

A rule-based geospatial reasoning system for trip price calculations



Stefan Schenk

Supervisor: Willem Brouwer

Advisor: Mewis Koeman

Department of Software Engineering
Amsterdam University of Applied Sciences

This dissertation is submitted for the degree of
Bachelor Software Engineering

May 2018

Todo list

■ Abstract, will be the last step	iii
■ Rewrite section: why not PHP, MySQL, GraphQL??	27
■ Expand upon choices: CircleCI, Buddy-Works, Typescript, and what is functional programming?	28
■ Expand on interfaces, static strong types, type hinting, OOP en FP mixen, OOP voor grote structuur, FP voor solide operaties, SOLID, Gang of Four, Loose coupling high cohesion, Async, Strategy pattern,	32
■ Working on locations	33
■ Changes a lot, needs final version	37
■ Conclusion for chapter 4	42
■ Skip Portal Solution Chapter?	43
■ Sprint 6 - 8	47
■ Final Result (realization)	47
■ Final Conclusion	49

Table of contents

1	Introduction	1
1.1	Context	1
1.2	Problem Definition	2
1.3	Assignment	3
1.4	Research	3
1.4.1	Questions	4
1.5	Process	5
2	Encoding Locations	7
2.1	Introduction	7
2.2	A Brief History Of Geographic Locations	7
2.3	Requisite Location Types	8
2.3.1	The Point	8
2.3.2	The Area	9
2.3.3	Postal Codes, Addresses, and Polygons	10
2.3.4	Requirements for Location Matching	11
2.4	Literature Review	11
2.5	Database Prerequisites	13
2.5.1	OpenGIS Compatible databases	13
2.5.2	OpenGIS Incompatible databases	15
2.6	Overlapping Locations	17
2.7	Performance and Clustering Trade-offs	17
2.8	Locations Conclusion	17
3	System Architecture	19
3.1	Introduction	19
3.2	Architectural Patterns	19
3.2.1	Monoliths	20

3.2.2	Microservices	21
3.2.3	Frontend and Backend	21
3.3	Information Dependencies	22
3.4	Authentication and Authorization	23
3.4.1	OAuth 2.0	24
3.4.2	JSON Web Tokens	25
3.4.3	API Gateway	26
3.5	Technologies	27
3.6	Methods and Techniques	28
3.7	Conclusion	28
4	Trip Price Calculation System	31
4.1	Introduction	31
4.2	The System	31
4.3	Locations	33
4.4	Timeframes	33
4.4.1	Conventional Approach	34
4.4.2	Bitmap	34
4.5	The Trip Price Calculation	35
4.5.1	Discounts	37
4.5.2	Rules	37
4.6	Price Calculation Types	38
4.6.1	Dynamic	38
4.6.2	Fixed	38
4.6.3	Meter	39
4.7	Threshold Calculations	39
4.8	Breakdown	40
4.9	Conclusion	42
5	Proposed Portal Solution	43
5.1	Introduction	43
5.2	Required Views	43
5.2.1	Entities	43
5.2.2	Hierarchy	43
6	Realization	45
6.1	Introduction	45

6.2	Methods and Techniques	45
6.3	Sprint 1 - Dynamic Price Calculations	45
6.4	Sprint 2 - Authentication and Authorization	46
6.5	Sprint 3 - Setting up the Portal	46
6.6	Sprint 4 - Expanding the Portal	46
6.7	Sprint 5 - Expanding the Portal	47
6.8	Sprint 6 - Locations	47
6.9	Result	47
7	Conclusion	49
8	Recommendations	51
8.1	Frontend	51
8.2	Backend	51
8.3	Functionalities	52
8.3.1	Authentication section Authorization	53
8.4	Database	53
8.4.1	section Interface	53
References		55
List of figures		57
List of tables		59
Appendix A	Pregame	61
Appendix B	Sprint Review and Proposal Slides	97
B.1	Sprint 1 - review	97
B.2	Sprint 2 - breakdown	109
B.3	Sprint 2 - authentication	121
B.4	Sprint 2 - review	130
B.5	Sprint 3 - review	146
B.6	Sprint 4 - review	155

Chapter 2

Encoding Locations

2.1 Introduction

Encoding of locations has historically been of great importance, and is always being modernized. This chapter explains the general definition of locations, which types of locations are important for this project, and how to represent these locations so that they are universally interpretable and do not rely on a postal code system.

2.2 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. The combination of coordinates is a distinct encoding of a location. This particular system is still commonly used today.

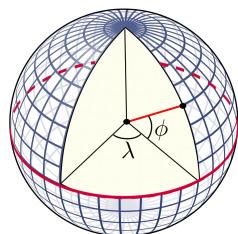


Fig. 2.1 A perspective view of the Earth showing how latitude and longitude are defined on a spherical model.

Today, navigation relies on satellites that are capable of providing information to determine a location with a precision of 9 meters. Hybrid methods using cell towers, Wi-Fi Location Services, and the new Galileo global navigation satellite system, provide tracking with a precision down to the meter range. These locations are ordinarily communicated using the same established latitude and longitude encoding. For a human being, it is not practical to exchange day-to-day locations as geographical coordinates. For that, addresses much more suitable, but can be ambiguous, imprecise, and 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. This forces the legacy system to support addresses for the fixed pricing functionality as well. In contrast to the geographic coordinate system, postal codes describe streets and areas of varying shapes and sizes. 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 radiuses 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 of this project, how to communicate the location of a collection with points or areas of differing shapes and sizes that may overlap?

2.3 Requisite Location Types

While setting up a backlog for a project, a shared knowledge about the terminology used in the issues must be achieved in order to collaborate effectively. Words or symbols do not have an absolute meaning, and ambiguity of abstract linguistic terms should be elucidated. In section 3.2.1 of Appendix A, an agreement was made on what the terms "area" and "point" meant. The MySQL documentation notes that "The term most commonly used is geometry, defined as a point or an aggregate of points representing anything in the world that has a location." in [2]. During the process of implementing TPS, the definitions of a location have been refined to represent a common and useful understanding.

2.3.1 The Point

A point is a unique place expressed as a distinct coordinate pair. An address in the legacy system could be translated to a point. For example, the address that is tied to Schiphol arrival is: Aankomstpassage, 1118 AX Schiphol Centrum. The point that encodes this location is (52.308891, 4.760900). This location is contained in the set of all possible points on Earth,

which could be expressed using set builder notation:

$$P = \{(\phi, \lambda) \in \mathbb{R}^2 \mid -90 < \phi < 90, -180 < \lambda < 180\}$$

$$(52.308891, 4.760900) \in P$$

A point itself can not be used to match whether another point is contained within it, because the probability of a match is infinitesimal. Only when decimals were disregarded to decrease the precision of a point it would be possible, in which case it would still not be useful in this application, because the imprecise point would be a square.

2.3.2 The Area

An area is a set of points with an infinite granularity. This definition allows for an area to have holes inside them, consist of other locations and contain other locations, and be infinitely precise. The most useful property of this area is to check whether a point is contained within the area, or which areas contain a given point. For this to be the case, the points must be packed together as it would form a shape. This definition, however conceptually valuable, will not be of much practical use. For example, P is an infinitely long set of coordinates, an area that represents the earth's surface. If ϕ ranged between 0 and 90, the set could describe all points located in the northern hemisphere, but would still be infinitely long. Checking whether a given point is contained by checking an infinite amount of real number pairs will take an infinite amount of time in the worst case scenario. An area can be described as a subset of all points:

$$a_1 \subseteq P$$

The set of all possible areas can be defined by the power set of P :

$$A = \mathcal{P}(P)$$

such that a subset of points, called an area, is in all possible areas A :

$$a_1 \in A$$

At the equator, 1 degree is 111320m, so 0.000001 degrees is around 11cm. Six decimal places will be sufficient for location matching for this application. But even when reducing coordinates to having six decimal places, it would be impractical. For this reason, it is more realistic to only describe the rough edges of an area using a polygon shape. Instead

of checking for a single point in a non-terminating iteration over all points in an area, a mathematical calculations could be used to check whether a unique point is contained within the polygon.

2.3.3 Postal Codes, Addresses, and Polygons

All postal codes that start with a ten describe the city of Amsterdam, the entire area of Amsterdam can be drawn as a big polygon containing all the postal codes that start with a ten.

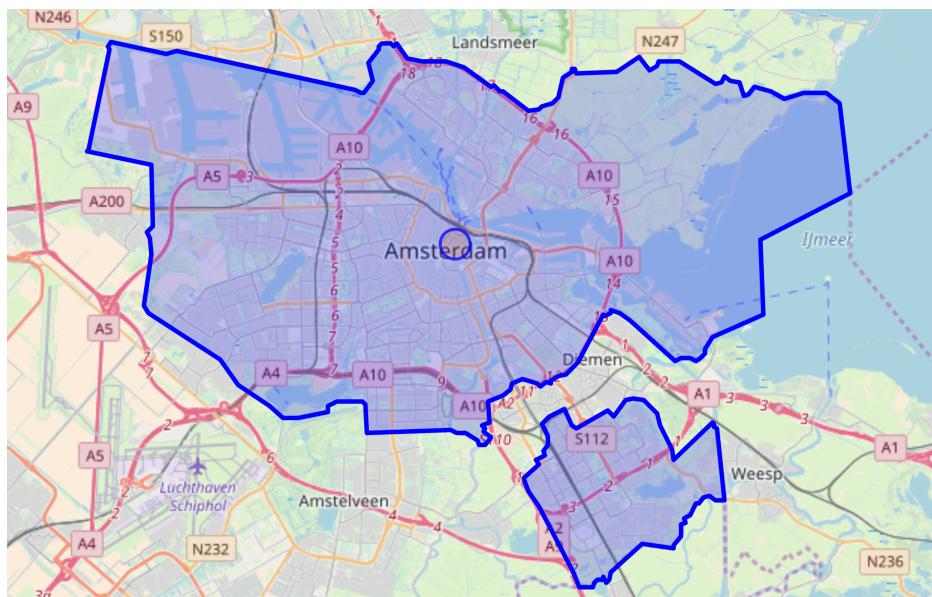


Fig. 2.2 Amsterdam - A single location comprised of multiple locations.

In reverse, this procedure would not work. If a polygon was drawn cutting Amsterdam in half diagonally, a single postal code pattern would never be flexible or precise enough to be able to describe the boundaries of the polygon. One big taxi company making use of taxiID's legacy system is located in the United Arab Emirates. This company would not be able to convert anything at all, because the United Arab Emirates does not have a postal code system to begin with.

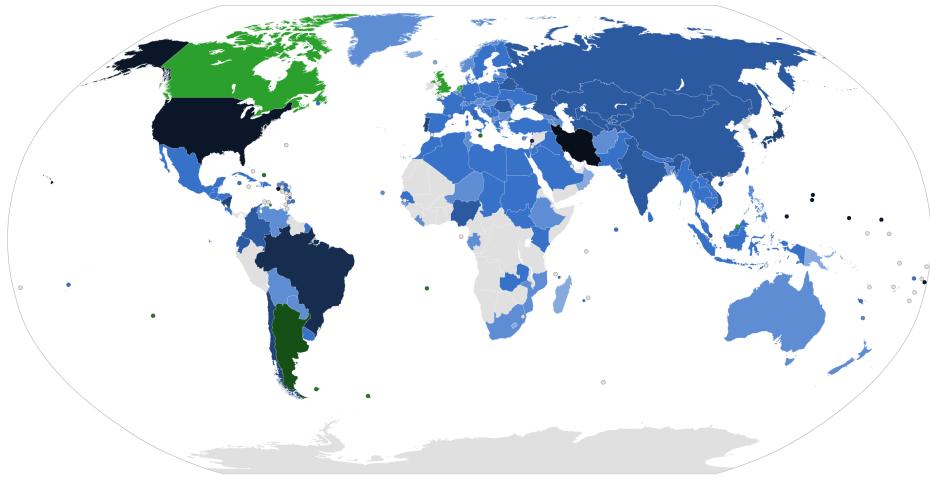


Fig. 2.3 Postal codes by country with amount of digits ranging from three digits (lightblue) to eight digits (darkgreen) and no postal code system (gray).

When a trip is booked in the United Arab Emirates, the only information provided as a destination or departure location are geospatial coordinates and addresses. These addresses are compared with long lists of matching addresses in the legacy database, as previously described with postal codes. Addresses and postal code systems do not provide universally interpretable and precise encodings of locations, especially for the locations that matter for this project: points and areas. Addresses can be ambiguous, addresses and postal codes can be imprecise, postal codes are not uniform, and some countries do not have a postal code system. In contrast, polygons would provide unique and precise location definition that is uniform and universal. But postal codes and addresses have been useful in the legacy system. When moving to other encoding techniques, this usefulness must be preserved.

2.3.4 Requirements for Location Matching

If the following statements are true for a given location encoding using the definitions of the Point and Area, the location encoding is useful and able to operate independently from the postal code and address systems.

2.4 Literature Review

In [3], CEO Chris Cheldrick explains how locations can be communicated more effectively by describing a three by three meter areas using deterministically assigned triplets of words. The system aims to solve the problem of ambiguity in address or postal code systems. The what3words API offers functionalities that can find what3word geocodings near a specified

Nr	Description
1.	Every location is stored in a database as a single entity
2.	Locations can consist of multiple locations (see figure 2.2)
3.	A deterministic predicate of whether a location is fully contained within a location is achievable
4.	A deterministic method of finding all locations containing a single location can be used
5.	A method of determining precedence of location in case of overlap must always yield one result, and discard all others
6.	Locations must be importable from external sources

Table 2.1 Location Matching Requirements

latitude and longitude location. The system is able to find results within a clamped area, as documented in [4], effectively acting like a spherical circle with a given radius in which points can be contained. In the paper [5] Markus elaborates on the distinction between structure-based spatial data and point sets, stating that: "Structure-based spatial data types have prevailed and form the basis of a large number of data models and query languages for spatial data". He elaborates on distinctions of operations and predicates between different spatial data models in [6]. Operations such as point-in-polygon test and intersection are categorized as spatial modeling. Regular spatial database systems support a basic Geometry hierarchy of Points, Polygons, MultiPoint and MultiPolygon Classes, as described in the OGC [7] and ISO 19125 [8] standard. MySQL, PostgreSQL, MariaDB and other systems having distinct implementations adhere to the OGC standard. Some other databases like MongoDB adopt the GeoJSON standard [9], providing similar operations and data types. Xiang et al proposed conventional flattened R-Tree indexing for the less mature MongoDB spatial system [10]. The built in Geohashing method is typically used to index points and centroids, having the possibility to inaccuracies and missing data. Locations should be importable in geography formats. Holmberg extracts data from OpenStreetMap as shape files, see [11, Chapter 6]. He uses two sources: <https://www.openstreetmap.org> and <http://download.geofabrik.de> see [11, Chapter 7.3]. The latter is used to obtain data for whole countries. OpenStreetMap offers a downloadable dataset at <https://planet.openstreetmap.org> from which geographic data can be exported. The OSM Nominatim Usage policy states that no heavy usage is allowed, that bulk geocoding is restricted, that auto-complete is not supported, and that attribution must be displayed [12].

2.5 Database Prerequisites

The database that is used must be able to aggregate all polygons containing a given point. Conversely, it must be able to aggregate all points that are contained within a given polygon. The scenario presented in image 2.4 should at least be replicable.

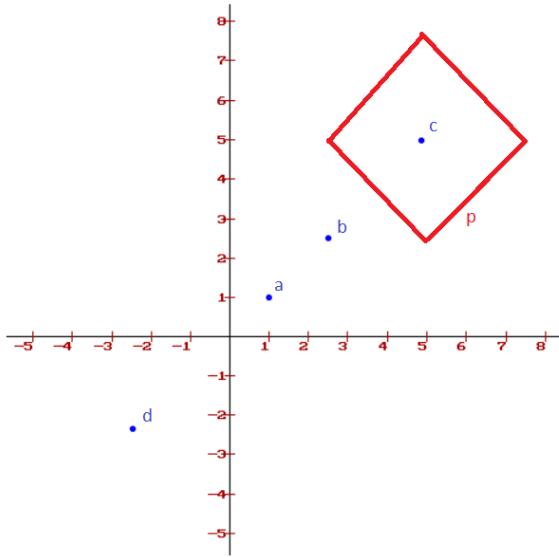


Fig. 2.4 Four Points, one Polygon p containing Point c.

As contained in Appendix A, this example provides a proof that a minimal requirement is satisfied, so that a list of candidate Database Management Systems could be constructed. In all cases, a polygon is a list of coordinates that define a closed path, meaning that the first and last index contain identical points.

2.5.1 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 [13] and MYSQL documentation [14] could be consulted. When a generic geometry column, or point column is created, points can be inserted as shown in snippet 2.1.

```

1 START TRANSACTION;
2 SET @a = ST_GeomFromText('POINT(1 1)');
3 INSERT INTO point (point) VALUES (@a);
4 SET @b = ST_GeomFromText('POINT(2.5 2.5)');
5 INSERT INTO point (point) VALUES (@b);
6 SET @c = ST_GeomFromText('POINT(5 5)');
7 INSERT INTO point (point) VALUES (@c);
8 SET @d = ST_GeomFromText('POINT(-2.5 -2.5)');
9 INSERT INTO point (point) VALUES (@d);
10 # also insert @b, @c, and @d
11 COMMIT;
12
13 START TRANSACTION;
14 # First and last point should be the same
15 SET @a = PolygonFromText('POLYGON((2.5 5,5 7.5,7.5 5,5 2.5,2.5 5))');
16 INSERT INTO polygon (polygon) VALUES (@a);
17 COMMIT;

```

Listing 2.1 Insert four points and one polygon in MySQL.

It is evident that c is contained in p. To determine which points are contained in p, the function as seen in Snippet 2.2 can be used, which returns the point with coordinates [5, 5] as expected.

```

1 // All points contained in polygon
2 SELECT ST_ASTEXT(POINT)
3 FROM POINT
4 WHERE
5 ST_CONTAINS(
6 (
7     SELECT POLYGON
8     FROM POLYGON
9     WHERE id = 1
10 ),
11 POINT
12 )
13
14 // All polygons containing point
15 SELECT ST_ASTEXT(POLYGON)
16 FROM POLYGON, POINT
17 WHERE
18 POINT.id = 3 AND ST_CONTAINS(
19     POLYGON.polygon,
20     POINT.point
21 )

```

Listing 2.2 Select points contained in polygon, and all polygons containing a point in MySQL.

A multipolygon can be inserted using triple braces, indicating a collection of polygons to be inserted as seen in Figure 2.3. The MultiPolygon class is able to support multiple polygons to be stored as a single entity. The standard provides containment predicate, and

methods to distinguish larger locations from smaller ones, which could be used in precedence checks.

```

1 START TRANSACTION;
2 # First and last point should be the same
3 SET @a = GeomFromText('MULTIPOLYGON(((1 1,2 2,3 3,1 1)),((5 5,6 6,8 8,5 5)))');
4 INSERT INTO multipolygon (multipolygon) VALUES (@a);
5 COMMIT;
```

Listing 2.3 Insert one multipolygon in MySQL.

2.5.2 OpenGIS Incompatible databases

MongoDB doesn't offer OpenGIS implementations but has geospatial query operators that may provide enough functionalities for current requirements [15]. 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 2.4 is recreated in MongoDB using queries shown in snippets 2.4 and 2.5. Geometry datatypes can be inserted as objects having a type and coordinates property. A polygon can be inserted in the same manner, having multiple points as a list instead of a single point.

```

1 db.point.insertMany([
2   { shape: { type: "Point", coordinates: [1, 1] } },
3   { shape: { type: "Point", coordinates: [2.5, 2.5] } },
4   { shape: { type: "Point", coordinates: [5, 5] } },
5   { shape: { type: "Point", coordinates: [-2.5, -2.5] } },
6 ])
7
8 db.polygon.insert({
9   shape: {
10     type: "Polygon",
11     coordinates: [ [2.5, 5], [5, 7.5], [7.5, 5], [5, 2.5], [2.5, 5] ]
12   }
13 })
14
15 db.point.createIndex({ 'shape': '2dsphere' })
16 db.polygon.createIndex({ 'shape': '2dsphere' })
```

Listing 2.4 Insert four points and one polygon in MongoDB.

A method named `$geoWithin` can be used to return points that are contained within the polygon. Conversely, all polygons that contain a certain point can be queried using the `$geoIntersects` method as seen in 2.5.

```

1 // All points contained in polygon
2 var p = db.polygon.find({})
3
4 db.point.find(
5   shape: {
```

```

6   $geoWithin: {
7     $polygon: [
8       [2.5, 5],
9       [5, 7.5],
10      [7.5, 5],
11      [5, 2.5],
12      [2.5, 5]
13    ]
14  }
15 }
16 })
17
18 // All polygons containing point
19 var p = db.point.findOne({ coordinates: [5, 5] })
20
21 db.polygon.find({
22   shape: {
23     $geoIntersects: {
24       $geometry: {
25         type: "Point",
26         coordinates: [5, 5]
27       }
28     }
29   }
30 })

```

Listing 2.5 Select points contained in polygon, and all polygons containing a point in MongoDB.

In MongoDB, a multipolygon can be inserted using extra pairs of braces, as shown in 2.6. Any predicate will fail if the type is defined as 'Polygon', but a MultiPolygon is stored in the coordinates property or vice versa. Therefore, it is important to manage the type property as more polygons are to be stored at once.

```

1 db.polygon.insert({
2   shape: {
3     type: "MultiPolygon",
4     coordinates: [
5       [ [ [2.5, 5], [5, 7.5], [7.5, 5], [5, 2.5], [2.5, 5] ] ],
6       [ [ [2.5, 5], [5, 7.5], [7.5, 5], [5, 2.5], [2.5, 5] ] ]
7     ]
8   }
9 })

```

Listing 2.6 Insert one multipolygon in MongoDB.

2.6 Overlapping Locations

If the destination is contained in several polygons associated with multiple rules, which rule should then be used to calculate the final price? A database will just pick the first result when the results are limited to one. Several solutions have been proposed to solve this problem:

1. Using the location with the shortest distance from its centroid to the destination.
2. Picking the location with the smallest area.
3. Picking the location that has the rule with the lowest price.
4. Picking the rule that has the highest precedence assigned by the user.
5. A combination of these proposals.

2.7 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 in [16]. 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, as per [17]. After the paper Schmid et al. 2015 [18] was published. The team argues that clustering is much easier in MongoDB, which may be important in the future when the company grows.

2.8 Locations Conclusion

Every point and area on earth can be described by geometric datatypes such as Polygons or MultiPolygons, yielding a consistent and universal solution of storing spatial data. This means that a new system does not have to rely on address and postal code data required by the legacy system. Some alternative encodings like what3words may be used to encode distinct points, but areas become quite a challenge in such systems. All physical locations located on the earth's surface that were described by addresses and postal codes can be encoded by spatial datatypes. The multipolygon may be used to represent more complex areas like the city Amsterdam as seen in 2.2. The OGC standard offers a mature and thorough set of geospatial features that MongoDB does not support. As the required data types, operations and predicates exist in both SQL and NoSQL, it is beneficial to opt for MongoDB for its performance and alignment with the team's experience. Although if robustness is desired,

or extra GIS functionalities required, SQL should be taken into consideration. Overlapping areas in conflicting rules can be resolved by assigning priorities to rules. This enables users to reason about their pricing rules more easily, and makes the intent of these pricing rules more interpretable for other people.

References

- [1] U. T. Inc. (2011) The uber story. [Online]. Available: <https://www.uber.com/en-NL/our-story/>
- [2] (2018) Mysql 5.7 reference manual - spatial data types. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/spatial-types.html>
- [3] C. Sheldrick, “Efficient and future-proof.” April 2018. [Online]. Available: <https://what3words.com/2018/04/read-our-white-paper-efficient-and-future-proof/>
- [4] (2016) what3words - restful api. [Online]. Available: <https://docs.what3words.com/api/v2/#autosuggest-params>
- [5] M. Schneider, “Spatial data types.” [Online]. Available: <https://www.cise.ufl.edu/~mschneid/Research/papers/Sch09BoChb.pdf>
- [6] ——. Conceptual foundation for the design and implementation of spatial database systems and gis. [Online]. Available: <https://www.cise.ufl.edu/~mschneid/Service/Tutorials/TutorialSDT.pdf>
- [7] J. R. Herring, “Implementation standard for geographic information - simple feature access - part 1: Common architecture.” May 2011. [Online]. Available: http://portal.opengeospatial.org/files/?artifact_id=25355
- [8] (2004) Geographic information – simple feature access – part 1: Common architecture. [Online]. Available: <https://www.iso.org/standard/40114.html>
- [9] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub. (2016) The geojson format. [Online]. Available: <https://tools.ietf.org/html/rfc7946>
- [10] L. Xiang, , X. Shao, and D. Wang. (2016) Providing r-tree support for mongodb . [Online]. Available: <https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLI-B4/545/2016/isprs-archives-XLI-B4-545-2016.pdf>
- [11] K. Holmberg, “On using openstreetmap and gps for optimization,” Linköping Institute of Technology, SE-581 83 Linköping, Sweden, Tech. Rep., November 2015. [Online]. Available: <https://www.diva-portal.org/smash/get/diva2:867759/FULLTEXT01.pdf>
- [12] O. Tech. (2017) Nominatim usage policy. [Online]. Available: <https://operations.osmfoundation.org/policies/nominatim/>
- [13] (2018) Postgis 2.4.5dev manual. [Online]. Available: https://postgis.net/docs/manual-2.4/using_postgis_dbmanagement.html#PostGIS_GeographyVSGeometry

- [14] (2018) Mysql 5.7 reference manual - geometry class. [Online]. Available: <https://dev.mysql.com/doc/refman/5.7/en/gis-class-geometry.html>
- [15] (2018) Geospatial query operators — mongodb manual 3.6. [Online]. Available: <https://docs.mongodb.com/manual/reference/operator/query/geospatial/>
- [16] K. R. Sarthak Agarwal, “Analyzing the performance of nosql vs. sql databases for spatial and aggregate queries.” September 2017. [Online]. Available: <https://scholarworks.umass.edu/cgi/viewcontent.cgi?article=1028&context=foss4g>
- [17] (2015) Geospatial performance improvements in mongodb 3.2. [Online]. Available: <https://www.mongodb.com/blog/post/geospatial-performance-improvements-in-mongodb-3-2>
- [18] W. R. Stephan Schmid, Eszter Galicz, “Performance investigation of selected sql and nosql databases.” June 2015. [Online]. Available: https://agile-online.org/conference_paper/cds/agile_2015/shortpapers/68/68_Paper_in_PDF.pdf
- [19] (2011) Iso/iec 25010:2011. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en>
- [20] I. K. Center. (2018) Three-tier architectures. [Online]. Available: https://www.ibm.com/support/knowledgecenter/en/SSAW57_8.5.5/com.ibm.websphere.nd.doc/ae/covr_3-tier.html
- [21] R. Stephens, *Beginning Software Engineering*. John Wiley & Sons, 2015.
- [22] P. Hauer. (2015) Microservices in a nutshell. pros and cons. [Online]. Available: <https://blog.philippauer.de/microservices-nutshell-pros-cons/>
- [23] J. Stenberg. (2014) Experiences from failing with microservices. [Online]. Available: <https://www.infoq.com/news/2014/08/failing-microservices>
- [24] N. S. M. Jones, J. Bradley, “Json web token (jwt).” May 2015. [Online]. Available: <https://tools.ietf.org/pdf/rfc7519.pdf>
- [25] M. Palladino. (2019) Microservices & api gateways. [Online]. Available: <https://www.nginx.com/blog/microservices-api-gateways-part-1-why-an-api-gateway>
- [26] (2018) Loopback 3.x - loopback documentation. [Online]. Available: <https://loopback.io/doc/en/lb3>
- [27] (2018) Mocha - the fun, simple, flexible javascript test framework. [Online]. Available: <https://mochajs.org>
- [28] (2018) Chai assertion library. [Online]. Available: <http://chaijs.com>
- [29] (2015) Mongodbs evaluation query operators. [Online]. Available: docs.mongodb.com/manual/reference/operator/query/regex/#index-use
- [30] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

List of figures

2.1	LatLngSphere	7
2.2	Amsterdam	10
2.3	PostalCodes	11
2.4	Square	13
3.1	Current System Architecture	20
3.2	Database Schema	22
3.3	OAuth 2.0	25
3.4	Stateless JWT	26
3.5	API Gateway	27
4.1	Class Diagram	32
4.2	DataModel	33
4.3	Calculation Flow	36

List of tables

1.1	Fixed Prices	2
1.2	Planning	6
2.1	Location Matching Requirements	12