GET to retrieve a resource; PUT to change the state of or update a resource, which can be an object, file or block; POST to create that resource; and DELETE to remove it.

With REST, networked components are a resource you request access to -- a black box whose implementation details are unclear. The presumption is that all calls are stateless; nothing can be retained by the RESTful service between executions.

Because the calls are stateless, REST is useful in cloud applications. Stateless components can be freely redeployed if something fails, and they can scale to accommodate load changes. This is because any request can be directed to any instance of a component; there can be nothing saved that has to be remembered by the next transaction. That makes REST preferred for web use, but the RESTful model is also helpful in cloud services because binding to a service through an API is a matter of controlling how the URL is decoded. Cloud computing and microservices are almost certain to make RESTful API design the rule in the future.

A typical REST API architecture diagram is shown in figure 4 below.

REST API Architecture

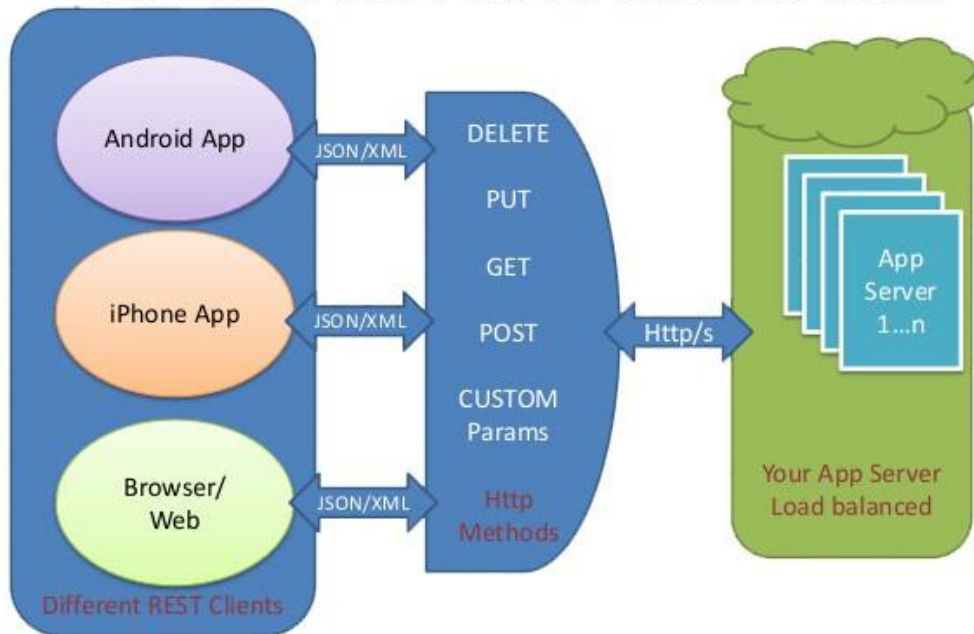


Figure 4: REST API architecture diagram

2.2 SERVE-SIDE PROGRAMING

In this thesis, I will be using the Finnpark API. Data content of the interface:

- location
- name
- address

The interface content has been made as a DATEX2 extension.

2.2.1 DATEX2

Datex2 (European-Union, 2018) is a data exchange standard for exchanging traffic information between traffic management centers, traffic service providers, traffic operators and media partners. It contains for example traffic incidents, current road works and other special traffic-related events. These data is presented in XML-format and is mod-

eled with UML. The standard is developed by the technical body intelligent transport systems

2.2.2 XML

XML (w3schools.com, 2018) is a file extension for an Extensible Markup Language (XML) file format used to create common information formats and share both the format and the data on the World Wide Web, intranets, and elsewhere using standard ASCII text.

XML is similar to HTML. Both XML and HTML contain markup symbols to describe the contents of a page or file. HTML, however, describes the content of a Web page (mainly text and graphic images) only in terms of how it is to be displayed and interacted with. For example, the letter "p" placed within markup tags starts a new paragraph.

XML describes the content in terms of what data is being described. For example, the word "phonenumber" placed within markup tags could indicate that the data that followed was a phone number. An XML file can be processed purely as data by a program or it can be stored with similar data on another computer or it can be displayed, like an HTML file. For example, depending on how the application in the receiving computer wanted to handle the phone number, it could be stored, displayed, or dialed.

XML is considered extensible because, unlike HTML, the markup symbols are unlimited and self-defining. XML is a simpler and easier-to-use subset of the Standard Generalized Markup Language (SGML) standard for how to create a document structure. It is expected that HTML and XML will be used together in many Web applications. XML markup, for example, may appear within an HTML page.

The reason for creating the XML markup are;

- It simplifies data sharing
- It simplifies data transport
- It simplifies platform changes
- It simplifies data availability

Example of how to represent an XML markup file formats is shown below,

```
<? xml version="1.0" encoding="UTF-8"?>
- <note>
  <to>Tampere</to>
  <from>Jyvaskyla</from>
  <heading>Destination</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

2.2.3 Finnpark API

Finnpark API is an open data DATEX2 web service that provides data from parking facilities in Tampere. The data format is in XML and the standard is the ITS_Standards|DATEXII. It was made available to the public since February 2005. The data is maintained by finnpark Oy.

The data content includes;

- Parking facility basic information: location, name, address
- DATEXII Parking Extension: parking Facility Table Publication.
- Parking facility status information: open, closed, spaces Available, almost full, full At Entrance, full, status unknown, normal Parking Conditions Suspended and special Parking Conditions in Force
- DATEXII Parking Extension: parking Facility Table Status Publication

The URL for the API access is <http://parkingdata.finnpark.fi:8080/Datex2/OpenData>

Finnpark provides more DATEXII information about parking facilities upon agreement.

These details include the following;

- more detailed facility parameters (capacity, structure, payment, services, max. vehicle sizes, etc.)
- more detailed occupancy information
- occupancy forecast
- similar data from street parking
- similar data from other cities in Finland.

2.3 IDE

An IDE is application that provides the environment for software developers to develop their code. They are simply code editors which help to facilitate code completion, indentation, testing, debugging and lots more.

2.3.1 Visual studio Code

Visual studio code is a code editor created by Microsoft development team. Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, like IntelliSense code completion and debugging.

Visual studio code editor is free to download that is why I choose to use it as the code editor for this thesis.

3 SYSTEM ARCHITECTURE AND USE CASES

3.1 System Architecture

In this thesis we are using Angular js for the client-side and connected to API from Finnpark. This is because Finnpark Tampere already has provided the API to their data base parking facilities. Therefore, client-side developer just needs to know the URL to the REST API of Finnpark.

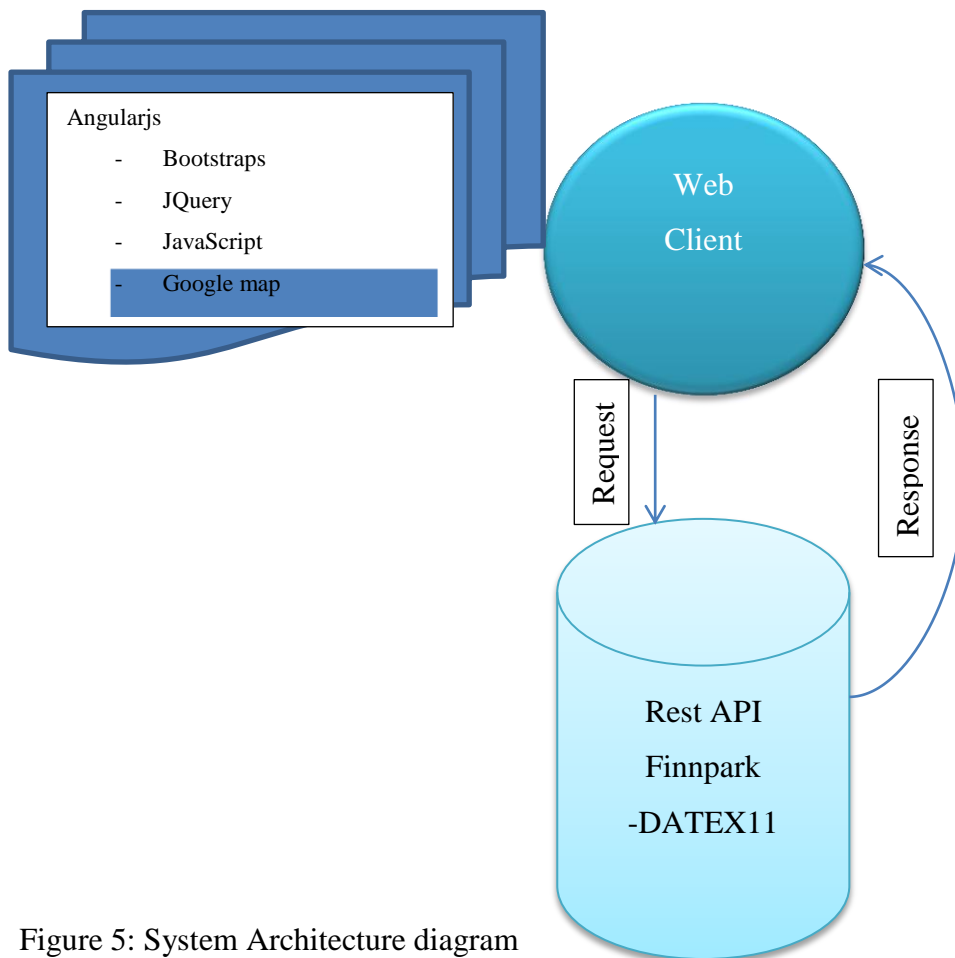


Figure 5: System Architecture diagram

3.1.1 HttpClient

Client-side applications mostly communicate with backend services over the HTTP protocol. Today's browser supports two types of APIs for making HTTP requests. The *XMLHttpRequest* interface and the *fetch ()* API.

In Angular applications, the HttpClient in *@angular/common/http* gives a simplified HTTP API that rests on the *XMLHttpRequest* interface visible by browsers. The advantages of the HttpClient are:

- Its test features,
- typed request and response objects,
- request and response interception,
- streamlined error handling

3.2 Use Case Diagram

In software engineering use cases diagram are very important to describe a specific way of using the software from the client-side who is the so user of the software. Therefore, a use case diagram is a UML (Unified Modeling Language) is a general-purpose, developmental, modeling language in the field of software engineering that is intended to provide a standard way to visualize the design of a system.

Some of the advantage of use case is that;

- Simplify system requirements
- Communicate with the end users and domain experts
- Test the system

Vehicle driver who is the system user searches for available vacant parking space prior to starting the journey and the use cases is the step by step sequence of actions that provide something of measurable value to the user like inspection the parking facilities for available parking vacant spaces.

It also provides history to find out which parking facilities are most occupied and less occupied. So, the driver finally decides which parking facility is best to park the vehicle.

The use case diagram below gives some details for vehicle drivers

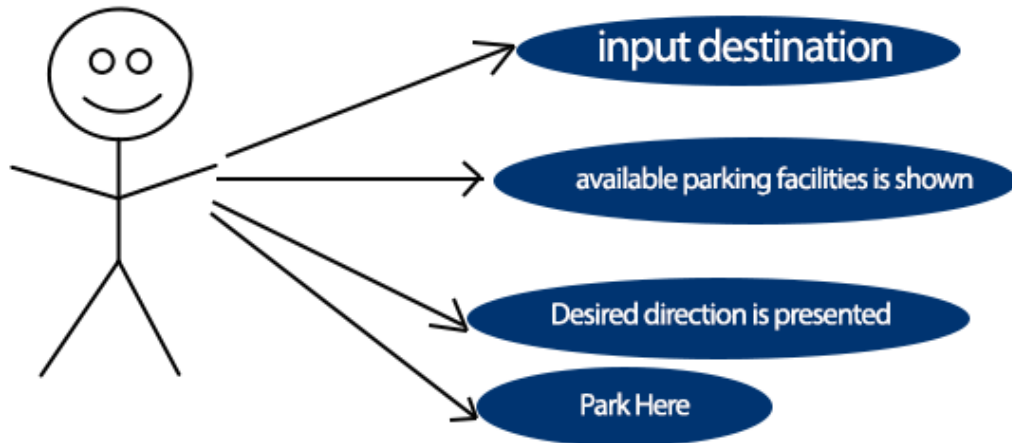


Figure 6: Vehicle driver use case diagram

3.3 Class Diagram

A class diagram represents the static view of an application. The class diagram represents the main building block activities in object-oriented language that are used to show the different objects in the system, their attributes, operations and the relationships of the classes. The figure 5 below shows a class diagram for parking system.

The class diagram consists of three major objects shown on a rectangular box- class, attributes and methods. The class is the name of the system to be design. The class name in this project is called the parking structure. The attributes shows a significant piece of data containing values that describe each instance of that class also known as fields, variables, properties.

The attributes of the class Parking Structure is city, address, type. The city variable is string, address also return string and the type returns structure type. In this case there is a composition relationship between the class name Parking Structure and the class name

Parking Level. The Parking Level attributes is FI Number (floor number) that is return as an integer. All attributes is made public that is every one can access the system.

The class name parking space also has a composition relationship to Parking Level that means the parking space class cannot exist without the Parking Level. There are three attributes to the class Parking Space. These attributes are Space Number (integer), Floor Number (parking level), and Space type (parking space type).

The methods in this Parking Space is park, unpark, getFeeAmount, pay. The other inheritances in the system from the class Parking Space are Regular Space, Handicapped Space, and Vehicle Interface whose inheritance relationships are car, truck and motorbike. The car class has an inheritance name class Electric and Convertible.

Other information that needs to be considered in the class diagram are class name;

- Structure Type whose attributes is either Public or Private.
- Parking Space Type – attributes are Regular, Compact, Handicapped and Motorbike.

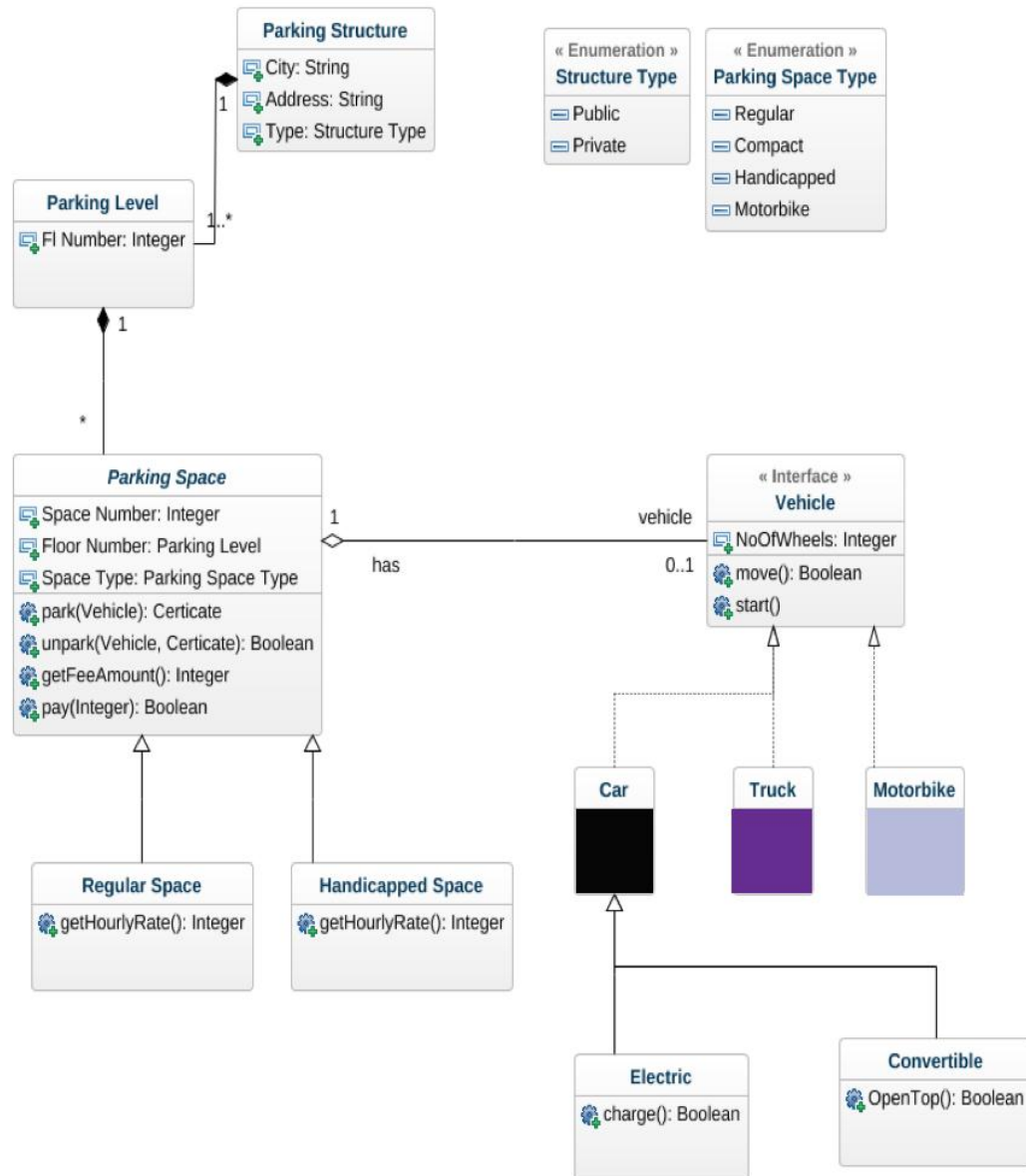


Figure 7: Parking system class diagram (hansOnUML, 2018).

3.4 ER Diagram

An ER diagram shows the basic logical structure of the data base. Figure 8 below shows basic features of a simple smart parking system as represented in the data base of the system. For the purpose of this thesis I am not going into details explanation between the entity, attribute and action relationships because the data base for this project is an open API from Finnpark. Hence I do not have the privilege right into the core data base ER diagram. The figure is just to show what basic information requirement for a smart parking database.

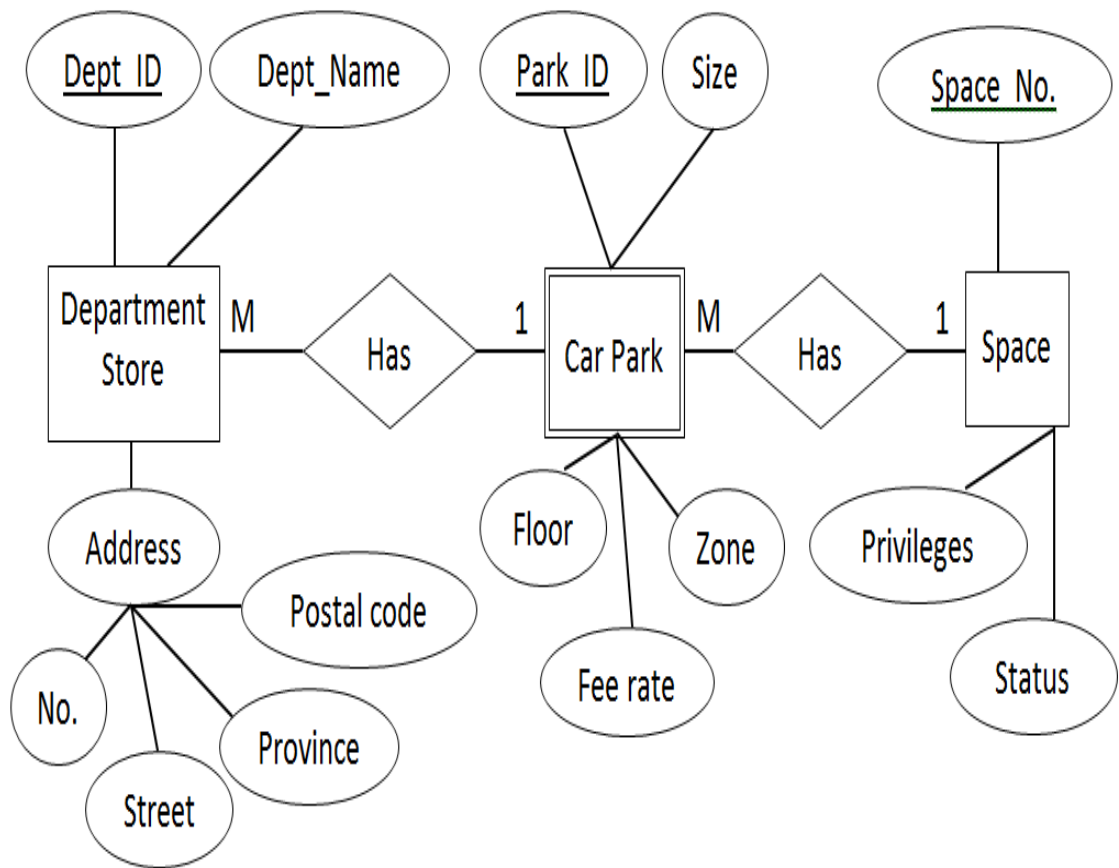


Figure 8: ER Diagram for Smart Parking (Super intelligence parking system (SIPS))

4 DESCRIPTION OF SYSTEM AND API

In this chapter we shall go through all the implementation processes. The system is implemented in with the user interface which consists of three main stages; destination, availability of parking spaces, and the get direction which is linked with Google map (Google, 2018) for drive direction.

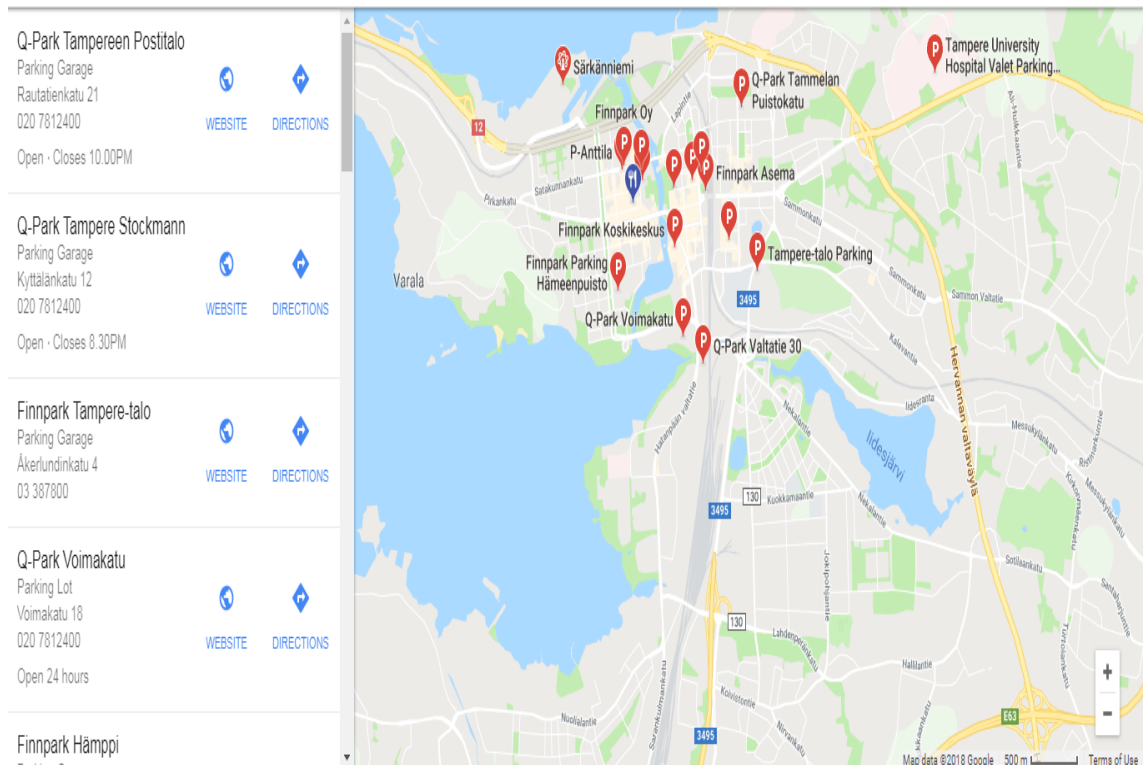


Figure 9: Shows Google search parking pin points in Tampere

In this project, Finnpark API which is an open source data serves as the source for the parking facilities in the Tampere region. The parking facilities basic information includes: location, name and address. The data content in the API uses DATEX11 standard and the data format is UML.

4.1 Data Contents

The data contents of the Finnpark DATEX11 parking facilities in Tampere.

Table 1: Shows the UML packages.

ParkingFacilityTablePublication	Main package for static parking information	
ParkingFacilityTableStatusPublication	Main package for dynamic parking information	
AreaExtension	Support package	
ParkingTariffs	Support package	
PeriodExtension	Support package	
SpecificDataTypeExtension	Support package	
ParkingExtensionEnumerations	Support package	

4.1.1 Parking facilities basic information.

The parking facilities basic information includes the location, name and addresses of the displayed available parking spaces in real time. The format is in XML (Rouse, 2018).

```

    <parkingFacilityRecordVersionTime>2015-06-16T16:00:00Z</parkingFacilityRecordVersionTime>
  <entranceLocation xsi:type="Point">
    <pointByCoordinates>
      <pointCoordinates>
        <latitude>61.505002</latitude>
        <longitude>23.817814</longitude>
      </pointCoordinates>
    </pointByCoordinates>
  </entranceLocation>
  <carParkDetails>
    <tag>Address</tag>
    <value>
      <values>
        <value lang="fi">Poikkitie, 33520, Tampere</value>
      </values>
    </value>
  </carParkDetails>
</parkingFacility>

```

Figure 10: Shows the parking information in XML format.

4.1.2 Parking facility table publication

This table gives a quick view of the header information, class and the mandatory recorded for all publications.

Table 2: DATEXII Parking Extension: parkingFacilityTablePublication

HeaderInformation	Class	Mandatory record for all publications
[0-1] areaOfInterest	Enum	<i>The extent of the geographic area to which the related information should be distributed.</i> Values; continentWide, national, neighbouringCountries, notSpecified and regional
confidentiality	Enum	<i>The extent to which the related information may be circulated, according to the recipient type. Recipients must comply with this confidentiality statement.</i> Values; internalUse, noRestrictions, restrictedToAuthorities, restrictedToAuthoritiesAndTrafficOperators, restrictedToAuthoritiesTrafficOperatorsAndPublishers, restrictedToAuthoritiesTrafficOperatorsAndVms
informationStatus	Enum	Status of parking extension information. Values; real, securityExercise, technicalExercise and test.
[0-1] urgency	Enum	<i>This indicates the urgency with which a message recipient or Client should distribute the enclosed</i>

4.1.3 Parking facility status information

The status information of the facilities shows if the facility is open, closed, spaces available, almost full, full at entrance, full, status unknown, normal parking conditions if suspended or a special parking condition is required.

Table 3:DATEXII Parking Extension: parkingFacilityTableStatusPublication

HeaderInformation	Class	Mandatory record for all publications
[0-1] areaOfInterest	Enum	See details from ParkingFacilityTablePublication
confidentiality	Enum	See details from ParkingFacilityTablePublication
informationStatus	Enum	See details from ParkingFacilityTablePublication
[0-1] urgency	Enum	See details from ParkingFacilityTablePublication
[0-n] ParkingAreaStatus	Class	Status may optionally divided into parking areas
[0-1] parkingAreaOccupancy	Percentage	<i>The percentage value of total parking spaces occupied.</i>
[0-1] parkingAreaOccupancyTrend	Enum	<i>The trend of the occupancy of the parking facility table.</i> Values; decreasing, increasing, stable
parkingAreaReference	Reference	<i>A reference to a versioned table which represents the collection of parking facilities</i>
[0-1] parkingAreaTotalNumberOfVacantParkingSpaces	Integer	<i>The total number of vacant parking spaces available in the specified parking facilityTable</i>
[0-1] totalParkingCapacityLongTermOverride	Integer	<i>Possibility to override the static value totalParkingCapacity</i>
[0-1] totalParkingCapacityOverride	Integer	<i>Possibility to override the static value totalParkingCapacity</i>
[0-1] totalParkingCapacityShortTermOverride	Integer	<i>Possibility to override the static value totalParkingCapacity</i>

[0-n] ParkingFacilityStatus	Class	This class may be either subclass for ParkingStatusPublication or ParkingArea
-----------------------------	-------	---

Note: see appendix for more tables

5 SYSTEM DEVELOPMENT AND TESTING

5.1 System Development

This section provides the system development environment setup instructions.

Angularjs requires some other third party libraries to be installed. Therefore before starting to setup the environment, Node.js must first be installed.

Node.js is a JavaScript runtime built on chrome's V8 JavaScript engine.

Node.js is also used for developing desktop applications and for deploying tools that make developing web sites simpler. For example, by installing Node.js on your desktop machine, you can quickly convert CoffeeScript to JavaScript, SASS to CSS, and shrink the size of your HTML, JavaScript and graphic files. Using NPM a tool that makes installing and managing Node modules, it's quite easy to add many useful tools to your web development toolkit.

First download the windows or mac installer form Node.js website and run the installer. The system must be restarted after installing Node.js before it can run.

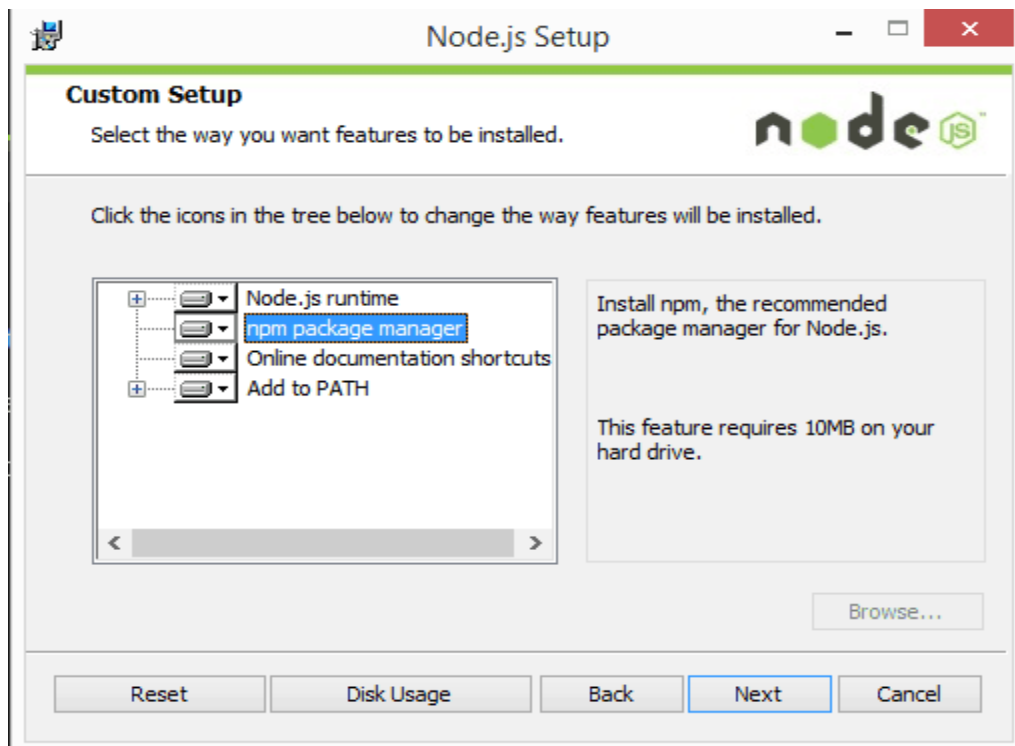


Figure 11: Node.js setup for windows.

After installing Node and NPM, the next step is to install angular.js from the Visual studio code editor with the command `npm install -g @angular/cli`. Note, any code editor can be used here. The next step is to create the project with a simple command line; `ng new Parkheretest`.

```

488 10492 silly decomposeActions install q@1.4.1cd
489 10719 silly decomposeActions refresh-package-json remove-trailing-separator@1.1.0

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
CREATE Parkheretest/src/app/app.component.ts (207 bytes)
CREATE Parkheretest/src/app/app.component.css (0 bytes)
CREATE Parkheretest/e2e/protractor.conf.js (752 bytes)
CREATE Parkheretest/e2e/src/app.e2e-spec.ts (299 bytes)
CREATE Parkheretest/e2e/src/app.po.ts (288 bytes)
CREATE Parkheretest/e2e/tsconfig.e2e.json (213 bytes)

> node-sass@4.9.4 install C:\Parkheretest\node_modules\node-sass
> node scripts/install.js

Downloading binary from https://github.com/sass/node-sass/releases/download/v4.9.4/win32-x64-57_binding.node
Download complete
Binary saved to C:\Parkheretest\node_modules\node-sass\vendor\win32-x64-57\binding.node
Caching binary to C:\Users\Techie24\AppData\Roaming\npm-cache\node-sass\4.9.4\win32-x64-57_binding.node

> node-sass@4.9.4 postinstall C:\Parkheretest\node_modules\node-sass
> node scripts/build.js

Binary found at C:\Parkheretest\node_modules\node-sass\vendor\win32-x64-57\binding.node
Testing binary
Binary is fine

> @angular/cli@6.0.8 postinstall C:\Parkheretest\node_modules\@angular\cli
> node .\bin/ng-update-message.js

npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@1.2.4 (node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.4: wanted {"os":"darwin","arch":"any"} (current: {"os":"win32","arch":"x64"})

added 1081 packages in 92.485s
'git' is not recognized as an internal or external command,
operable program or batch file.

C:\>

```

Figure 12: Creating angular project, parkhere test

5.1.1 Adding bootstraps and JQuery

Install latest Bootstrap version and JQuery for Bootstrap JavaScript support to your project:

```
$ npm install --save bootstrap jquery
```

The `--save` option will make Bootstrap appear in the dependencies.

Then edit/add the bootstrap scripts needed in scripts in `.angular-cli.json` file:

```

"scripts": [
  "../node_modules/jquery/dist/jquery.js",
  "../node_modules/bootstrap/dist/js/bootstrap.js"
],

```

Add/Edit also the Bootstrap css file in styles in `.angular-cli.json` file:

```

"styles": [
  "../node_modules/bootstrap/dist/css/bootstrap.min.css",
  "styles.css"
],

```

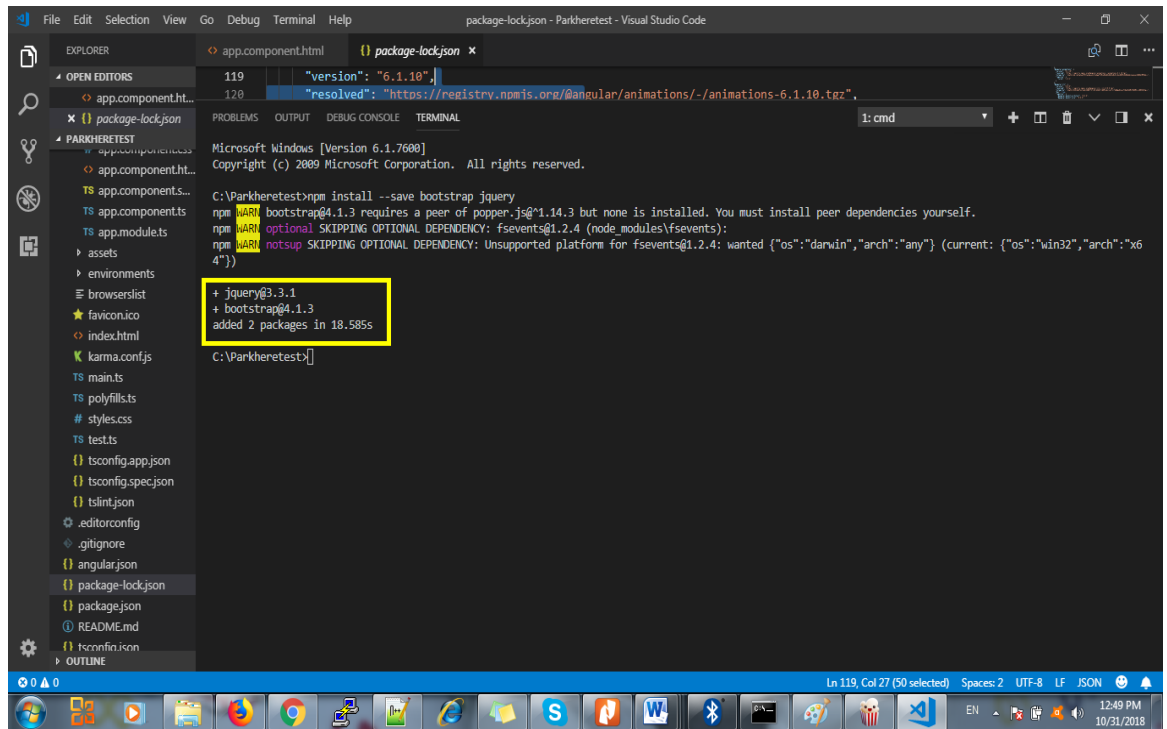


Figure 13: Adding bootstrap and JQuery to Angular cli

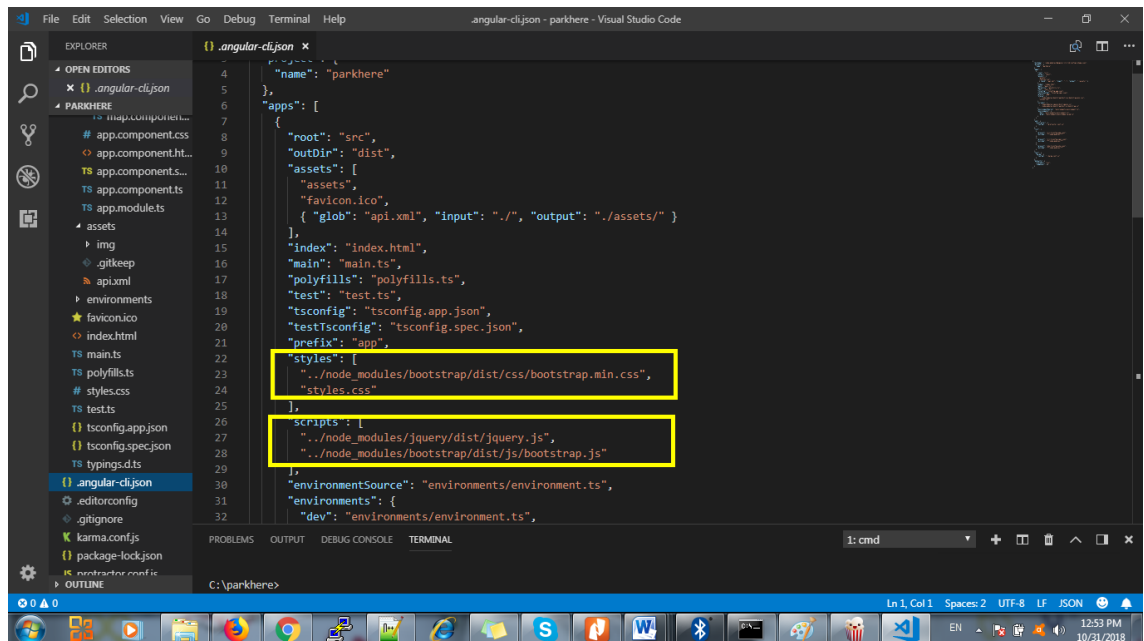
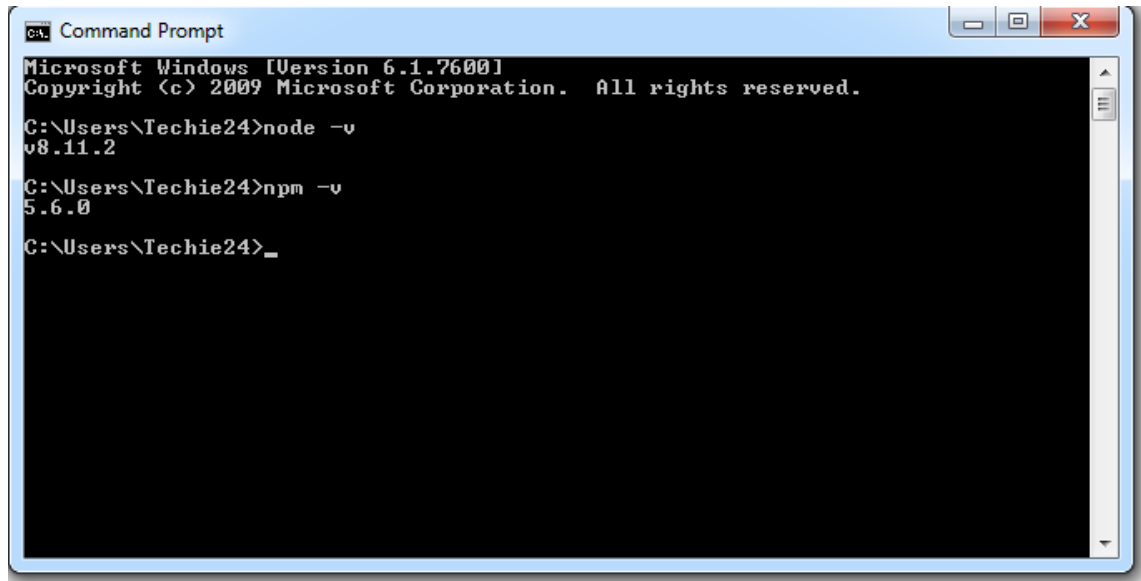


Figure 14: Bootstrap scripts needed in scripts and Bootstrap css file in styles in angular-cli.json file

5.2 Testing

To make sure the Node.js and NPM is installed is by running simple commands from the windows CMD command line and typing `node -v` and `npm -v`. the result will show on the command prompt displaying the different version of each packages. Figure 6 shows the expected result.

A screenshot of a Windows Command Prompt window. The title bar reads "c:\ Command Prompt". The window content shows the following text:

```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Techie24>node -v
v8.11.2

C:\Users\Techie24>npm -v
5.6.0

C:\Users\Techie24>_
```

Figure 15: Node and NPM installation testing.

6 USER INTERFACE IMPLEMENTATION.

The system user interface design is in three containers as already mentioned in the description in chapter 4. We will be discussing the user interface in this chapter. The user will be able to navigate the website easily because it simple and direct to the goal of this project. This project was design using the most available tools commonly used in web application.

6.1 Front Page

The front page of this GUI shows the information on how to get started in finding a parking facility in the Tampere region.

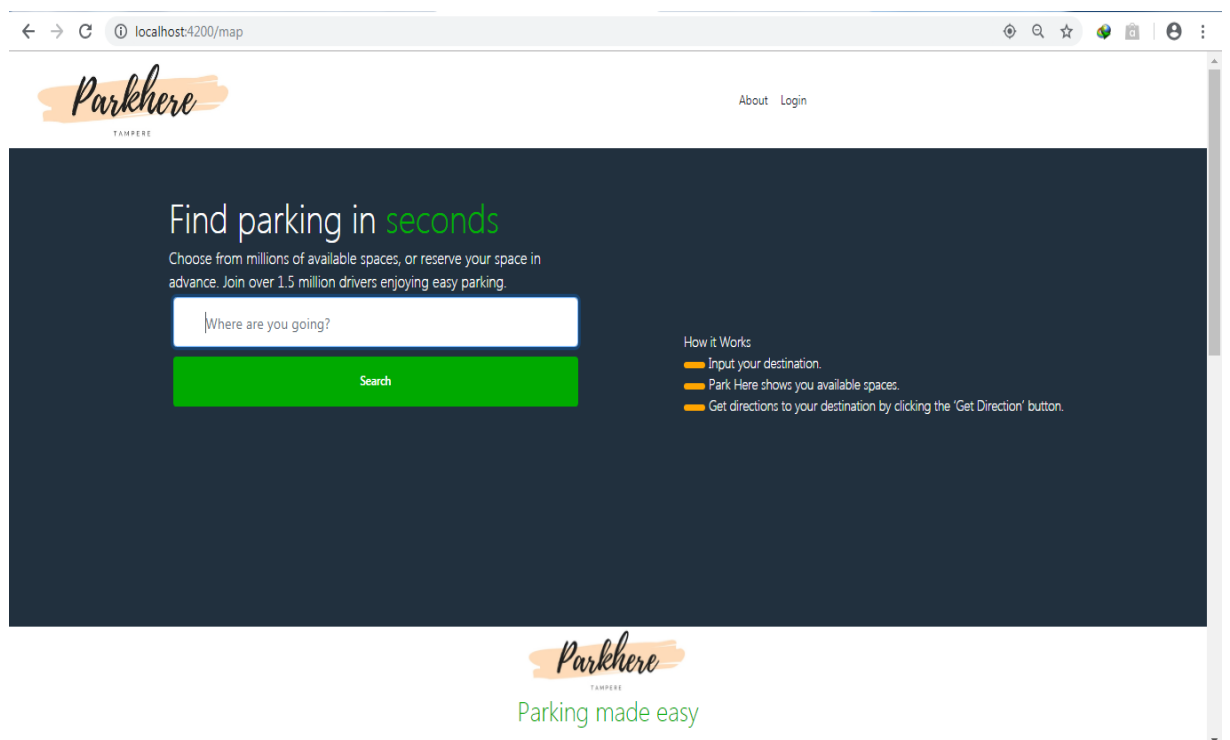


Figure 16: Parkhere interface

6.2 Input Destination

Parkhere web GUI is a very easy to navigate. The users simply need to input the destination address by typing the location on the search button and click search. A display of all available parking facilities near to the user destination will appear with the names. A click on the choice of the user chosen parking facilities will redirect you to the get direction page.

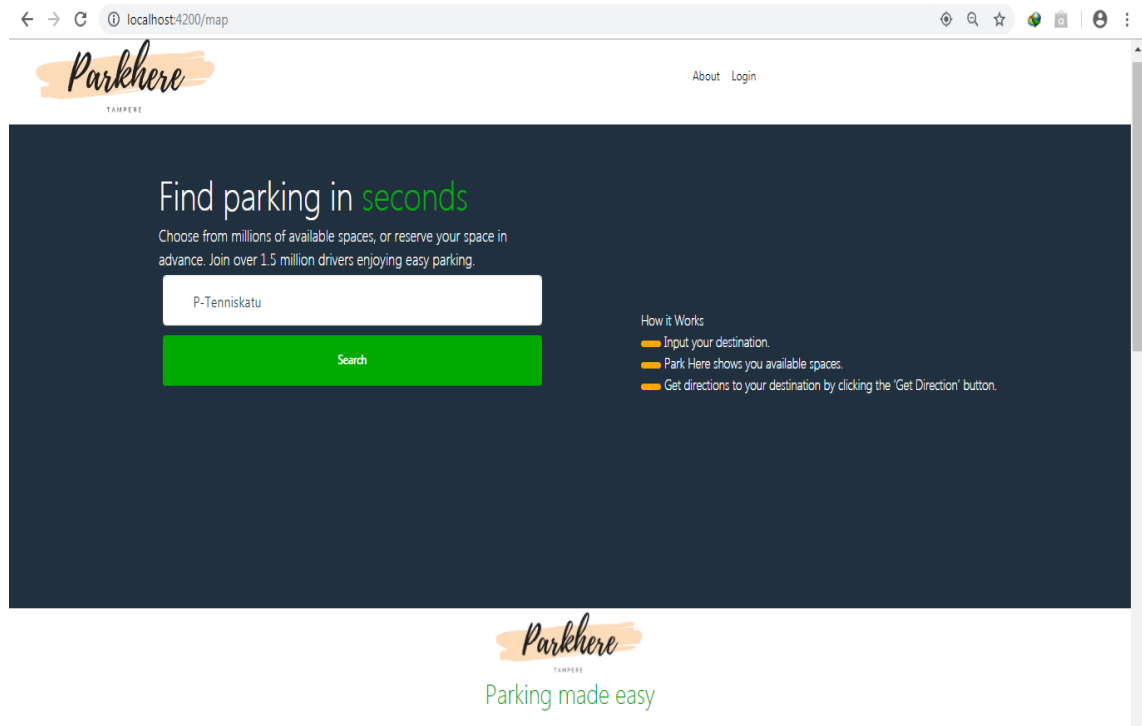


Figure 17: Parkhere inputting destination

6.2.1 Parking Facilities

A display of all available parking facilities closest to the destination address is displayed with Google map directions. The figure below shows available parking spaces around TOAS, Tampere.

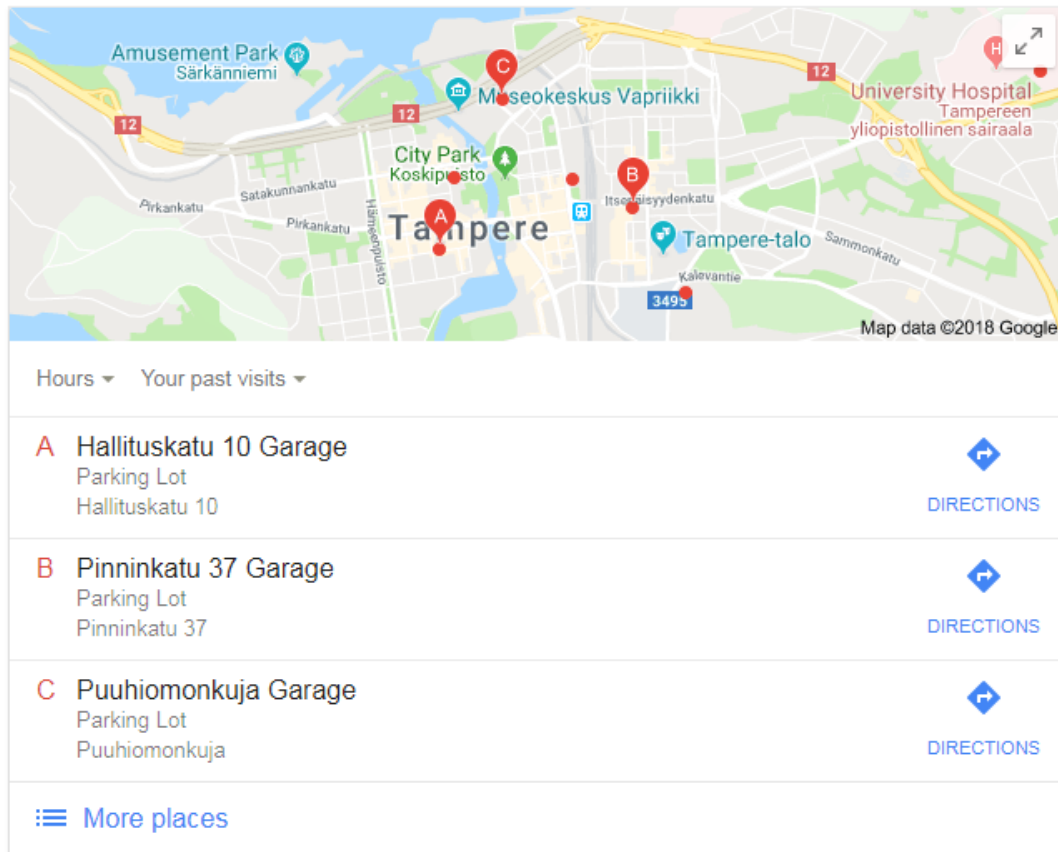


Figure 18: Google maps destination results

6.2.2 Availability

The information about the parking facility is displayed when you click on any of the parking facility. This details shows whether the parking facility is open, closed and the spaces available. It also shows if it is almost full and other information depending as provided by the parking facilities owners.

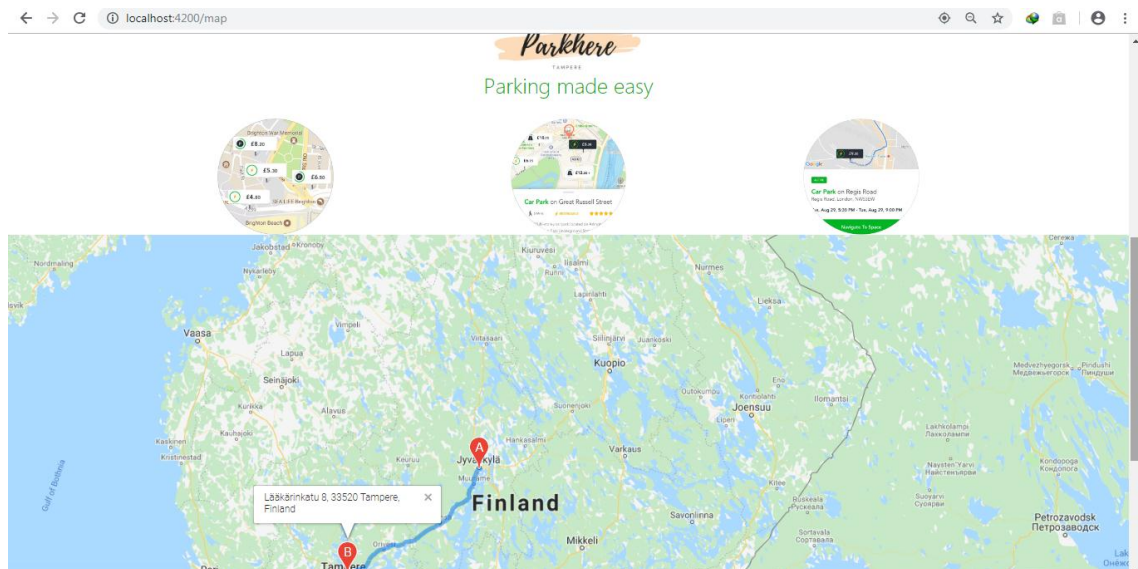


Figure 19: Parkhere results using google maps

6.3 Get Direction

The get direction navigates the user to Google map direction. To achieve this in this project we embedded the Google map for driving direction into the web site

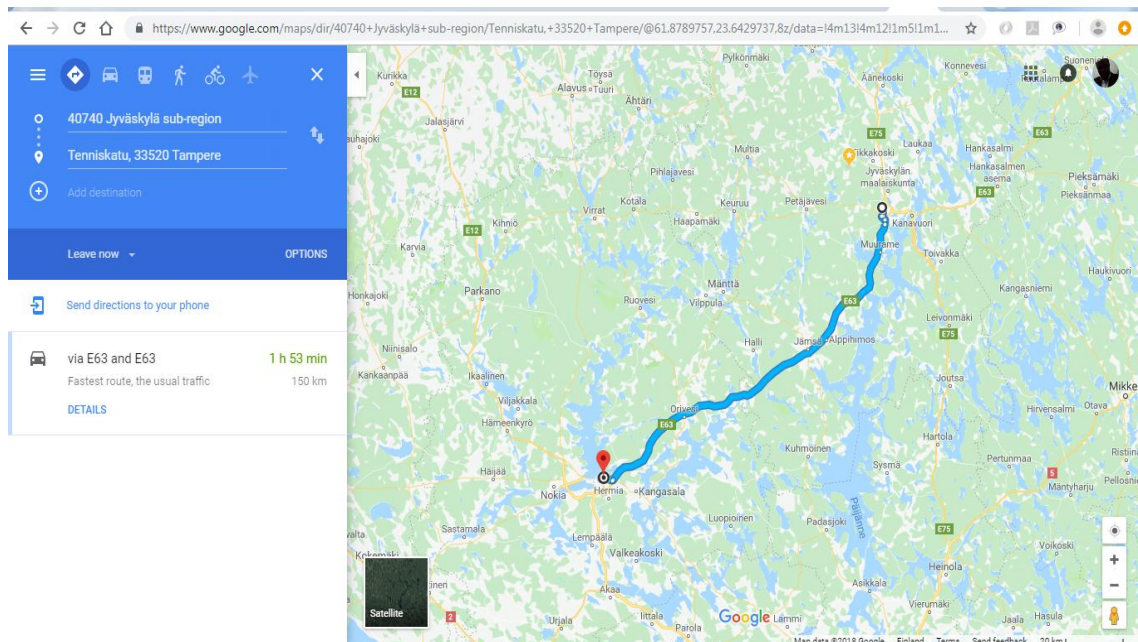


Figure 20: Integration of google maps in Parkhere

7 DISCUSSION

Parkhere web application was designed to ease cruising by vehicle drivers looking for parking facilities to park their cars. It is an addition to the existing parking system in Finland. However, this project was tailored to Tampere parking facilities providing real time information on the parking facilities in Tampere. I choose Tampere for this project because of the availability of already existing parking API provided by Finnpark.

The objective of this thesis was to design a system for finding parking facilities in Tampere by successfully implementing Finnpark open REST API for this project. Although, the process of implementing the API in my web application was quite challenging, I was able to navigate through it by creating an api.xml extension in my code in assets and copied the xml file from the parking data datex2 open data provided by Finnpark but there were still some other issues that require further investigation.

The system required extensive programming skills and this has taught me new programming best practices in current trend technologies like using bootstrap, jquery, angularjs, including embedded Google map for easy navigation. While the core coding technologies such as bootstrap, html and css, jquery, and angularjs served the operational framework in the backend, embedded google map presented an aesthetic and functional feature for users. This made the user interface accessible to users.

7.1 Future Development.

The initial plan for this project was specifically to point the drivers to the location of parking facilities but more information can be provided about the facilities which also Finnpark provided by reaching an agreement to obtain such data. Some of this further implementation would be;

- more detailed facility parameters (capacity, structure, payment, services, max. vehicle sizes, etc.)
- more detailed occupancy information like heating
- occupancy forecast
- similar data from street parking
- similar data from other cities in Finland
- a mobile application deployment.

8 REFERENCES

- AngularJS. (2018, November 7). *Scope of AngularJS*. Retrieved from AngularJS: <https://docs.angularjs.org/guide/scope>
- European-Union. (2018, November 2). *Tampere Parking DATEX2*. Retrieved from Transforming transport: <https://data.transformingtransport.eu/dataset/tampere-parking-datex2>
- Fluin, S. (2018, 10 27). *Angular: Branding Guidelines for AngularJS*. Retrieved from Angular JS: <https://blog.angularjs.org/2017/01/branding-guidelines-for-angular-and.html>
- Google. (2018, October 24). *Embedded Google Map*. Retrieved from Google Map: <https://www.embedgooglemap.net>
- hansOnUML. (2018, November 15). *Parking Lot*. Retrieved from GenMyModel: <https://repository.genmymodel.com/hansOnUML/ParkingLot>
- Rouse, M. (2018, November 24). *XML File Format*. Retrieved from Whatis.com: <https://whatis.techtarget.com/fileformat/XML-eXtensible-markup-language>
- Shoup, D. C. (2006). Cruising for parking. *Transport Policy*, 479–486.
- Super intelligence parking system (SIPS)*. (n.d.). Retrieved 11 9, 2018, from ER Diagram: <https://sites.google.com/site/sips5688/project/er-diagram>
- Tampere-city. (2018, April 19). *City of Tampere*. Retrieved October 22, 2018, from <https://www.tampere.fi/en/city-of-tampere/information-on-tampere.html>
- w3schools.com. (2018, October 13). *Introduction to XML*. Retrieved from W3Schools.com: https://www.w3schools.com/xml/xml_what.asp

9 APPENDICES

Appendix 1

Finnpark DATEX 11 parking extension table

https://www.dropbox.com/sh/ujs2hoim8z9py2y/AABIpIU25NWZktVokYbZMWk_a?dl=0

Appendix 2

Finnpark parking data API source code

<http://parkingdata.finnpark.fi:8080/Datex2/OpenData>

Appendix 3

Front end code

```
<div class="container">
  <nav class="navbar navbar-expand-lg" style="color: #17222c;">
    <div class="col-md-3 main_logo">
      <span class="span-logo">
        
      </span>
    </div>
    <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false"
aria-label="Toggle navigation">
      <span class="navbar-toggler-icon"></span>
    </button>

    <div class="collapse navbar-collapse" id="navbarSupportedContent">
      <ul class="navbar-nav c-header__menu">
        <li class="nav-item active">
          <a class="nav-link" href="#">About <span class="sr-only">(current)</span></a>
        </li>
        <li class="nav-item">
          <a class="nav-link" href="#">Login</a>
        </li>
      </ul>
    </div>
  </nav>
  <div class="row board">
    <div class="row">
      <div class="col-md-5">
        <div class="heading">Find parking in <span class="green-text">seconds</span></div>
      </div>
```

Choose from millions of available spaces, or reserve your space in advance. Join over 1.5 million drivers enjoying easy parking.

```

</div>
<div class="row">
  <div class="col-md-12">
    <input type="text" #name class="c-searchform__input form-control" placeholder="Where
are you going?"/>
  </div>
  <div class="col-md-12">
    <button class="c-searchform__submit c-searchform__input"
(click)="open(name.value)">Search</button>
  </div>
</div>
</div>
<div class="col-md-1"></div>
<div class="col-md-6 right-li">
  <div>How it Works</div>
  <div>
    <span class="li-custom"></span> Input your destination.
  </div>
  <div>
    <span class="li-custom"></span> Park Here shows you available spaces.
  </div>
  <div>
    <span class="li-custom"></span> Get directions to your destination by clicking the ‘Get Di-
rection’ button.
  </div>
</div>
</div>
</div>
<div class="row main-body">
  <div class="col-md-12 span-logo">
    
  </div>
  <div class="c-slider col-md-12">
    <h2>Parking made easy</h2>
  </div>
  <div class="col-md-4">
    <div>
      
    </div>
  </div>
  <div class="col-md-4">
    <div class="w">
      
    </div>
  </div>
  <div class="col-md-4">
    <div>
      
    </div>
  </div>

```

```

</div>

</div>
</div>

<app-map #map [name]="name.value"></app-map>

```

Appendix 4

Implementing map component code

```

import { Component, OnInit, NgModule, Input, OnChanges } from '@angular/core';
import { MouseEvent } from '@agm/core';
import { BrowserModule } from '@angular/platform-browser';
import { Http, Headers } from '@angular/http';
import { parseString } from 'xml2js';
import 'rxjs/add/operator/map';

@NgModule({
  imports: [
    BrowserModule,
  ]
})
@Component({
  selector: 'app-map',
  templateUrl: './map.component.html',
  styleUrls: ['./map.component.css']
})

export class MapComponent implements OnInit {
  constructor(private http: Http) { }

  @Input() name;
  markers: Marker[] = [{
    lat: 51.673858,
    lng: 7.815982,
    draggable: false
  }];
  lat = 51.673858;
  lng = 7.815982;
  zoom = 7;

  ngOnChanges(changes) {
    console.log(this.name);
    let text = "";
    this.http.get('./assets/api.xml').toPromise()

```

```

.then((res) => {
  return text = res.text();
})
.then((res) => {
  parseString(res, (err, result) => {
    const nst1 = result.d2LogicalModel.payloadPublication[0].genericPublicationExtension[0];
    const nst2 = nst1.parkingFacilityTablePublication[0].parkingFacilityTable[0];
    const parkingFacility = nst2.parkingFacility;
    for (let i = 0; i < parkingFacility.length; i++) {
      const log = parkingFacility[i].entranceLocation[0].pointByCoordinates[0].pointCoordinates[0].longitude[0];
      const lat = parkingFacility[i].entranceLocation[0].pointByCoordinates[0].pointCoordinates[0].latitude[0];
      const obj = {log, lat};
      this.markers.push({lat: lat, lng: log, draggable: true});
    }

  });
});
}

clickedMarker(label: string, index: number) {
  console.log(`clicked the marker: ${label || index}`);
}

mapClicked($event: MouseEvent) {
  this.markers.push({
    lat: $event.coords.lat,
    lng: $event.coords.lng,
    draggable: true
  });
}

ngOnInit() {
}
}

interface Marker {
  lat: number;
  lng: number;
  draggable: boolean;
}

```