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

- Keep trying to find if other people found solutions for determining precedence of polygon matches 13
- Currently implementing solution 16
- 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, ... 28

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 9
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 12
2.5 Database Prerequisites 13
2.5.1 OpenGIS Compatible databases 13
2.5.2 OpenGIS Incompatible databases 15
2.6 Overlapping Locations 16
2.7 Performance and Clustering Trade-offs 17
2.8 Conclusion 17

3 System Architecture **17**

3.1 Introduction 17
3.2 Architectural Patterns 17
3.2.1 Monoliths 18

3.2.2 Microservices . . . 19
3.2.3 Frontend and Backend . . . 19
3.3 Information Dependencies . . . 20
3.4 Authentication and Authorization . . . 21
3.4.1 OAuth 2.0 . . . 22
3.4.2 JSON Web Tokens . . . 23
3.4.3 API Gateway . . . 24
3.5 Suitability of Methods and Techniques . . . 25
3.6 Conclusion . . . 26

4 Trip Price Calculation System 27
4.1 Introduction . . . 27
4.2 The System . . . 27
4.3 Breakdown . . . 29
4.4 Timeframes . . . 31
4.4.1 Conventional Approach . . . 31
4.4.2 Bitmap . . . 31
4.5 Distance and Duration . . . 33
4.6 Aggregate Query . . . 34
4.6.1 Identification . . . 34
4.6.2 Links . . . 34
4.6.3 Country . . . 35
4.6.4 Rules . . . 35
4.6.5 Discounts . . . 35
4.6.6 Products . . . 35
4.6.7 Prices . . . 35
4.6.8 Sorting and Formatting . . . 35
4.7 Price Calculation Types . . . 35
4.8 Threshold Calculations . . . 35
4.9 Logical Flow of a Price Calculation . . . 35
4.10 Conclusion . . . 36

5 Proposed Portal Solution 35
5.1 Introduction . . . 35
5.2 Required Views . . . 35
5.2.1 Entities . . . 35
5.2.2 Hierarchy . . . 35

5.3 Methods and Techniques . . . 36
5.3.1 Pricing Rules . . . 36
5.3.2 Timeframes . . . 36
5.4 . . . 36
5.5 . . . 36

6 Realization **37**
6.1 Introduction . . . 37
6.2 Methods and Techniques . . . 37
6.3 Sprint 1 - Dynamic Price Calculations . . . 37
6.4 Sprint 2 - Authentication and Authorization . . . 38
6.5 Sprint 3 - Setting up the Portal . . . 38
6.6 Sprint 4 - Expanding the Portal . . . 38
6.7 Sprint 5 - Expanding the Portal . . . 39
6.8 Sprint 6 - Locations . . . 39
6.9 Result . . . 39

7 Conclusion **41**

8 Recommendations **43**
8.1 Frontend . . . 43
8.2 Backend . . . 43
8.3 Functionalities . . . 44
8.3.1 Authentication section Authorization . . . 45
8.4 Database . . . 45
8.4.1 section Interface . . . 45

References **47**

List of figures **49**

List of tables **51**

Appendix A Pregame **53**

Appendix B Sprint Review and Proposal Slides **89**
B.1 Sprint 1 - review . . . 89
B.2 Sprint 2 - breakdown . . . 101
B.3 Sprint 2 - authentication . . . 113

B.4 Sprint 2 - review 122
B.5 Sprint 3 - review 138
B.6 Sprint 4 - review 147

Chapter 4

Trip Price Calculation System

4.1 Introduction

In the previous chapter, information dependencies were discussed. This chapter clarifies which information should comprise a price breakdown to reflect that of the legacy system, what logical flow of information is to be contrived, and how different pieces of information that are stored and processed should restrict the time and space dimensions of a price rule without blurring the straightforwardness of the system.

4.2 The System

The object-oriented programming (OOP) paradigm offers many ways to keep a system structured. Words like 'organized', 'structured', and 'modular' boil down to the same idea: 'a cohesive whole built up of distinct parts'. Good software design has low coupling and high cohesion, meaning that software components should have a high degree of belongingness, and a low degree of dependence in respect of eachother. As stated in the previous chapter, another paradigm was used that compliments price calculation very well called: functional programming (FP). These two paradigms together are capable of providing a modular system that is highly cohesive, and very low coupled. To understand more about the structure of the system, a class diagram visualizes the most important components in 4.1.

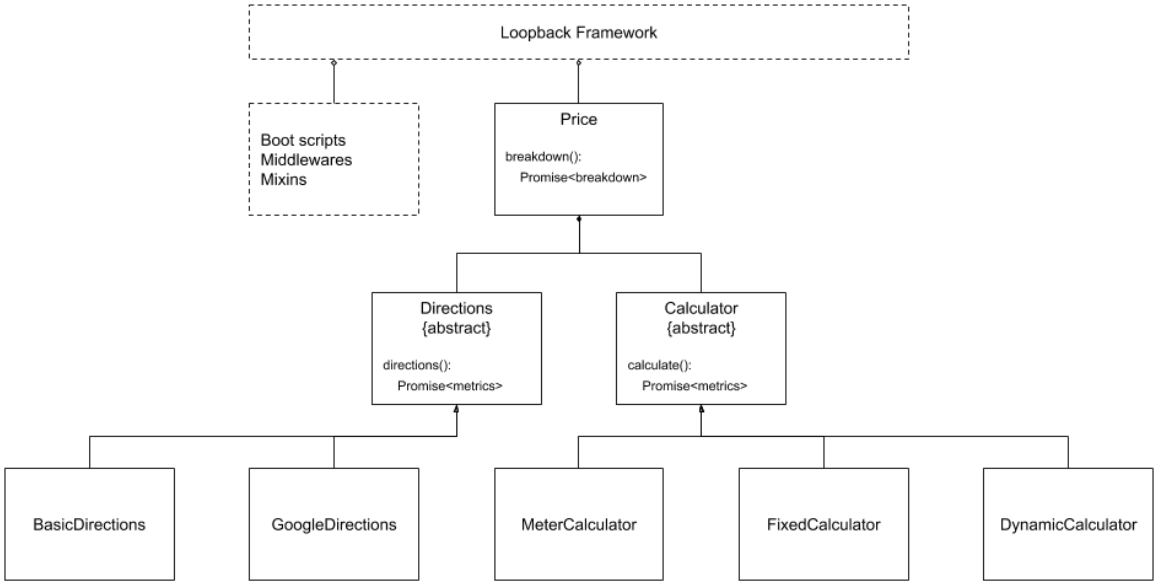

Fig. 4.1 Class diagram.

The Price object is composed of The instance methods shown in the diagram are are used by the associated class that is composed of Within each component, state and mutations are fully encapsulated, leaving only static functions exposed. These functions aim to mutate data in a pure way, meaning that no state is changed outside of the function scope, and that the function is absolutely honest about its parameters and return values. As discussed in the previous chapter, Typescript plays an important role in mixing OOP and FP together.

kks32: 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,

The database schema desgin as shown in the previous chapter gives an inpression on the different pieces of information required to calculate a price. Such a schema provides a good insight in the relationships that different entities have, but may distract from the actual story that is happening within each calculation. In Figure 4.2 a conceptual model can be seen having association and composition relations in UML notation. This model will be used to refer to throughout this chapter.

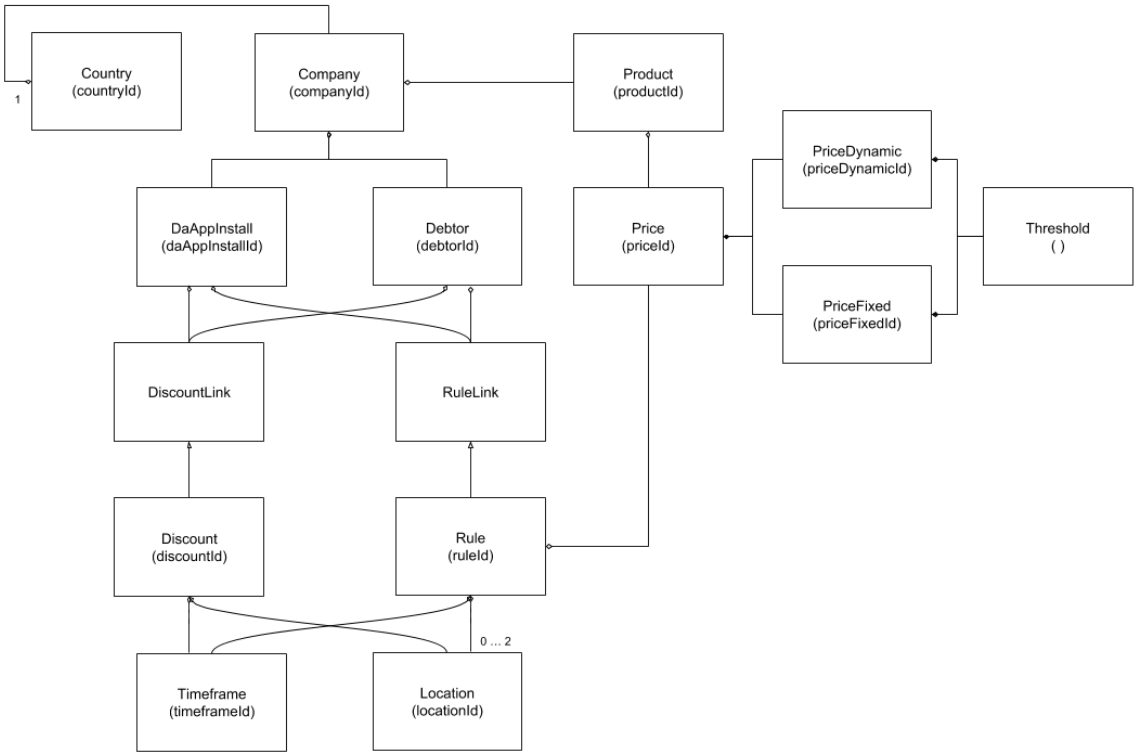

Fig. 4.2 Conceptual data model showing database entity relations.

4.3 Breakdown

To ensure a seamless transition from the legacy price calculation system to TPS, the response formats should be identical. Still an improvement, if profitable enough, could be taken into consideration. One requirement of the price breakdown states that the tax should be included, but as shown in Listing 4.1 the included tax is part of the breakdown. Is it by mistake or design?

```
[
  {
    "vehicleType": "saloon",
    "maxPassengers": "4",
    "price": {
      "currency": "EUR",
      "total": 850,
      "breakdown": {
        "route": 802,
        "tax": 48,
        "toll": 0,
        "parking": 0,
        "waiting": 0,
        "discount": 0
      }
```

```
    },
    "fixedPrice": "true"
  }
]
```

Listing 4.1 Legacy price breakdown

```
[
  {
    "vehicleType": "estate",
    "maxPassengers": 4,
    "isEstimated": false,
    "price": {
      "breakdown": {
        "route": 8300,
        "toll": 0,
        "parking": 0,
        "waiting": 0,
        "discount": -1650
      },
      "currency": "EUR",
      "total": 6650,
      "tax": {
        "amount": 400,
        "percentage": 6
      }
    }
  },
  ...
]
```

Listing 4.2 Improved price breakdown

Two possible solutions were proposed having VAT included in the price. The first solution extracts the tax element from the breakdown, so that the sum of the breakdown would add up to the total price where VAT is included in the price as shown in Listing 4.2. As demonstrated in Appendix B.2, a breakdown is easily constructed in four steps when VAT is included. Keep in mind that unlike the listings the prices in the proposal are not displayed in cents. The second solution maintains the legacy format, but has to recalculate the prices without VAT. This could have downsides unlike the first approach:

1. If an error is detected in the calculation, it is hard to trace back which components contributed to the total VAT. This would be even harder when each component uses its own VAT percentage.
2. It takes extra steps to calculate the price of each component excluding VAT.
3. Rounding the individual components could result in a sum that is not equal to the total displayed in the breakdown.

The first proposal is chosen to be implemented, where the flag 'fixedPrice' is replaced by the 'isEstimated' flag to clearly reflect its purpose.

4.4 Timeframes

Next to the three dimensions of space, time will play a role in determining whether a rule has matched. The implementation of this concept should preferably offer enough freedom in the future, and should not be tailored toward one specific entity relation. Being able to reuse the timeframe entity improves maintainability of the system. The requirements state that the user must be able to define a start and end time, the days on which the times are active, and the start and end date of the timeframe. This either means that the timeframe one window of time, or that each given day has a single window of time. But if a discount should be active during night of New Years Eve, between 23h and 5h, this description would not be sufficient to cover this use case under any interpretation.

4.4.1 Conventional Approach

The legacy system takes a straight forward approach of storing time in a relational database. The begin and end of a window are stored in a record that is related to a parent timeframe entity. The timeframe has many windows that could contain a timestamp. It either finds one or many time windows that contain the timeframe. This approach covers all possibilities imaginable.

4.4.2 Bitmap

For this reason, a proposal was made to implement timeframes in a way that let users choose to describe each hour of the week, being stored as a bit map. The windows could be decreased to half an hour, resulting in twice as many bits. Three implementations have been tested, where the bitstring format offered the best outcome, as seen in B.6. A timeframe is stored having two ISODates (international standard: ISO 8601), and a bitstring representing the schedule for which the insert statement is shown in Listing 4.3.

```
db.Timeframe.insert({
  startDate: new Date(2018, 4, 7),
  endDate: new Date(2019, 4, 7),
  weekSchedule:
    "001101000110011011000011
    011010110011000010111100
    101010101110100011111000
    111110011111011100100001
    101000000010111011100100
    110010000001000010101101
    010111101000000101001110"
})
```

Listing 4.3 Improved timeframe.

A string is a very flexible datatype. Using a regex in a query makes checking multiple bits in the string relatively easy, and enables different values next to 0 and 1. 3. A bitarray would only allow for 0 and 1 to be used. A bitstring also makes querying the data really stable, as the query will simply not match if the content of the data is not of expected length or value. Performance is not an issue if the regex column is indexed, and when prefix expressions (/^/) are used, as per documentation in [22]. As noted before, the system is easy to scale if existing data can be migrated to deal with a new amount of bits, or new character usage over bits.

```
/**
 * Date object days start at sunday, in order let monday be
 * index 0, decrease the index by one, but limit numbers
 * in the range of [0, 7).
 */
const startMonday = (d: number) => (d - 1) % 7;

/**
 * Creates a regex that spreads bits across hours of each
 * day of the week.
 */
export const regexFromDate = (date: Date) => {

  const skip =
    // Day of the week multiplied by hours a day
    startMonday(date.getDay()) * 24
    // Hour of the day
    + date.getUTCHours();

  return { skip, timeRegex: new RegExp(`^.{${skip}}1`) };
};
```

Listing 4.4 Opening timeframe.

Skip is an integer representing the number of bits that should be skipped to get to the moment represented by the date. So in order to get 11 AM - 12 AM in the presented schedule, 3 * 24 skips + 11 skip = 83 skips are to be made to find the digit 1 on thursday.

4.5 Distance and Duration

The directions class provides an interface to retrieve trip related data. The BasicDirections class returns a base case result, while the GoogleDirections class retrieves the data from the Google Directions API. The trip price calculation flow changes drastically when no destination or departure locations are provided in the request body. During development, many requirements were added and removed, handling different cases, returning different results. For this reason, a base case and the google case are defined using a behavioral strategy pattern as described in [23], only using an abstract class instead of an interface.

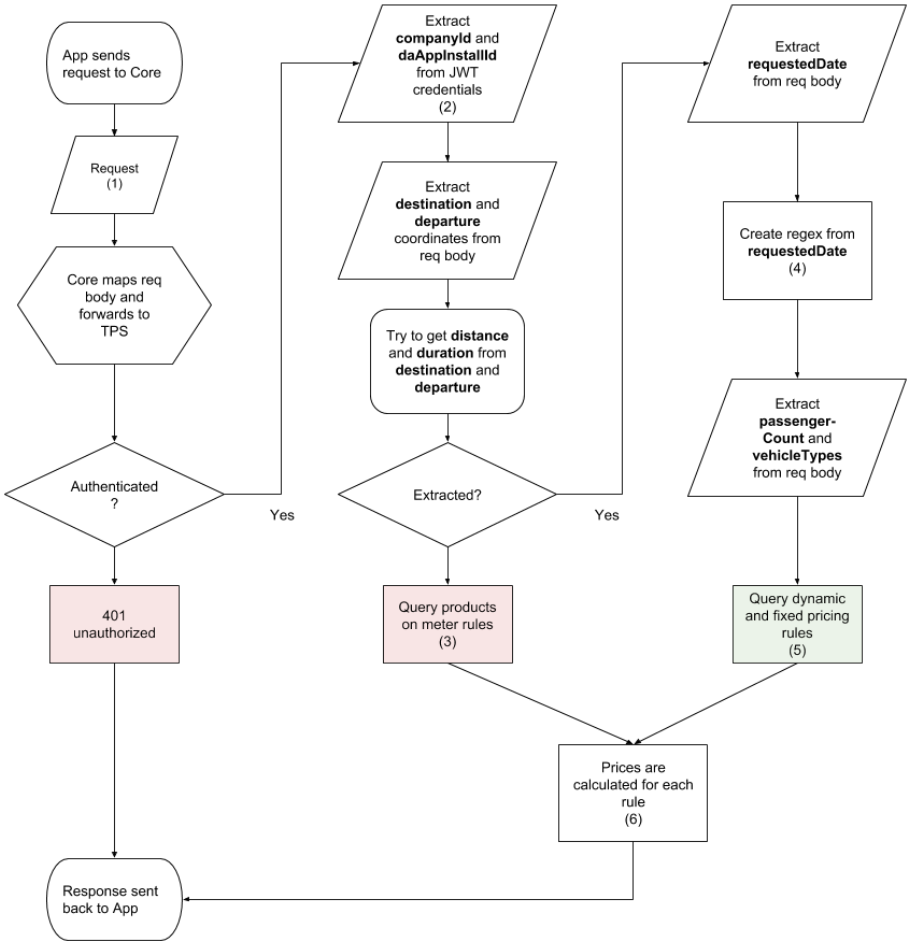

Fig. 4.3 The flow of a trip price calculation.

4.6 Aggregate Query

4.6.1 Identification

companyId daAppInstallId

4.6.2 Links

polymorphic hasthroughmany relation

4.6.3 Country

4.6.4 Rules

4.6.5 Discounts

4.6.6 Products

4.6.7 Prices

4.6.8 Sorting and Formatting

4.7 Price Calculation Types

4.8 Threshold Calculations

4.9 Logical Flow of a Price Calculation

- authentication - extracting companyId and daAppInstallId - extracting departure and destination coordinates - using directions service to aqquire distance and duration - converting to minutes and km - executing query - find daAppInstall model with matching companyId and daAppInstallId - find company country - find matching discounts ordered by priority - enabled - timeframes - departure - destination - find matching rules ordered by priority - enabled - timeframes - departure - destination - get pricing information of all products for highest priority rule

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. 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.

4.10 Conclusion

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] 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

[4] (2004) Geographic information – simple feature access – part 1: Common architecture. [Online]. Available: https://www.iso.org/standard/40114.html

[5] C. Sheldrick, "Efficient and future-proof." April 2018. [Online]. Available: https://what3words.com/2018/04/read-our-white-paper-efficient-and-future-proof/

[6] (2016) what3words - restful api. [Online]. Available: https://docs.what3words.com/api/v2/#autosuggest-params

[7] (2018) Postgis 2.4.5dev manual. [Online]. Available: https://postgis.net/docs/manual-2.4/using_postgis_dbmanagement.html#PostGIS_GeographyVSGeometry

[8] (2018) Mysql 5.7 reference manual - geometry class. [Online]. Available: https://dev.mysql.com/doc/refman/5.7/en/gis-class-geometry.html

[9] (2018) Geospatial query operators — mongodb manual 3.6. [Online]. Available: https://docs.mongodb.com/manual/reference/operator/query-geospatial/

[10] 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

[11] (2015) Geospatial performance improvements in mongodb 3.2. [Online]. Available: https://www.mongodb.com/blog/post/geospatial-performance-improvements-in-mongodb-3-2

[12] 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

[13] (2011) Iso/iec 25010:2011. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

[14] 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

[15] R. Stephens, *Beginning Software Engineering*. John Wiley & Sons, 2015.

[16] P. Hauer. (2015) Microservices in a nutshell. pros and cons. [Online]. Available: https://blog.philipphauer.de/microservices-nutshell-pros-cons/

[17] J. Stenberg. (2014) Experiences from failing with microservices. [Online]. Available: https://www.infoq.com/news/2014/08/failing-microservices

[18] N. S. M. Jones, J. Bradley, "Json web token (jwt)." May 2015. [Online]. Available: https://tools.ietf.org/pdf/rfc7519.pdf

[19] M. Palladino. (2019) Microservices & api gateways. [Online]. Available: https://www.nginx.com/blog/microservices-api-gateways-part-1-why-an-api-gateway

[20] (2018) Mocha - the fun, simple, flexible javascript test framework. [Online]. Available: https://mochajs.org

[21] (2018) Chai assertion library. [Online]. Available: http://chaijs.com

[22] (2015) Mongodb evaluation query operators. [Online]. Available: docs.mongodb.com/manual/reference/operator/query/regex/#index-use

[23] 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 18
3.2 Database Schema 20
3.3 OAuth 2.0 23
3.4 Stateless JWT 24
3.5 API Gateway 25

4.1 Class Diagram 28
4.2 DataModel 29
4.3 Calculation Flow 34

List of tables

1.1 Fixed Prices 2
1.2 Planning 6

2.1 Location Matching Requirements 12