Todo list

Table of contents

Chapter 1

Encoding Locations

1.1 Introduction

Encoding of locations has historically been of great importance, and is always being modernized. This chapter aims to explain which types of locations are important for this project, and which method of representing locations, that is universally interpretable, should be used, and how the encoded locations are to be stored in a database.

1.2 A Brief History Of Geographic Locations

A location is roughly described as a place or position. Throughout history, various navigational techniques and tools like the sextant, nautical chart and marinner's compass were used, measuring the altitude of the North Star to determine the latitude ϕ, in conjunction with a chronometer to determine the longitude λ of a location on the Earth's surface.

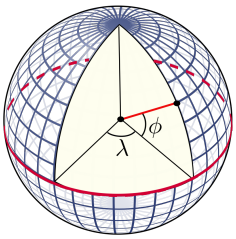

Fig. 1.1 A perspective view of the Earth showing how latitude and longitude are defined on a spherical model.

But the history of this encoding goes way back to its first to when it was first proposed in the 3rd century BC by Eratosthenes. He invented the discipline of geography, and was known for also being the first person to calculate the circumference of the Earth with remarkable

accuracy. Today, navigation relies on sattelites that are capable of providing information to determine a location with an accuracy of 9 meters. The precision of hybrid methods using cell towers and Wi-Fi Location Services enables precise tracking of modern devices.

kks32: lat lng at gps needs to be rewritten

Latitude and longitude are often referred to as GPS coordinates, as GPS is often used to calculate latitude and longitude by receiving position and time code sequences from at least four sattelites.

Addresses are another representation of a location used in navigation. Addresses are easier to communicate than a pair of GPS coordinates, but can be ambiguous, imprecise, inconsistent in format. Addresses commonly make use of Postal Code systems, which have reliably been assigned to geographical areas with the purpose of sorting mail. Although even today, there are countries that do not have a Postal Code system. A location being roughly described as a place or position, can be decomposed as an abstract term to describe physical or imaginary areas with varying radiusses and shapes. You could prepend 'the location of' to the following terms as an example: America, the birthplace of Sokrates, Wall Street, the center of the universe, the Laryngeal Nerve of the Giraffe, churches in the Netherlands. The final example presents the main challenge before mentioned of this project.

1.3 Requisites of Location Types

While setting up a backlog, a shared knowledge about the terminology used in the issues must be achieved. In the pregame document, the term "area" was defined as a collection of three or more coordinate pairs, or a collection of postal codes. A "point" was defined as one distinct coordinate pair or one distinct postal code. This way a location could either be an area or a point, with which all possibilities are covered. As stated in Appendix A, the definition of an area is precise, unambiguous and easy to use in compare in computer programs. A single point may match another single point if it's the exact same point. A point may be sitting on top of a line or is contained within an area. The only other option is the negation of these statements. Because use cases for lines will be non-existent, points and areas are the proper candidates for spatial queries.

A taxi company director wants to be able to set price or define discounts from or to a certain location. They would like to define prices based only on departure locations, or only on destination locations, or both. For example: 'to Schiphol, a trip should cost €10,-', or 'from van der Valk hotels, a trip should cost €5,-', or 'from van der Valk hotels to Schiphol, the km price should be €0,60'. In the current implementation, a record would be stored containing departure location, destination location and price for every combination, where

locations were defined as zip codes. Instead, it would make sense to be able to reuse locations after they have been defined once.

1.4 Literature Review

What 3 words, a multi-award winning global addressing system, bases 3m x 3m squares, covering the planet, on a combination of three words.

Geospatial

Postal code

kks32: See if other people solved locations

1.5 Database Prerequisites

The database must be capable of determining whether a virtual perimeter contains a set of coordinates, more specifically, it must adhere to The Open Geospatial Consortium (OGC) Simple Feature Access ISO 19125-1 [1] and ISO 19125-2 [2], including spatial data types, analysis functions, measurements and predicates for this requirement, or have some comparable implementation. The scenario presented in image 1.2 should be replicable.

kks32: Speak of database specific example requirements

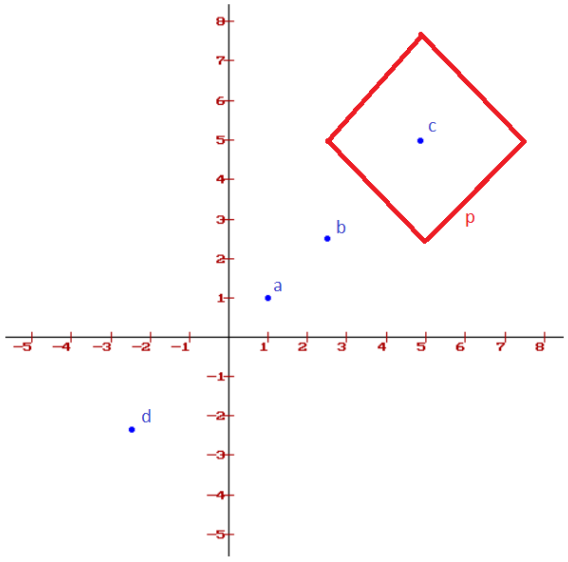

Fig. 1.2 Four Points, one Polygon p containing Point c.

1.5.1 OpenGIS Compatible databases

MYSQL's innate integrity is a good reason to opt for a full MYSQL database setup. MariaDB is a fork of MYSQL that performs better according to benchmarks, however they don't always translate to real life situations. It's easy to migrate from MYSQL to MariaDB, so choosing MYSQL at first could be preferable as an instance of MYSQL is already used at TaxiID. PostgreSQL offers a spatial database extender for that is OpenGIS compliant called PostGIS that adds support for geographic objects and location queries.

All spatial data types inherit properties such as type and spatial reference identifier (SRID). For rigorous documentation, both PostGIS documentation [3] and MYSQL documentation [4] could be consulted. When a generic geometry column, or point column is created, points can be inserted as shown in snippet 1.1 and 1.2

```
START TRANSACTION;
SET @a = ST_GeomFromText('POINT(1 1)');
INSERT INTO point (point) VALUES (@a);
SET @b = ST_GeomFromText('POINT(2.5 2.5)')
  ;
INSERT INTO point (point) VALUES (@b);
SET @c = ST_GeomFromText('POINT(5 5)');
INSERT INTO point (point) VALUES (@c);
SET @d = ST_GeomFromText('POINT(-2.5 -2.5)
  ');
INSERT INTO point (point) VALUES (@a);
# also insert @b, @c, and @d
COMMIT;
```

Listing 1.1 Insert four points

```
START TRANSACTION;
# First and last point must be the same
SET @a = PolygonFromText('POLYGON((2.5 5,5
   7.5,7.5 5,5 2.5,2.5 5))');
INSERT INTO polygon (polygon) VALUES (@a);
COMMIT;
```

Listing 1.2 Insert polygon

It is evident that c is contained in p. To determine which points are contained in p, the function as seen in Snippet

kks32: Add ref to snippet

can be used, which returns the point with coordinates $[5,5]$ as expected.

```
// All points contained in polygon
SELECT ST_ASTEXT(POINT)
FROM POINT
WHERE
ST_CONTAINS(
  (
    SELECT POLYGON
    FROM POLYGON
    WHERE id = 1
  ),
  POINT
)
```

Listing 1.3 Select points contained in polygon

```
// All polygons containing point
SELECT ST_ASTEXT(POLYGON)
FROM POLYGON, POINT
WHERE
  POINT.id = 3 AND ST_CONTAINS(
    POLYGON.polygon,
    POINT.point
  )
```

Listing 1.4 Select polygons containing point

1.5.2 OpenGIS Incompatible databases

MongoDB doesn't offer OpenGIS implementations but has geospatial query operators that may provide enough functionalities for current requirements

kks32: Add ref to Geospatial Query Operators — MongoDB Manual 3.6

. The argument for choosing one over the other depends on the vast differences between SQL and NoSQL, next to performance and extensiveness of geospatial features. The setup displayed in image

kks32: Add ref to image

is recreated in MongoDB using queries shown in snippet

kks32: Add ref to snippet

.

```
db.point.insertMany([
{ shape: { type: "Point", coordinates: [1, 1] } },
{ shape: { type: "Point", coordinates: [2.5, 2.5] } },
{ shape: { type: "Point", coordinates: [5, 5] } },
{ shape: { type: "Point", coordinates: [-2.5, -2.5] } },
])

db.polygon.insert({
shape: {
type: "Polygon",
coordinates: [ [ [2.5, 5], [5, 7.5], [7.5, 5], [5, 2.5], [2.5, 5] ] ]
}
})

db.point.createIndex({ 'shape': '2dsphere' })
db.polygon.createIndex({ 'shape': '2dsphere' })
```

```
// All points contained in polygon
var p = db.polygon.find({})

db.point.find({
shape: {
$geoWithin: {
$polygon: [
[2.5, 5],
[5, 7.5],
[7.5, 5],
[5, 2.5],
[2.5, 5]
]
}
}
})

// All polygons containing point
var p = db.point.findOne({ coordinates: [5, 5] })

db.polygon.find({
shape: {
$geoIntersects: {
```

Listing 1.5 Select points contained in polygon

```
24    $geometry: {
25    type: "Point",
26    coordinates: [5, 5]
27    }
28    }
29    }
30    })
```

Listing 1.6 Select points contained in polygon

Next to database solutions for this requirement, services exist that are capable of geofencing. Although these services may not be free, and the added dependencies restrict extensibility.

1.6 Performance and Clustering Trade-offs

Agarwal and Rajan state that NoSQL take advantage of cheap memory and processing power, thereby handling the four V's of big data more effectively, but lack the robustness over SQL databases

kks32: Add reference to Agarwal and Rajan

. The report dives deeper into spatial queries and concludes that their tests suggest that MongoDB performs better by an average factor of 10, which increases exponentially as the data size increases, but lack many spatial functions that OpenGIS supports. Although improvements have been made

kks32: Add reference to Geospatial Performance Improvements in MongoDB 3.2," MongoDB

after the cited paper Schmid et al. 2015

kks32: Add ref to Stephan Schmid Eszter Galicz

was published. The team argues that clustering is much easier in MongoDB, which may be important in the future when the company grows. As the required functionalities exist in both SQL and NoSQL, it is beneficial to opt for MongoDB for its performance and alignment with the teams experience. Although if robustness is desired, or extra GIS functionalities required, SQL should be taken into consideration.

In what way can locations be represented to be universally interpretable?

1. Which types of locations should be distinguished?
2. What are the main differences between postal systems used around the globe?
3. Can postal codes be abstracted to geospatial data while retaining the same usefulness in the system?
4. How can different types of locations be effectively stored in a database?

Chapter 2

System Architecture

2.1 Introduction

The price calculation system that is being made should integrate in the existing architecture seamlessly. This chapter aims to answer question number two, and its subquestions. First it is determined whether the frontend and backend that are to be developed should be integrated in existing projects, or should be made in separate projects altogether. If the backend is created as a separate project, authentication and authorization are directly affected by these decisions. Separation implies more complex identity management, or less separation of concern. Then, the database should be capable of storing geometry, and accept queries to determine which polygons contain a set of points, and which points are contained within a polygon.

2.2 Architectural Patterns

The current system architecture consists of three public API's and eight private API's that connect to four databases, as can be seen in 2.1 The core API is the interface to which mobile applications make requests. It is possible to integrate the pricing system as a module in the existing project. This simplifies authentication and authorization, because it already exists is that project. Another option is to implement the system as a microservice, which is infamously known as a service-oriented architecture (SOA). A microservice architecture is an architectural style that focuses on loosely-coupled services, improving scaling and continuous development of complex applications. Each microservice is responsible for managing and containing state to enable users who would like to use the system to be authenticated and authorized. Advantages of a microservice are the fact that a microservice is a self-contained and naturally modular structure, but authentication and authorization must be handled by the

microservice itself, unless state is shared amongst services, which would eliminate the reason to use a microservice at all. In the present architecture, different services implement different authentication methods, store different information about different users. Authorization is managed by sending extra headers for each crucial piece of information. For example: company information, application information and user information are all sent in separate headers.

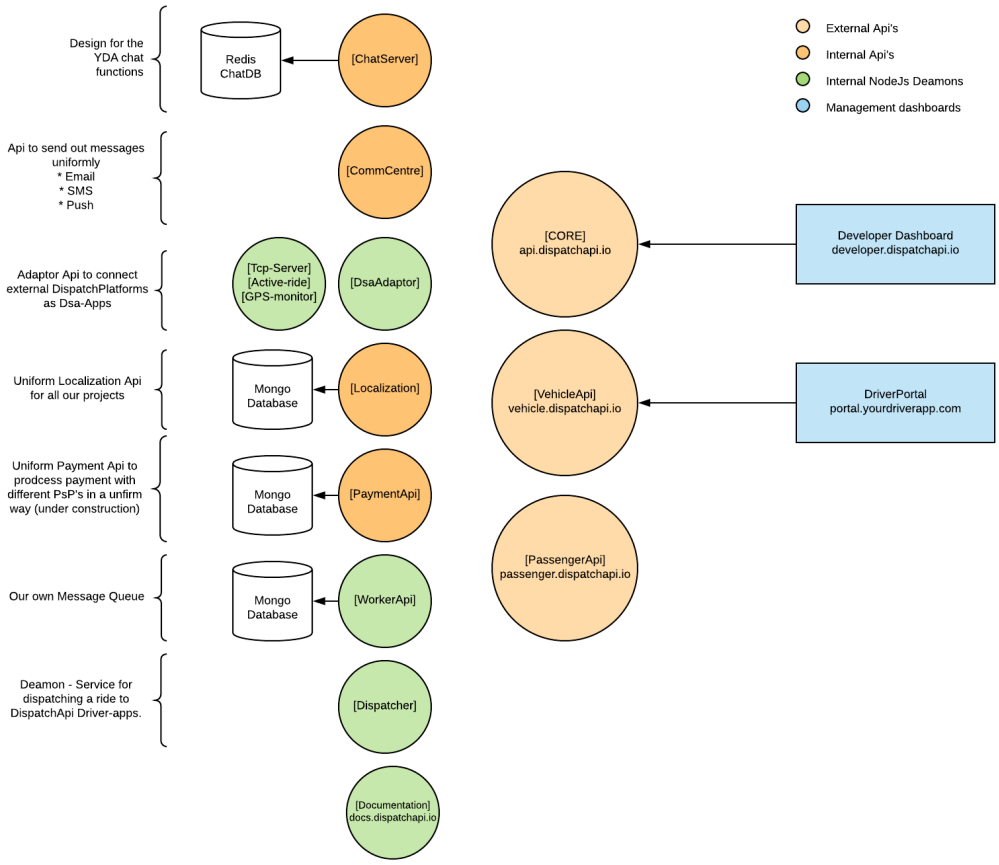

Fig. 2.1 Current System Architecture

When the amount of services that are added to the architecture increases, the mount of information that is no longer centralized increases as well.

2.3 Sharing Necessary Data

- User data
- Vehicle data
- etc..

2.4 Authentication and Authorization

Mobile applications should be able to make requests, just like the portals that are to be developed. But portal users make use of the microservice in a different way. Mobile apps merely request prices of products, based on the rules that group admins define through the portal. To make sure that only the portal users have the right to mutate their data, users have to be authenticated and authorized within the microservice. Identity management becomes a problem if data duplication is not desired. If a user makes a direct request to the microservice, the credentials have to be compared to user data in a database. To prevent duplication, the microservice could be connected to the database that is used by the core system. But this makes the microservice less decoupled, and directly contradicts the desire to separate concerns. Four examples demonstrate this problem:

- Example 1: The microservice authenticates and authorizes users all by itself, managing sessions and storing user data in its database.
- Example 2: The microservice connects to an exising databases to acquire the required information about the user.
- Example 3: The core system authenticates the user and provides a token that can be verified by the microservice, containing user identity.
- Example 4: A separate service is used for authentication and authorization so that the core system is not involved at all.

In the first example, the microservice seems to work independently, because it has knowledge about the users identity without making requests to adjecent systems, or connecting to external databases. But this is not true. If data about the user is mutated in the core system,

the microservice needs to be notified or synced. This greatly hinders scaling and makes it harder to keep data consistent. Example two solves the inconsistency part by connecting to the central database that holds user data, but contradicts the strive for encapsulation.

2.4.1 JSON Web Tokens

Example three entirely removes the database connection to any user data. This is possible when a JSON Web Token (JWT) is used. A JWT may be signed with a cryptographic algorithm or even a public/private key pair using RSA. After the user enters valid credentials, the core system validates the credentials by comparing them with user data in the database. The core system signs a token that with a secret that is known by the microservice. The token consists of three parts, separated by a fullstop. The first part (header) of the token contains information about the hashing algorithm that was used to encrypt the payload. This part is Base64Url encoded. The payload itself contains information stored in JSON format:

kks32: Show diagram with hierarchy of companies and apps

kks32: Make

```
{
        "companyId": "59ea0846f1fea03858e16311",
        "daAppInstallId": "599d39b67c4cae5f11475e93",
        "iat": 1521729818,
        "exp": 1521816218,
        "aud": "tps.dispatchapi.io",
        "iss": "api.dispatchapi.io",
        "sub": "getPrices"
}
```

The identity of the user is stored in the payload that can only be revealed by whoever holds the secret with which it was signed. Then the message can be verified using the third part of the token, which is the signature. This verification step prevents tampering with the payload.

2.4.2 oAuth 2.0

Example four delegates managing user identity to a separate authentication service that, similar to the pricing microservice, has its own single task of authenticating users. OAuth

2.0 is a protocol that has been designed to allow third-party apps to grant access to an HTTP service on behalf of the resource owner. This behaviour could be utilized to allow users to make use of services within the architecture, controlled by a single service, stored in a single token. A proposal was made in the Pregame document to combine oAuth with JWT and an API Gateway to introduce an automated authentication flow with a single token, instead of sending multiple headers, see Appendix A.

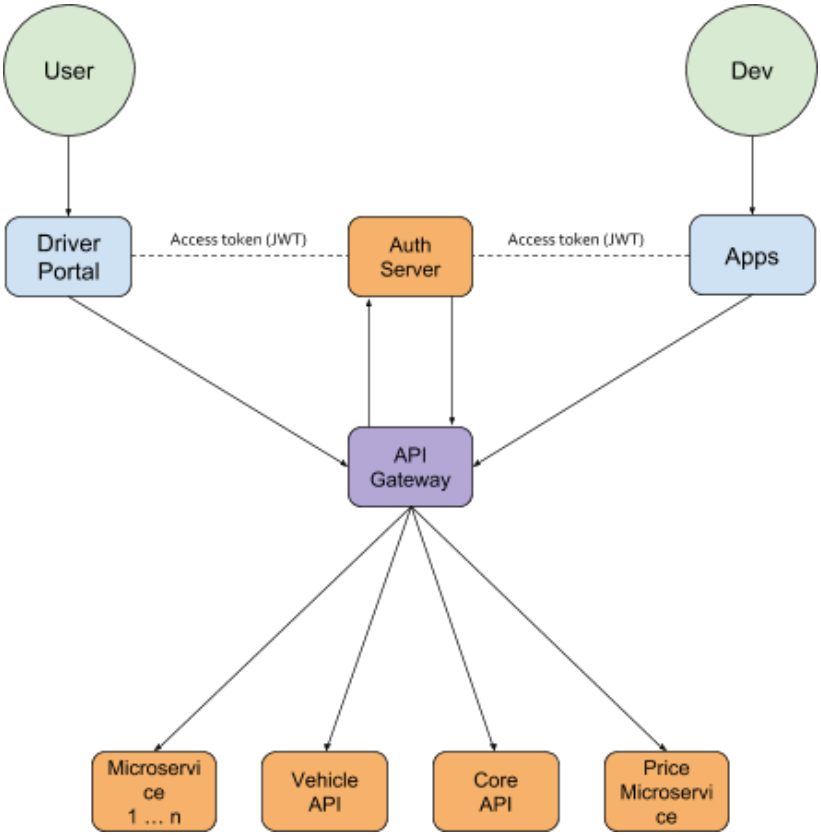

Fig. 2.2 oAuth, API Gateaway, using JWT

2.4.3 API Gateway

Another common structure that allows services to be used by external agents is the API Gateway. It allows for a central middleware in which authentication and authorization is handled, where the microservices are shielded from public access, and all communication is established through the API Gateway [4].

Next to authentication, the gateway could optimize the endpoints so that no multiple requests are needed from external agents to gather different types of resources. These calls could be made internally to the microservices behind the gateway. This also opens the possibility the freely change the microservices without changing the public endpoints exposed by the gateway, and even offers slow or instant transitions to different versions of microservices.

The different proposals explain the improvements they may bring over some system. But the advice given is not tied to this project, instead to the entire Dispatch API. It's advised to have a constructive dialogue about the future of the company, and the way it's planning to scale. One could put a API Gateway in front of a monolithic app to help with transitioning to a microservice-oriented app.

2.5 Suitability of Methods and Technologies

kks32: Write chapter about methods and technologies

- Node JS, PHP, MongoDB, MySQL, Microservice, Loopback, Graph QL,
- Slides and proposals
- Mocha, Buddy-Works, Circle CI, Typescript, Chai, Functional Programming

Chapter 3

TPS

3.1 Introduction

kks32: Write entire TPS chapter

What does a price calculation look like? (How should they be calculated)

1. Which parts of the system have an impact on the calculation result?
2. Query, Aggregate, ()
3. Proposals

A tier price system, that calculates fixed prices based cascading thresholds, and a dynamic pricing system that calculates prices per distance unit and minute is a very specific problem that must be split up into sizable categories to increase the chances of finding relevant solutions. The term 'distance unit' is used on purpose, as distances are measured using different metrics in various countries. Pricing rules should be constrained by time frames, making rules available only for some hours a day, or only on christmas for example. Rules should be specifyable per product as different vehicle types have different prices, but are included in the same pricing rules. Discounts may be calculated with the trip price, and VAT should be displayed in the price breakdown. Some additional requirements to the system may be added in later phases, as Scrum is used to manage work iterations (this fact is covered later in this chapter). The system should be accessible to other systems, meaning that applications that currently rely on the old system should be able to migrate to the new system. As the old system shouldn't be used for new applications, as it was not designed for this use case. The system should have a single responsibility, and should be autonomous in that regard.

Chapter 4

Proposed Portal Solution

4.1 Introduction

This chapter covers the actual implementation plan of connecting the pricing system with the portal frontend. How the system should behave under different circumstances, how the user is able to interact with the system. The YTA-portal should integrate the frontend that allows taxi company users to modify their pricing rules. This chapter aims to answer question number three, which aims to

4.2 Required Views

4.3 Methods and Techniques

4.4 Proposal Pricing Rules View

4.5

kks32: This is not researched yet, as it's covered in later sprints

How can the task of defining rules be as insightful as possible to the user?

1. Which views should exist, does a logical hierarchy exist among views?
2. How should locations be defined and managed by the user?
3. How should timeframes be handled in the interface?

Chapter 5

Realization

5.1 Introduction

During the second phase, issues from the backlog were implemented in an iterative SCRUM process. In this chapter, the final realization of the project is evaluated. Findings and observations by considering the assumptions and limitations are discussed. During development, two main applications were written. The price calculation system, and the portal that enables users to manage pricing rules in the price calculation system.

5.2 Methods and Techniques

In the first sprint, a project was set up in NodeJS using Typescript. The existing projects were built using Javascript, but Typescript is a much safer language, preventing bugs because the compiler can catch errors early on, warning programmers instead before the application crashes. Types also document code, expressing the intention of the programmer.

5.3 Sprint 1 - Dynamic Price Calculations

A basic dynamic price calculation system was implemented in the first sprint, aiming to deliver a first version of the system, including fake data generators, validation of models, a single service to determine the distance and duration of a ride, rules that contain pricing information, a calculation that produces the total price of a ride using a companies rules, a formatter that produces an expected response, and tests for all of the functionalities.

5.4 Sprint 2 - Authentication and Authorization

Company pricing rules can be used by applications so that each application uses a subset of the pricing rules. For this reason, TPS requires two identifiers to make a price calculation using rules for a particular company application: a companyId and a daAppInstallId. JSON Web Tokens that are signed by the core system hold the identifiers in the payload, so that the TPS can use the identifiers after decrypting the token. Companies have one country by default, which determines the currency and VAT percentage. In the breakdown, the VAT percentage is calculated from the actual price, as VAT is included. Discounts are part of the breakdown, being a percentage of the route price, or a fixed price. On top of that, it is possible that a company application uses rules that are related to a debtor, instead of its own subset of rules. Finally, the project is deployed to a staging environment so that the system could be used by the applications in the staging environment.

5.5 Sprint 3 - Setting up the Portal

At this point the system is fully operational, but company and daAppInstall information has to be inserted in the database manually. An endpoint is made that inserts a full company setup into the database so that prices can be calculated with five products by default. No wireframes were made beforehand, making it a task for the current sprint being executed while setting up the portal project. Angular in conjunction with Covalents UI platform is used to make the user interface, consisting of an overview and detail page for products and pricing rules. The pricing rules overview shows pricing information for each product that a company has. This is automatically maintained whenever a product is added or deleted. Furthermore, threshold rules can be added or deleted for distances and durations, making this particular view very complex. This final task was not finished in time.

5.6 Sprint 4 - Expanding the Portal

- thresholds continued - processing feedback - pricing rules overview, rules must be draggable to prioritize - priority must be maintained distinctly - app installations must be displayed - pricing rules and special rates must be enabled for specific app installs - allow on meter calculations - internationalization - timeframes -

5.7 Result

Chapter 6

Conclusion

Chapter 7

Recommendations

References

[1] “Simple feature access - part 1: Common architecture | ogc,” 2018.

[2] “Simple feature access - part 2: Sql option | ogc,” 2018.

[3] “Postgis 2.4.5dev manual,” 2018.

[4] “Mysql 5.7 reference manual - geometry class,” 2018.

List of figures

List of tables

Appendix A

Pregame

Pregame

TeXLive package - full version

1. Download the TeXLive ISO (2.2GB) from
   https://www.tug.org/texlive/
2. Download WinCDEmu (if you don't have a virtual drive) from
   http://wincdemu.sysprogs.org/download/
3. To install Windows CD Emulator follow the instructions at
   http://wincdemu.sysprogs.org/tutorials/install/
4. Right click the iso and mount it using the WinCDEmu as shown in
   http://wincdemu.sysprogs.org/tutorials/mount/
5. Open your virtual drive and run setup.pl

or

Basic MikTeX - TEX distribution

1. Download Basic-MiKTEX(32bit or 64bit) from
   http://miktex.org/download
2. Run the installer
3. To add a new package go to Start » All Programs » MikTex » Maintenance (Admin)
   and choose Package Manager

4. Select or search for packages to install

TexStudio - TEX editor

1. Download TexStudio from
   http://texstudio.sourceforge.net/#downloads
2. Run the installer

Mac OS X

MacTeX - TEX distribution

1. Download the file from
   https://www.tug.org/mactex/
2. Extract and double click to run the installer. It does the entire configuration, sit back and relax.

TexStudio - TEX editor

1. Download TexStudio from
   http://texstudio.sourceforge.net/#downloads
2. Extract and Start

Unix/Linux

TeXLive - TEX distribution

Getting the distribution:

1. TexLive can be downloaded from
   http://www.tug.org/texlive/acquire-netinstall.html.
2. TexLive is provided by most operating system you can use (rpm,apt-get or yum) to get TexLive distributions

Installation

1. Mount the ISO file in the mnt directory

   ```
   mount -t iso9660 -o ro,loop,noauto /your/texlive####.iso /mnt
   ```

2. Install wget on your OS (use rpm, apt-get or yum install)

3. Run the installer script install-tl.

   ```
   cd /your/download/directory
   ./install-tl
   ```

4. Enter command 'i' for installation

5. Post-Installation configuration:
   http://www.tug.org/texlive/doc/texlive-en/texlive-en.html#x1-320003.4.1

6. Set the path for the directory of TexLive binaries in your .bashrc file

For 32bit OS

For Bourne-compatible shells such as bash, and using Intel x86 GNU/Linux and a default directory setup as an example, the file to edit might be

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/i386-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

For 64bit OS

```
edit $~/.bashrc file and add following lines
PATH=/usr/local/texlive/2011/bin/x86_64-linux:$PATH;
export PATH
MANPATH=/usr/local/texlive/2011/texmf/doc/man:$MANPATH;
export MANPATH
```

```
INFOPATH=/usr/local/texlive/2011/texmf/doc/info:$INFOPATH;
export INFOPATH
```

Fedora/RedHat/CentOS:

```
sudo yum install texlive
sudo yum install psutils
```

SUSE:

```
sudo zypper install texlive
```

Debian/Ubuntu:

```
sudo apt-get install texlive texlive-latex-extra
sudo apt-get install psutils
```

Appendix B

Installing the CUED class file

LaTeX.cls files can be accessed system-wide when they are placed in the <texmf>/tex/latex directory, where <texmf> is the root directory of the user's TeXinstallation. On systems that have a local texmf tree (<texmflocal>), which may be named "texmf-local" or "localtexmf", it may be advisable to install packages in <texmflocal>, rather than <texmf> as the contents of the former, unlike that of the latter, are preserved after the LaTeXsystem is reinstalled and/or upgraded.

It is recommended that the user create a subdirectory <texmf>/tex/latex/CUED for all CUED related LaTeXclass and package files. On some LaTeXsystems, the directory look-up tables will need to be refreshed after making additions or deletions to the system files. For TeXLive systems this is accomplished via executing "texhash" as root. MIKTeXusers can run "initexmf -u" to accomplish the same thing.

Users not willing or able to install the files system-wide can install them in their personal directories, but will then have to provide the path (full or relative) in addition to the filename when referring to them in LaTeX.