University of Applied Sciences
Rapperswil

Student Research Project Thesis

Truck Route

Kyburz, Cyril *Scheiwiller, Sandro*

supervised by
Mirko Stocker

Industrial Partner
Bernhard Zindel, Scoping AG

December 21, 2018

Abstract

Problem

Recycling containers need to be emptied regularly using trucks. *Live Track AG*, a new startup, wants to support the process of emptying these containers. To achieve this, the recycling containers will be equipped with sensors, which will transmit their current filling level to the cloud.

Goal

Based on the collected data the aim of this project is to assist the operator in planning routes for the collection of containers by providing various optimized routes. The routes should be optimized for the ratio between the amount loaded and the kilometers driven.

Procedure

To achieve this a stateless HTTP API was implemented which returns four optimized routes based on four different selection strategies. Furthermore a React app to demonstrate the power of the API was developed. Both applications were developed using the programming language TypeScript and deployed into the *AWS Cloud*. They also share a similar Continuous Integration and Continuous Deployment setup.

Result

The result is a fully tested and robust API that is vertically and horizontally scalable and allows to calculate optimized routes. It is documented with Swagger and paired with a web app to demonstrate and test the API.

Management Summary

Initial Situation

Live Track AG, a new startup, wants to support the process of emptying recycling containers. Currently the containers need to be emptied regularly using trucks. To accomplish a more efficient process, the recycling containers will be equipped with ultrasonic sensors or related technologies measuring the filling state. This data will then be synced into the cloud.

The collected data opens up new possibilities: As of now, an operator has to plan a route for the truck driver based on his experience. The goal of this project is to help the operator by providing different, optimized routes using the collected data. The route should be optimized for the ratio between the amount loaded and the kilometers driven. In addition, the truck should not necessarily be filled to the maximum.

Procedure, Technologies

Two applications were developed during this thesis:

- A HTTP API which calculates routes. The client sends the API a request, including a truck, a collection of containers and the start and end point of the route. Based on the data provided, the API returns multiple possible routes. The API was developed using TypeScript and deployed as a stateless *AWS Lambda function*.

  The process by which the routes are generated can be divided into three steps:

  1. The container cluster collection is first analyzed. For this analysis a dynamic programming in-advance algorithm to solve our version of the knapsack problem and a greedy algorithm using three different prioritization strategies are used. Depending on the strategy, the containers nearby, the containers with the highest filling level or the containers with the most volume are prioritized. This results in four different subsets of the original container cluster collection.
  2. For each subset an API request is sent to the *Google Directions API*. The Directions API solves the traveling salesman problem and returns a route where the container clusters are visited in an order as to minimize the travel time of the route.
  3. For each received solution of the Directions API, six key figures are calculated, such as load per kilometer, cost and duration of the journey. Based on the key figure m^3/km the solutions are sorted and returned, best first.

- A web app to demonstrate and test the API. A user can select container clusters, a truck and a content type and send the request to the API (shown

in Figure 1). After receiving the response, the different route candidates are displayed with all of their key figures and a map showing the route as seen in Figure 2. The web app was developed using TypeScript and the frontend library React.

Result

The result is a production ready API that is vertically and horizontally scalable and allows to calculate optimized routes. To meet our high quality requirements we followed state of the art coding guidelines. The API is also flexible and works for all regions of the world covered by the *Directions API*. It currently supports glass and waste containers. In the future the variety of supported container contents could be easily extended.

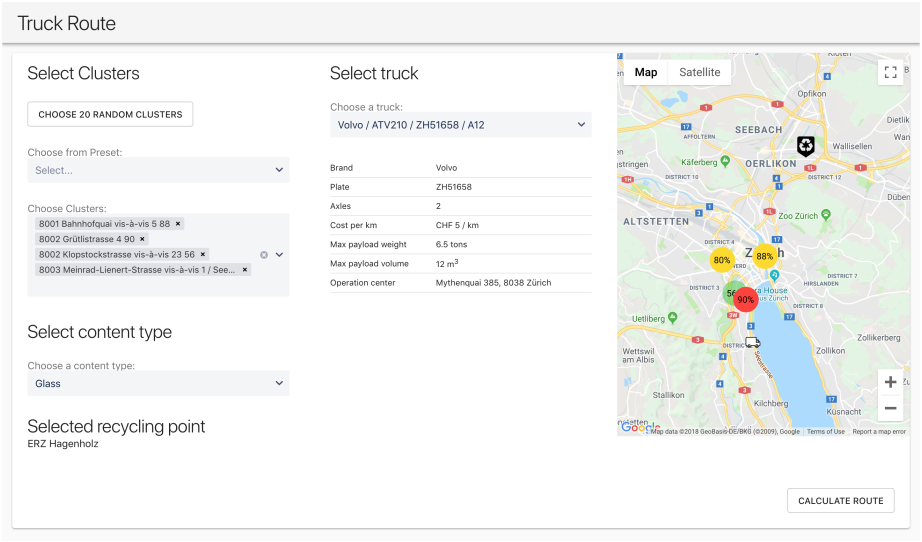


Figure 1: The form in which the user is able to select container clusters, a truck and a content type and send the request to the API.

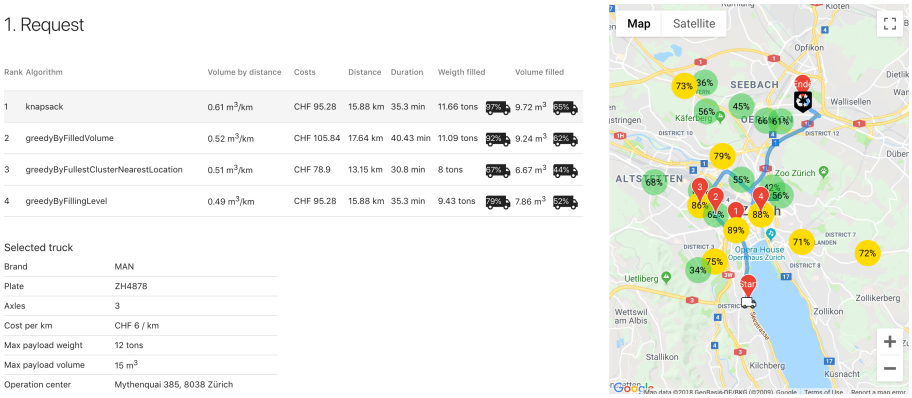


Figure 2: The result view in which the different algorithms can be compared to each other. In this result the knapsack algorithm performed best according to our chosen metric m^3/km.

Declaration of Authorship

We hereby declare that this thesis and the work presented in it is our own, original work. All the sources we consulted and cited are clearly attributed. We have acknowledged all main sources of help.

Location, Date: Rapperswil, 21. December 2018

Cyril Kyburz .....

Sandro Scheiwiller .....

Contents

1 Analysis

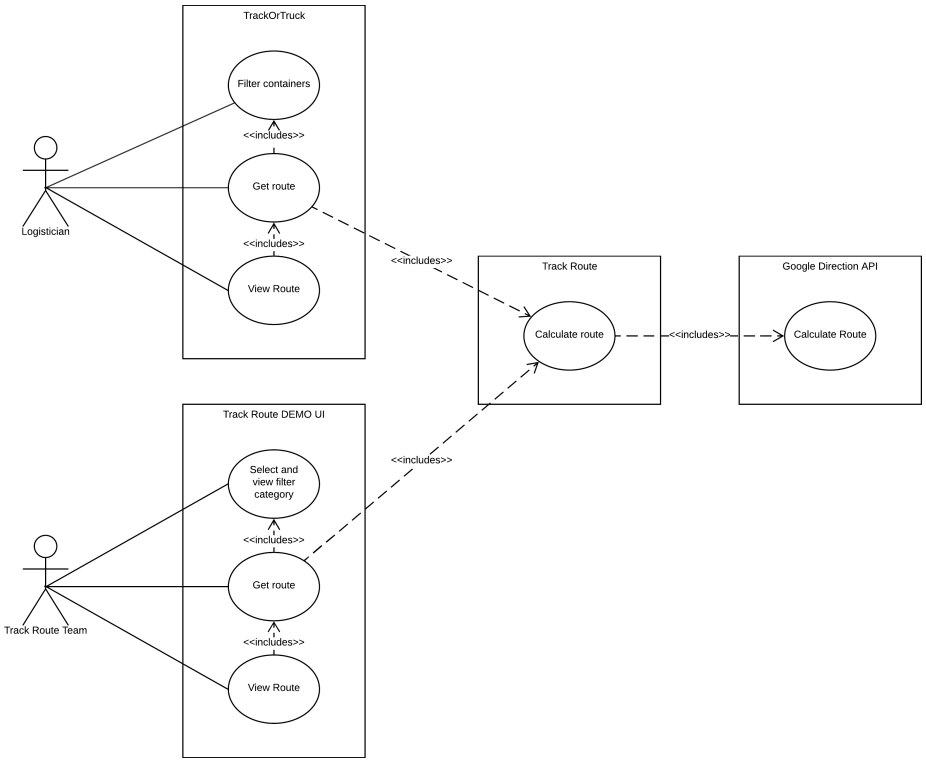
1.1 Initial Situation and Goal

Recycling containers need to be regularly emptied using trucks. *Live Track AG*, a new startup, wants to support the process of emptying these containers. To achieve this, the recycling containers will be equipped with sensors, which will transmit their current filling level to the cloud.

The collected data opens up new possibilities: As of now, an operator has to plan a route for the truck driver based on his experience. The goal of this project is to help the operator by providing different, optimized routes using the collected data. The route should be optimized for the ratio between the amount loaded and the kilometers driven. In addition, the truck should not necessarily be filled to the maximum.

1.2 Use-Cases

A use case is a list of actions or event steps typically defining the interactions between a role (human or other external system) and a system to achieve a goal.



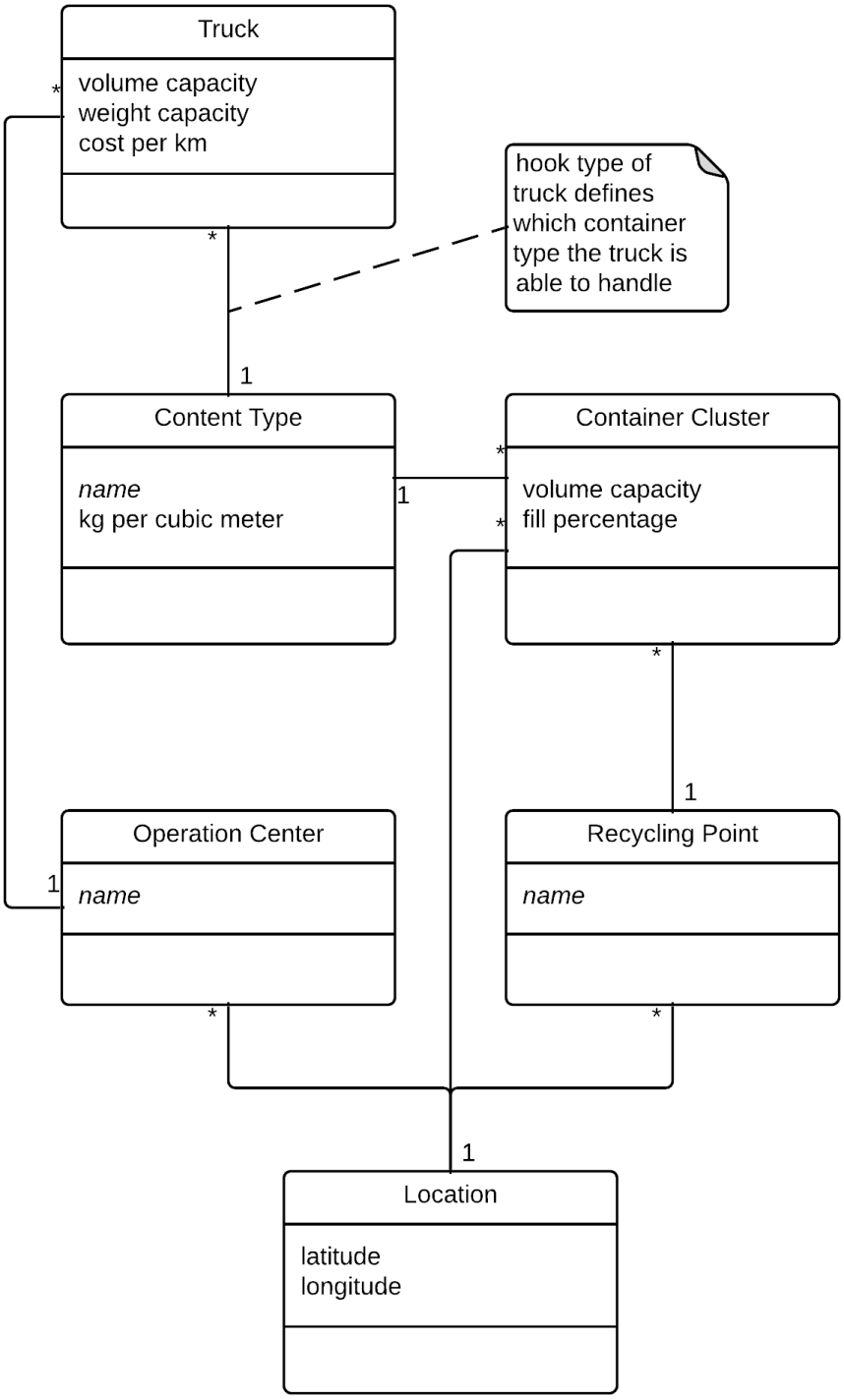
1.2.1 Actors

- **Logistician**
  The logistician uses the API indirectly via a user interface. His interests are to receive optimized routes.
- **Truck Route Team**
  The Truck Route Team wants to demonstrate and test the API.

1.2.2 Use Cases Brief

- **UC 01: Calculate Route**
  It returns multiple possible optimized routes based on the provided data. The data contains a truck, a collection of containers including their location and the start and recycling point of the route.
- **UC 02: Select and View Filter Category**
  The data which should be sent to the API can be selected and filtered.
- **UC 03: Get Route**
  The selected data is sent to the API.
- **UC 04: View Route**
  The API result can be inspected and the possible solutions can be compared.

1.3 Domain Model



1.3.1 Container Cluster

A container can be filled with recycled goods, e.g. glass or garbage.

Sometimes multiple containers of the same material are located at the same location. For example there are three containers containing green glass at the same location. In this case Truck Route does not need to consider each container for itself, instead the three containers are clustered together as one, as seen in Figure 3.

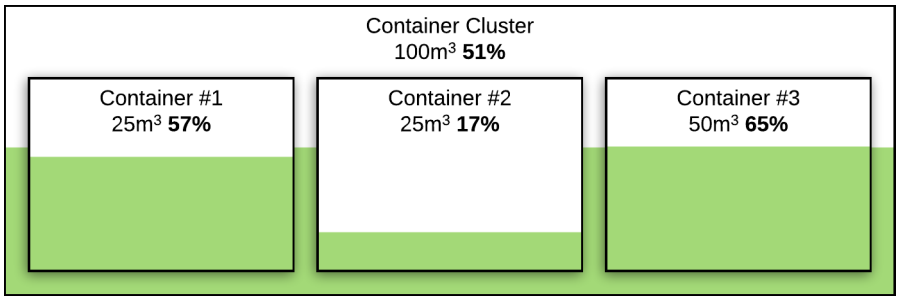


Figure 3: Multiple containers at the same location storing the same recycling material are clustered together as one container cluster.

1.3.2 Content Type

Each container cluster is associated with a type, e.g. glass. This type also contains information about the average weight of the recycled material in t/m^3.

Content Type	Conversion rate (t/m^3)
Glass	1.2
Garbage	0.1

1.3.3 Recycling Point

When the truck should be emptied it is driven to a recycling point. One challenge is that some containers cannot simply be emptied everywhere and instead are restricted to a certain recycling point defined by the owner of the containers.

1.3.4 Truck

Trucks are used to pick up containers. Each truck has a volume capacity (in cubic meters) and a weight capacity (in kilograms). Trucks are only able to empty containers using the right adapter for the head of the container. The connection between truck and container type indicates the container type a truck is able to empty with his current setup.

1.3.5 Operation Center

The trucks are parked in the operation center after use. Often the location of the operation center is the first starting point of the day and the last end point.

1.4 Functional Requirements

1.4.1 Container Selection

The selection of containers which should get emptied happens in multiple phases:

1. Multiple possible containers which can be emptied are selected.
2. A truck to pick up said containers is selected. The current location of the truck is the starting point.
3. Truck Route calculates multiple subsets of the clusters.

All these steps are necessary in order to define which clusters should be emptied. The containers used in the final route depend on:

- Selected starting point by operator.
- Selected clusters.
- Volume and weight capacity of selected truck.
- Truck Route subset finder algorithm.
- Metric used by Truck Route to calculate route efficiency.

Truck Route will implement different extensible algorithms:

Preconditions The algorithms expect that a few prerequisites are fulfilled:

- All containers which are selected by the operator can be emptied at the same recycling point.
- Material may not be mixed, the same material is expected to be in all containers.
- The containers can be emptied by the selected truck. This means the truck needs to have a compatible container adapter for all containers.

General conditions Requirements that all algorithms have to fulfill:

- Clusters must have a filling level of at least 70% before being considered for emptying. (Valid only for the greedy algorithms.)
- Truck does not have to get completely filled, but may also not get filled more than 100%.

Algorithms

1. **Greedy by Filling Level**
    - Clusters are selected based on there filling level.
    - Clusters with a filling level of over 90% must be emptied.
2. **Greedy by Filled Volume**
    - Clusters are selected based on there filled volume, cluster with the highest volume gets selected first.
3. **Greedy by Fullest Cluster Nearest Location**
    - Clusters are selected based on the the metric m^3/km.
4. **Knapsack**
    - Clusters are selected based on the the metric m^3/km.
    - Clusters must have a filling level of at least 50% before being considered for emptying.
    - Clusters must have a score of at least 1 m^3/km before being considered for emptying.

1.4.2 Route Creation

Once the container clusters that should be emptied have been selected, a third party API will be used to generate a route. This route is then returned to the client.

1.4.3 Cost Calculation

A excel sheet capable of calculating the costs based on usage will be created as part of the thesis.

1.5 Non-Functional Requirements

1.5.1 Performance

The Truck Route API should return a value in less than or equal to 4 seconds. As seen in Figure 4, requests from a client to the *Truck Route API* will always include at least four network calls, two between client and Truck Route and two between Truck Route and a third party API.

Since the client can call the *Truck Route API* from where-ever network latency between the client and the *Truck Route API* can vary a lot, so network latency from the client to the *Truck Route API* is ignored.

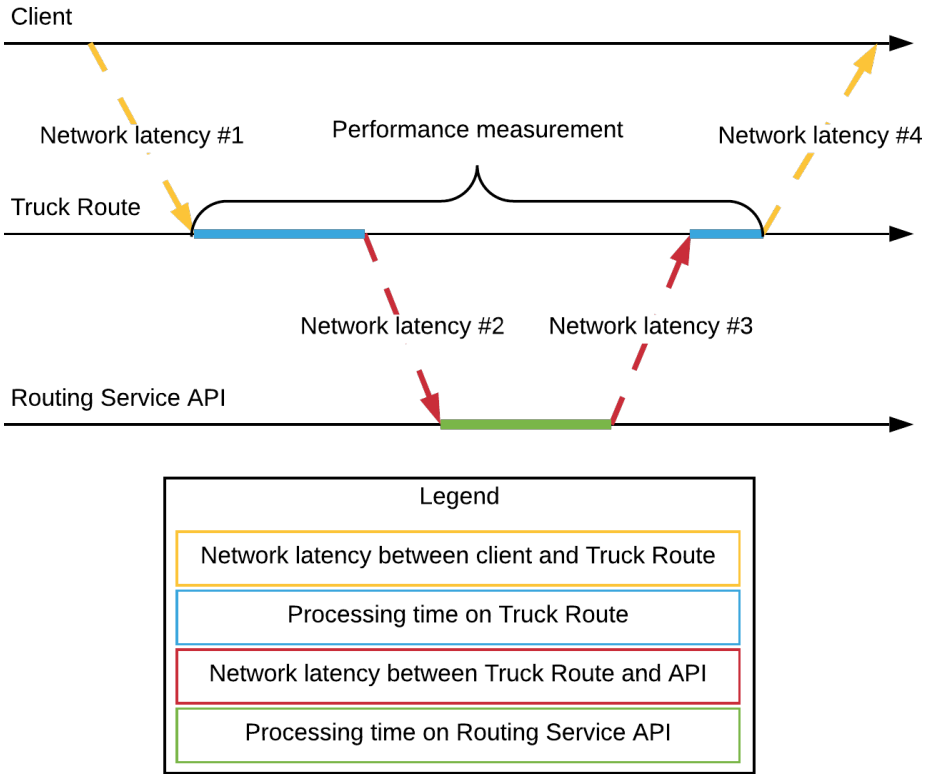


Figure 4: A diagram showing where time is spent from the call of the client to *Truck Route* until the client receives the response of *Truck Route*

1.5.2 Scalability

The application should be scalable. This is achieved by using stateless *AWS Lambda* which can be replicated as many times as necessary as described in an official guide [1] by *Amazon*:

> **For Scalability, Think concurrent requests:** Serverless applications take advantage of the concurrency model, and tradeoffs at the design level are evaluated based on concurrency.

1.5.3 Data Format

To transfer data between the different computers, **JSON** is used. JSON is the recommended output format of *Google Directions API* [2].

1.5.4 On Demand

For some application it makes sense to trigger execution automatically, for example:

- on a predicate (if the containers have a fill level of more than 90%)
- on a time interval (every two hours)

The decision for Truck Route was that, at least at the beginning, executing the service would be a manual task. Triggering the service automatically is possible, but not part of this thesis.

1.6 UX / UI Requirements

A mockup, shown in Figure 5, was created containing all use cases to discuss and validate a possible outcome of this work. From a discussion about the mockup several important decisions originated: the layout should be similar and container clusters should be marked on the map.

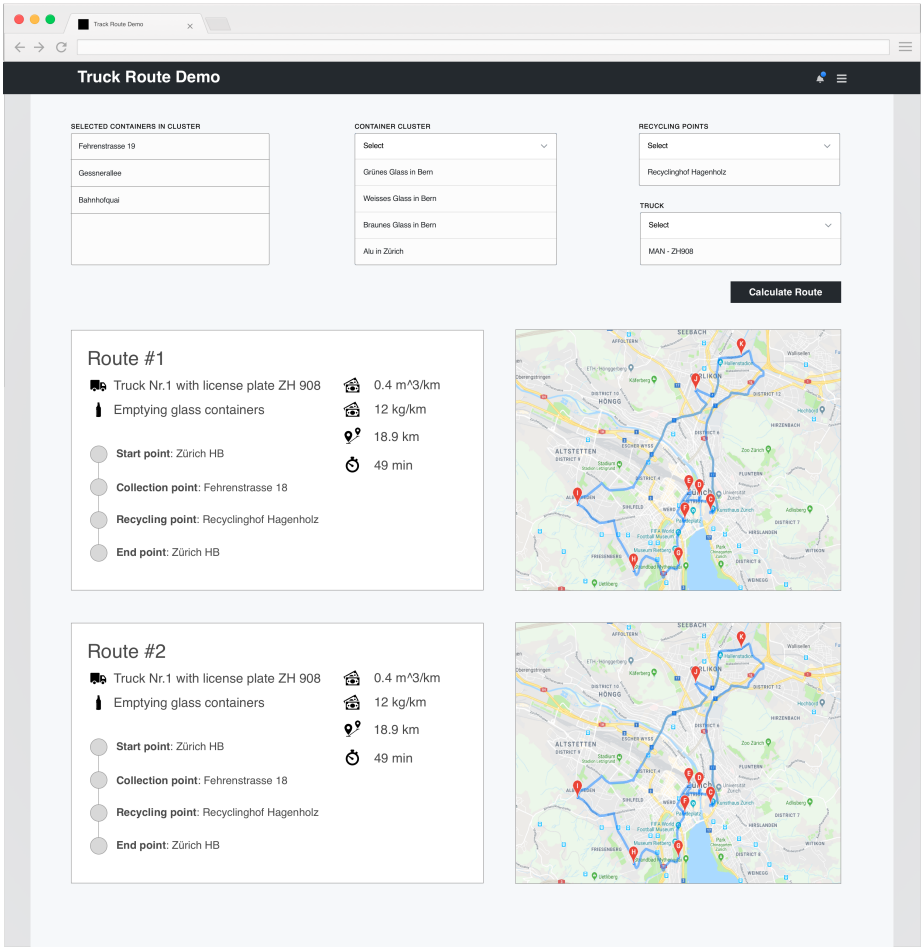


Figure 5: A mockup of our frontend demo showcasing all the use cases.

Further important key figures for a route were discussed and the following were selected for implementation and visualizations:

Name	Unit	Example
Cost per kilometer	currency	5
Total distance	kilometers	33.36
Total duration	minutes	43.78
Total volume	m^3	12.5
Total weight	t	5
Volume by distance	m^3/kilometers	1.21

1.7 Evaluation of Routing Service

For every evaluated routing service a prototype was developed to validate the requirements.

1.7.1 Requirements

Important requirements in the selection process for the routing service are:

- The ability to generate a route specifically for a truck. As the *Bing Maps Truck Routing API* states on its front page[3]:

  > The Bing Maps Truck Routing API provides travel routes which take truck attributes such as size, weight and type of cargo. This is important as not all trucks can travel the same routes as other vehicles. Here are some examples:
  >
  > – Bridges have heights and weight limits.
  > – Tunnels often have restrictions on flammable or hazardous materials.
  > – Longer trucks have difficulty making tight turns.
  > – Highways often have a separate speed limit for trucks.
  > – Certain trucks may want to avoid roads that have steep gradients.

- The power to solve the traveling salesman problem for given unsorted collection points.
- An appealing design and a user-friendly interface of the map component.

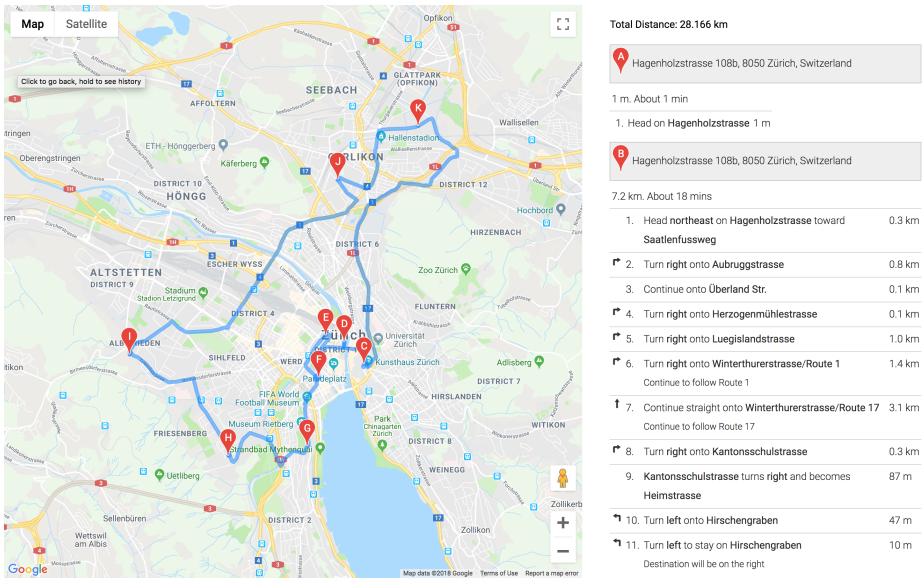
1.7.2 Google Directions API

Technical limitations

- Up to 27 waypoints (including start & end).
- Not able to generate routes for trucks specifically.

- No possibility to influence route by providing zones to avoid.[4] That means that even if we knew that a truck would not be allowed, for example, to cross a bridge, we would not be able to provide the API with this information.

Prototype This prototype using *Google Directions API* shows an optimized route.



Pricing Transparent pricing is a strength of *Google Direction API*.

Because we need to solve the travelling salesman problem, which Google calls "waypoints optimization", we would use the *Directions Advanced SKU* Therefore the following prices apply:

Monthly Volume Range	Price per Query
0–100,000	0.01 USD per each(10.00 USD per 1000)
100,001–500,000	0.008 USD per each(8.00 USD per 1000)
500,000+	Volume pricing with sales

Table 1: *Google Directions API* Advanced Pricing [5]

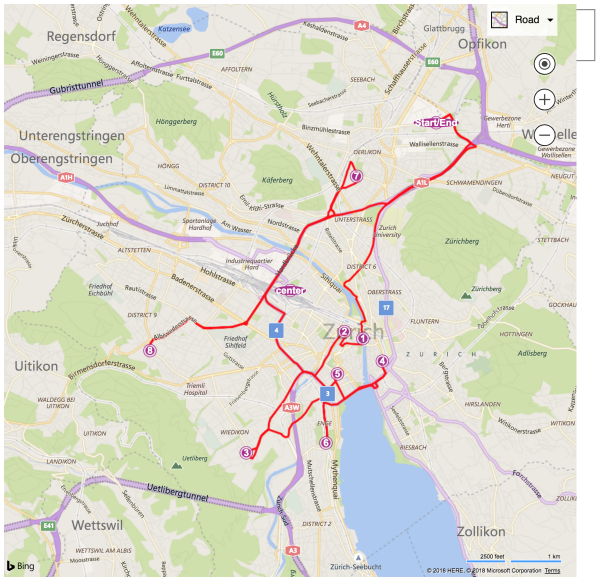
UI / UX *Google Directions API* has the big advantage that their map component is the well known and popular *Google Maps*. At of now, *Google Maps* sets the bar for navigation systems.

1.7.3 Bing Maps Truck Routing API

Technical limitations

- Up to 25 waypoints (including start & end)
- Generates routes for trucks

Prototype This prototype uses *Bing Maps Truck Routing API.*



Pricing Fasten your seatbelts, it is about to get complicated.

The *Bing Maps API* works with billable transactions. One call to this specific API will result in three billable transactions.[6] The licensing options[7] state:

> Application using up to 125,000 billable transactions per calendar year qualifies for a limited website and consumer app use. Applications that qualify for a limited website and consumer app use, which will be free of charge as defined in the SDKs per calendar year, can be licensed under the Terms of Use.

125'000 transactions would amount to almost 42'000 calls to the *Truck Route API*. If the app is used every day in a year it would allow for about 115 API calls per day free of charge.

For apps using more than 125'000 transactions a year, Microsft refers to a *Bing Maps representative.*[7]

Application using more than 125,000 billable transactions per calendar year will need a user or transaction based Bing Maps license.

Fill out the Request for Quote form to have a Bing Maps representative assist you with your licensing options.

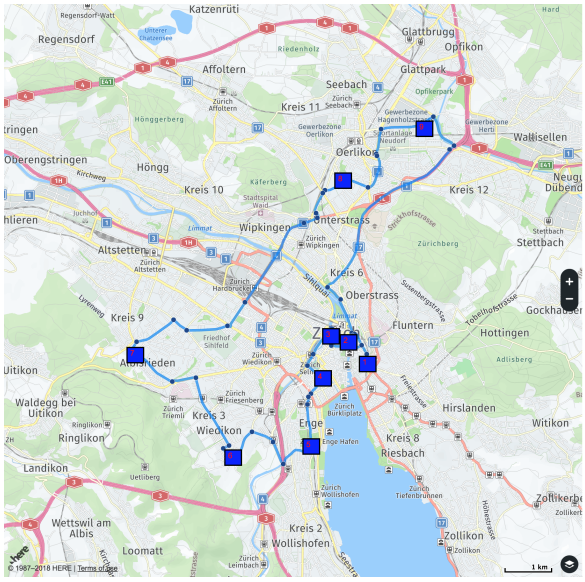
UI / UX Bing does not have own card material. Instead it uses a version of the Here MAP.

1.7.4 HERE Fleet Telematics Waypoints Sequence API

Technical limitations

- The maximum number of waypoints including the start point and end point is 120 without traffic and 50 with traffic.

Prototype This prototype uses the *HERE Fleet Telematics Waypoints Sequence API* to display an optimized route.



Hagenholzstrasse - Hirschengraben - Bahnhofquai - Gessnerallee Am Schanzengraben - Brunaustrasse - Frauentalweg - Hagenbuchrain - Ulmenweg - Hagenholzstrasse

1. Head northeast on Hagenholzstrasse. Go for 364 m.
2. Turn right onto Aubruggstrasse. Go for 762 m.
3. Turn right onto Wallisellenstrasse. Go for 120 m.
4. Take ramp onto A1L toward Chur/Luzern/Zürich-City. Go for 4.0 km.
5. Turn left onto Wasserwerkstrasse. Go for 363 m.
6. Turn right onto Neumühlequai. Go for 732 m.
7. Continue on Central. Go for 56 m.
8. Make a U-Turn onto Central. Go for 64 m.
9. Keep right onto Seilergraben toward Forch Rüti. Go for 493 m.
10. Continue on Hirschengraben. Go for 175 m.
11. Turn right onto Hirschengraben. Go for 14 m.
12. Turn right onto Hirschengraben. Go for 27 m.
13. Turn left onto Hirschengraben. Go for 21 m.
14. Arrive at Hirschengraben. Your destination is on the right.
15. Head toward Untere Zäune on Hirschengraben. Go for 86 m.
16. Turn left onto Untere Zäune. Go for 19 m.
17. Turn left onto Hirschengraben. Go for 85 m.
18. Turn left onto Hirschengraben. Go for 180 m.
19. Continue on Seilergraben. Go for 207 m.
20. Turn left onto Mühlegasse. Go for 240 m.
21. Continue on Uraniastrasse. Go for 20 m.
22. Continue on Rudolf-Brun-Brücke. Go for 65 m.
23. Continue on Uraniastrasse. Go for 12 m.
24. Turn right onto Bahnhofquai. Go for 125 m.
25. Arrive at Bahnhofquai. Your destination is on the left.

Pricing 1 million transaction for 449 Euros. After using this quota the sales team should be contacted.

One transaction is counted for each individual service feature included in a request. Broken down for a request to optimize 20 waypoints it would cost 0.009 Euro.

UI / UX The Here Map looks a bit dusty and outdated. But still able to do the work well.

1.7.5 Comparison

	Google	Bing	Here
Truck support	N	Y[1]	Y
Waypoint Limit	25+2	23+2	120 / 50[2]
Price during development	0-100 $	0	450 Euro
Price during production	Custom deals		
Core business	N	N	Y
UI / UX	1	3	2
API documentation	2	3	1
API customizability	1	1	1

Table 2: Comparison between the routing services
[1] Bing offers only routing without waypoint optimization for trucks.
[2] 50 with traffic; 120 without traffic.

1.7.6 Conclusion

Based on the comparison, the *HERE Fleet Telematics Waypoints Sequence API* clearly wins the evaluation. It is their core business and they provide the most features. The biggest downside are the costs which incur during development.

1.7.7 Decision

After carefully examining each result and discussing the results with our student advisor and industry partner we came to the result that the *Google Directions API* should be used based on the development costs.

2 Architecture

2.1 Deployment View

The system contains three parts: A frontend, *Truck Route Demo*, for demonstration, a backend, *Truck Route API* and the *Google Directions API*. In the following chapters they are explained in detail.

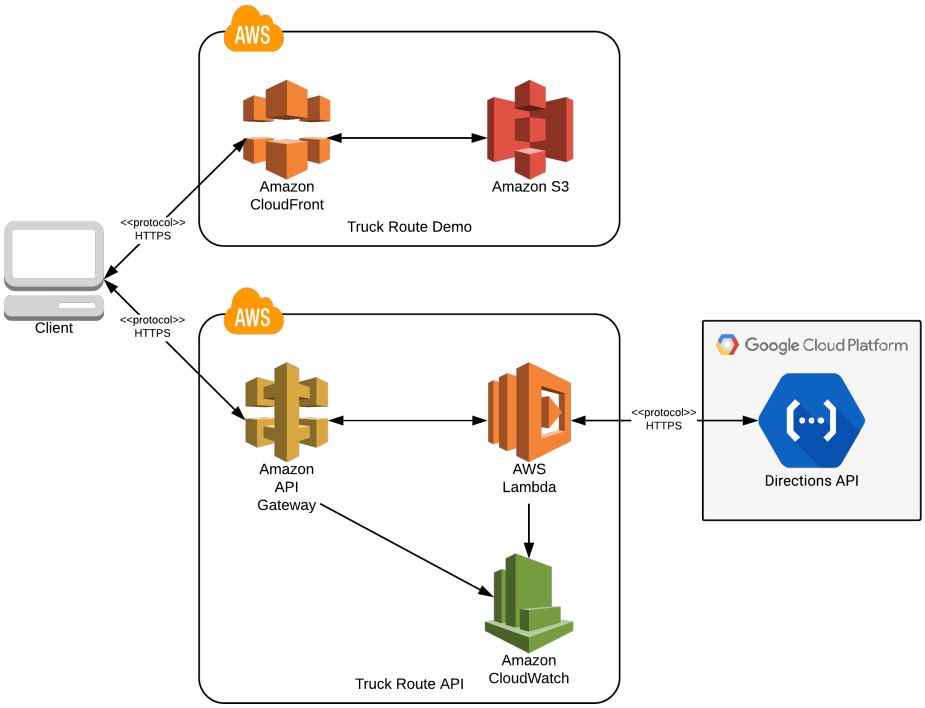


Figure 6: Overview over the entire cloud infrastructure.

2.1.1 Backend

The backend has two deployed instances one for developing and one for production. The backend itself is built with 3 *AWS* components as seen in Figure 6.

API Gateway The *API Gateway* acts as a "front door" for our business logic, handling the routing, monitoring and authorization of the API. The authorization is done via API key which can be configured per client. It also it handles the parsing of HTTP request and building of HTTP responses. For every request it triggers a lambda function with an event as parameter.

Lambda The *AWS Lambda* is a serverless computing platform where our business logic is running. It gets invoked by the *API Gateway* and calculates the routes. For a detailed view on the internal procedures see chapter 3.

CloudWatch *CloudWatch* is used by the API gateway and the lambda for performance monitoring and logging. Based on this data *CloudWatch* provides important insights which can be used to improve the quality of our infrastructure and software.

Google Directions API As described in chapter 1.7, the *Google Directions API* is used to calculate the routes based on our optimized clusters. Like our *AWS* infrastructure it is cloud-based. To be able to communicate with the *Google Directions API* an API key needs to be supplied as environment variable in the lambda function.

2.1.2 Frontend

The frontend infrastructure includes a storage (S3) of the React application and a corresponding distribution (CloudFront). Since the frontend is used for demonstration, only one instance is deployed.

CloudFront CloudFront is a content delivery network (CDN) that delivers the frontend with low latency and high transfer speeds form all over the world. Our CloudFront distribution origin is the frontend S3 bucket. It also allows us to use our own domains with SSL certificates, which were created with the *AWS Certificate Manager*.

S3 Simple Storage Service is an object storage service where the frontend gets deployed to. For every deployment the S3 frontend bucket gets overwritten and the new version is live.

2.2 Class Diagram

The class diagram is strongly influenced by the domain model. For this reason, the domain model can be used to help with naming ambiguities. Regarding the type information, these are the common TypeScript types used. The biggest differences are listed below:

- The material conversion, an utility class for the content type including mapping for type to conversion rate or attribute is added.
- The request which contains all the other classes is added.
- There is no recycling point or operation center, they are replaced by just their location.

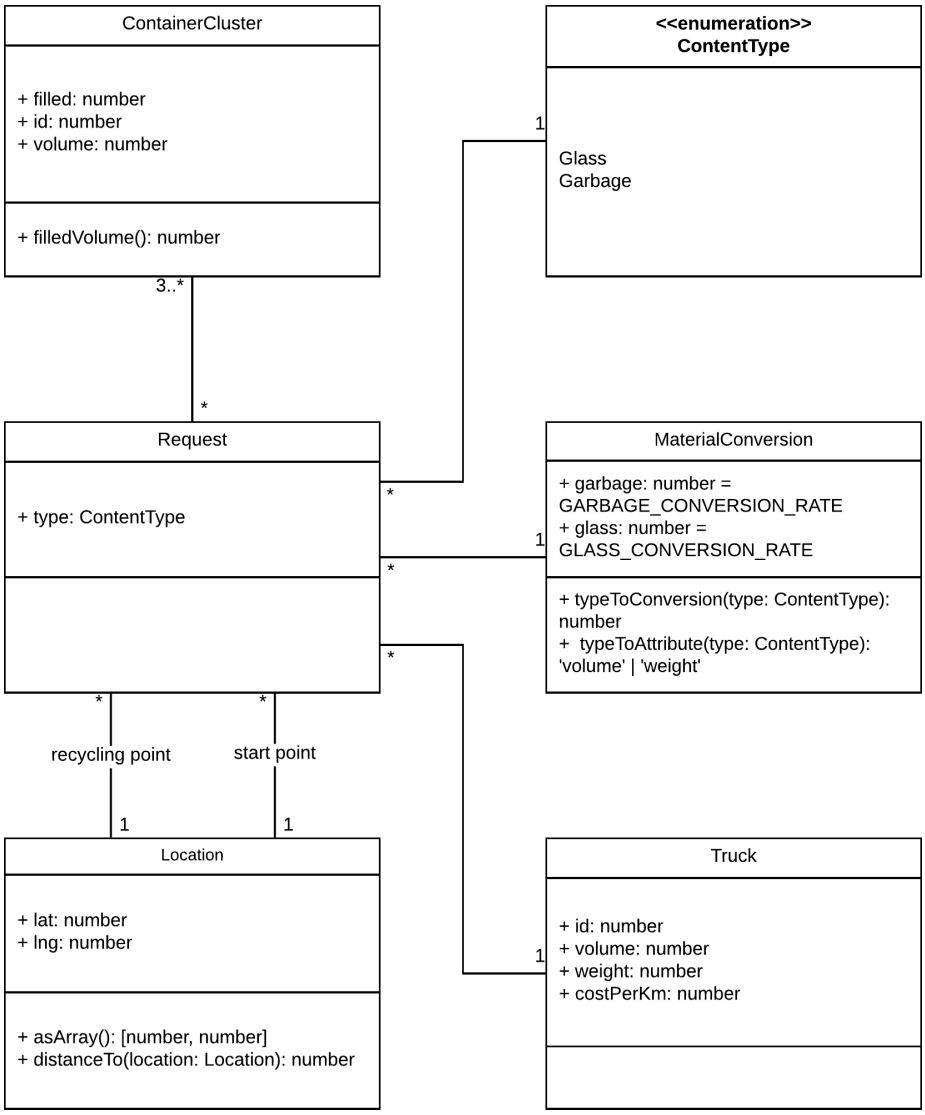


Figure 7: Typescript class diagram

Request object A request object looks like the following (class names are written out):

```
Request {
  containerClusters: [
    ContainerCluster { filled: 88, id: 1,  location: Location { lat: 47.3755475,  lng: 8.5417528   }, volume: 1 },
    ContainerCluster { filled: 57, id: 2,  location: Location { lat: 47.37668325, lng: 8.537054386 }, volume: 2 },
    ContainerCluster { filled: 82, id: 3,  location: Location { lat: 47.3716004,  lng: 8.546891672 }, volume: 1 },
    ContainerCluster { filled: 45, id: 4,  location: Location { lat: 47.3690001,  lng: 8.5349164   }, volume: 2 },
    ContainerCluster { filled: 51, id: 5,  location: Location { lat: 47.35653905, lng: 8.531746152 }, volume: 1 },
    ContainerCluster { filled: 90, id: 6,  location: Location { lat: 47.3597033,  lng: 8.5315136   }, volume: 4 },
    ContainerCluster { filled: 56, id: 7,  location: Location { lat: 47.3620391,  lng: 8.525885985 }, volume: 2 },
    ContainerCluster { filled: 34, id: 8,  location: Location { lat: 47.36882235, lng: 8.523452223 }, volume: 3 },
    ContainerCluster { filled: 62, id: 9,  location: Location { lat: 47.37278125, lng: 8.511842391 }, volume: 3 },
    ContainerCluster { filled: 68, id: 10, location: Location { lat: 47.3759083,  lng: 8.512865957 }, volume: 1 }
  ],
  recyclingPoint:     Location { lat: 47.4145355, lng: 8.5621876 },
  startPoint:         Location { lat: 47.386369, lng: 8.5364623 },
  truck:              Truck { id: 759753, volume: 19, weight: 16, costPerKm: 7 },
  materialConversion: MaterialConversion { garbage: 0.1, glass: 1.2 },
  contentType:        'Glas'
}
```

2.3 Solution Strategy

2.3.1 Evaluation of Backend Technology

AWS Lambda *Live Track AG* already uses *AWS* infrastructure. Furthermore, an *AWS Lambda* function has the following benefits:

- no provisioning or managing servers
- includes code monitoring and logging
- code only runs when needed
- stateless (easy to scale and test)
- scales automatically

TypeScript TypeScript is a strict syntactical superset of JavaScript, and adds optional static typing to the language. To run, it is first compiled to JavaScript. During the compilation the compiler type checks the program.

- *In connection with* the programming language on the server
- *considering* past experience
- *we have decided to* use TypeScript
- *and against* the use of any other language
- *to* be able to start and develop quickly
- *for this we accept as consequence* to miss a chance to learn a new programming language

Node.js Node.js enables us to run the generated JavaScript on a server. It is built on top of the V8 JavaScript engine, which is used to run JavaScript in Chrome. It is open-source and we both have already gained some experience with it.

- *In connection with* the server runtime
- *considering* few alternatives
- *we made a decision for* Node.js
- *and against* less established runtimes like deno[8]
- *to* be future-proof
- *because of this we accept as consequence* the need of compiling TypeScript to JavaScript

2.3.2 Evaluation of Frontend Technology

React React is a JavaScript frontend library for building user interfaces.

- *In connection with* the implementation of the frontend
- *considering* the interactive UI
- *we made a decision for* React
- *and against* Vue.js and Angular
- *to* use a lightweight *and* industry established library
- *because of this we accept as consequence* that the frontend will not be compatible with the prototype of *Live Track AG*, which is built using Angular

2.3.3 Continuous Integration

Continuous integration (CI) and continuous deployment (CD) are set up. For both the frontend and the backend Travis[9], a continuous integration SaaS, is used. Both application share a similar continuous integration flow, based on the same technology.

Prerequisite Our build setup has some prerequisites which have to be installed on the CI. The required software is listed below, including the minimum version required.

- nvm[10] (Node.js version manager) 0.31
- Yarn[11] (package manager) 1.10
- Git[12] (version control system) 2.12

Additionally required software is installed automatically using Yarn.

Procedure backend The API has a straight forward procedure, broken down into single logical steps. (Multiple steps can occur on the same infrastructure.)

1. We developers commit our code into the version control system provider *GitHub*.
2. For every commit a web-hook is triggered which starts the CI on *Travis*.
3. On *Travis* the tests are executed and the code is linted.
4. If the checks have been successfully and the branch has a server, the deployment is started.
5. For the API we first build the application and the deploy it via serverless framework into the *AWS ecosystem*. The serverless framework uses the *AWS CloudFormation* under the hood.
6. Once deployed the application runs as a *AWS Lambda function*.

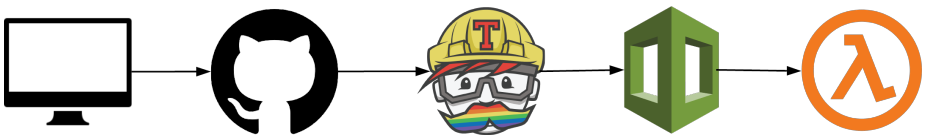


Figure 8: Procedure of the CI for the API.

Procedure frontend The frontend shares the first steps with the backend, just the deployment is different. Instead of deploying via serverless, the Amazon SDK is used directly and upload the deployment to an S3 bucket.

2.3.4 Testing

To ensure the fulfilling of the functional and non functional requirements we wrote unit and End-to-End (E2E) tests using Jest[13], an open source testing framework by *Facebook*.

Test execution The tests are platform independent and currently the execution is guaranteed on macOS, Ubuntu and Windows. If the E2E test should be executed, the API key for the *Google Directions API* has to be provided via environment variable.

Test Execution

```
truck-route git:(master) yarn test:w:e2e
yarn run v1.10.1
$ cross-env E2E=true yarn test
$ jest --forceExit --coverage --verbose --detectOpenHandles
 PASS  tests/helpers/routingAPI.test.ts
  callRoutingAPI
    ✓ returns a route (11ms)

 PASS  tests/services/clusterFilter/greedyClusterFilter.test.ts
  GreedyClusterFilter
    request with big truck and type glass
      prioritySelectionStrategyByFillingLevel
        ✓ loaded weight is below truck weight (8ms)
        ✓ selects a greedy solution of clusters (3ms)
      prioritySelectionStrategyByFilledVolume
        ✓ loaded weight is below truck weight (2ms)
        ✓ selects a greedy solution of clusters (1ms)
      prioritySelectionStrategyByFullestClusterNearestLocation
        ✓ loaded weight is below truck weight (6ms)
        ✓ selects a greedy solution of clusters (12ms)
    request with small truck and type garbage
      prioritySelectionStrategyByFillingLevel
        ✓ loaded weight is below truck weight (2ms)
        ✓ selects a greedy solution of clusters (2ms)
      prioritySelectionStrategyByFilledVolume
        ✓ loaded weight is below truck weight (2ms)
        ✓ selects a greedy solution of clusters (5ms)
      prioritySelectionStrategyByFullestClusterNearestLocation
        ✓ loaded weight is below truck weight (4ms)
        ✓ selects a greedy solution of clusters (14ms)

 PASS  tests/services/routingApi.test.ts
  RoutingAPI
    ✓ processes the routing api result correct (72ms)
    ✓ processes the routing api result correct even if no routes are returned (10ms)
    ✓ calls the api with the correct params (13ms)

 PASS  tests/models/location.test.ts
  Location
    ✓ returns no error for a valid location (258ms)
    ✓ returns an error for a invalid location (value not a number) (3ms)
    ✓ returns an error for a invalid location (value not present) (1ms)
    ✓ returns an array with lat and lng (3ms)
    ✓ returns the distance between this location and given one (2ms)

 PASS  tests/models/request.test.ts
  Request
    ✓ returns no error for a valid request (without  materialConversion) (255ms)
    ✓ returns no error for a valid request (with materialConversion) (1ms)
    ✓ returns an error for a invalid request (wrong truck) (7ms)
    ✓ returns an error for a invalid request (wrong contentType) (2ms)
    ✓ returns an error for a invalid request (wrong containerClusters) (1ms)
    ✓ returns an error for a invalid request (empty containerClusters) (2ms)
    ✓ returns an error for a invalid request (2ms)

 PASS  tests/services/responseBuilder.test.ts
  ResponseBuilder
    ✓ wrap the provided body into an aas lambda processable way (2ms)

 PASS  tests/lambda.test.ts
  Lambda
    ✓ runs trough the full stack with nothing mocked (valid request) (682ms)
    ✓ runs trough the full stack with nothing mocked (route start point in china) (226ms)
    ✓ runs trough the full stack with nothing mocked (invalid request values) (17ms)
    ✓ runs trough the full stack with nothing mocked (malformed request JSON)

 PASS  tests/models/truck.test.ts
  Truck
    ✓ returns no error for a valid truck (273ms)
    ✓ returns an error for a invalid truck (1ms)

 PASS  tests/services/requestParser.test.ts
  RequestParser
    ✓ parses an api gateway proxy event and provides the result (21ms)
    ✓ can handle invalid JSON (1ms)

 PASS  tests/helpers/clusterFilter.test.ts
  selectBaseCluster
    ✓ selects the cluster with the highest filling (7ms)

 PASS  tests/models/containerCluster.test.ts
  ContainerCluster
    ✓ returns no error for a valid containerCluster (298ms)
    ✓ returns an error for a invalid containerCluster (invalid values) (5ms)
    ✓ returns an error for a invalid containerCluster (invalid location)
    ✓ returns an error for a invalid containerCluster (empty) (1ms)
    ✓ calculates the filled volume correct (1ms)

 PASS  tests/helpers/geoUtils.test.ts
  distanceBetween
    ✓ calculates the distance between two locations (6ms)

 PASS  tests/services/clusterFilter/knapsackClusterFilter.test.ts
  KnapsackClusterFilter
    request with big truck and type glass
      ✓ selects the correct base cluster (8ms)
      ✓ loaded weight is below truck weight (4ms)
      ✓ selects optimal solution of clusters (6ms)
    request with medium truck and type garbage
      ✓ selects the correct base cluster (5ms)
      ✓ loaded weight is below truck weight (3ms)
      ✓ selects optimal solution of clusters (5ms)

 PASS  tests/helpers/knapsack.test.ts
  knapsack
    ✓ works for test data set with index 0 (2ms)
    ✓ works for test data set with index 1
    ✓ works for test data set with index 2
    ✓ works for test data set with index 3 (2ms)
    ✓ works for test data set with index 4
    ✓ works for test data set with index 5 (1ms)

 PASS  tests/models/materialConversion.test.ts
  MaterialConversion
    ✓ returns no error for a valid materialConversion (251ms)
    ✓ returns an error for a invalid materialConversion (value not a number) (3ms)
    ✓ returns no error for a valid empty materialConversion (2ms)
    ✓ returns no error for a valid materialConversion with only one value
  utils
    typeToConversion
      ✓ returns the correct conversion for garbage
      ✓ returns the correct conversion for glass
    typeToAttribute
      ✓ returns the correct attribute for garbage (1ms)
      ✓ returns the correct attribute for glass
```

Test coverage Our tests are covering 100% of the application code. Even if a high coverage does not ensure a bug free software, we know that all the code and code paths have been executed at least once during tests.

Coverage

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Line #s
All files	100	100	100	100	
src	100	100	100	100	
lambda.ts	100	100	100	100	
truckRouteConstants.ts	100	100	100	100	
src/helpers	100	100	100	100	
clusterFilter.ts	100	100	100	100	
errorMessages.ts	100	100	100	100	
geoUtils.ts	100	100	100	100	
knapsack.ts	100	100	100	100	
routingAPI.ts	100	100	100	100	
src/models	100	100	100	100	
containerCluster.ts	100	100	100	100	
contentType.ts	100	100	100	100	
location.ts	100	100	100	100	
materialConversion.ts	100	100	100	100	
request.ts	100	100	100	100	
truck.ts	100	100	100	100	
src/services	100	100	100	100	
requestParser.ts	100	100	100	100	
responseBuilder.ts	100	100	100	100	
routingApi.ts	100	100	100	100	
src/services/clusterFilter	100	100	100	100	
clusterFilter.ts	100	100	100	100	
greedyClusterFilter.ts	100	100	100	100	
knapsackClusterFilter.ts	100	100	100	100	
src/services/prioritySelectionStrategies	100	100	100	100	
byFilledVolume.ts	100	100	100	100	
byFillingLevel.ts	100	100	100	100	
byFullestClusterNearestLocation.ts	100	100	100	100	

Test Suites: 15 passed, 15 total
Tests: 64 passed, 64 total
Snapshots: 0 total
Time: 4.321s

Manual tests Manual Tests were done via *Swagger* and the frontend. All use cases were tested, including corner cases:

Nr	Name	Satisfied
1	Non authorized request	Yes
2	Non parsable request	Yes
3	Empty request	Yes
4	Invalid data request (partial missing data)	Yes
5	Invalid data request (wrong data type)	Yes
6	Invalid data request (not enough clusters)	Yes
7	Invalid logical data request (clusters in China and Australia)	Yes
8	Valid request	Yes

2.3.5 API Documentation - Swagger

The API is documented with *Swagger* following the OpenAPI[14] specification 3. Based on the *Swagger* file it is possible to simulate API requests and validate request and response against the development API of Truck Route as seen in Figure 9. The swagger.json file is located in our git repository.

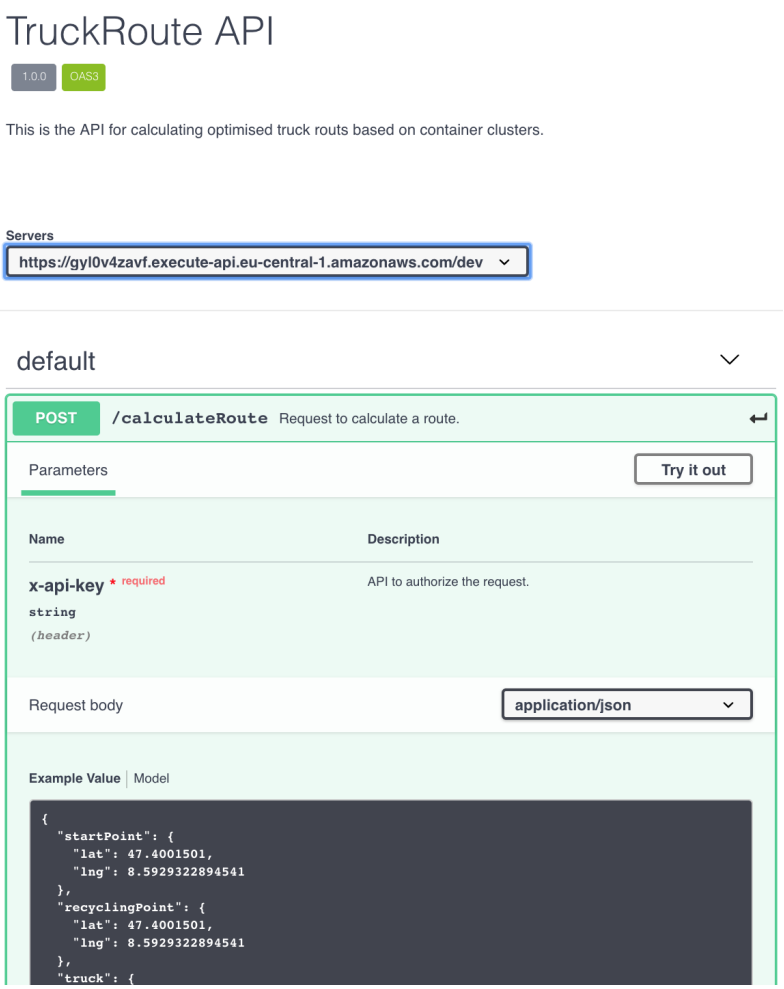
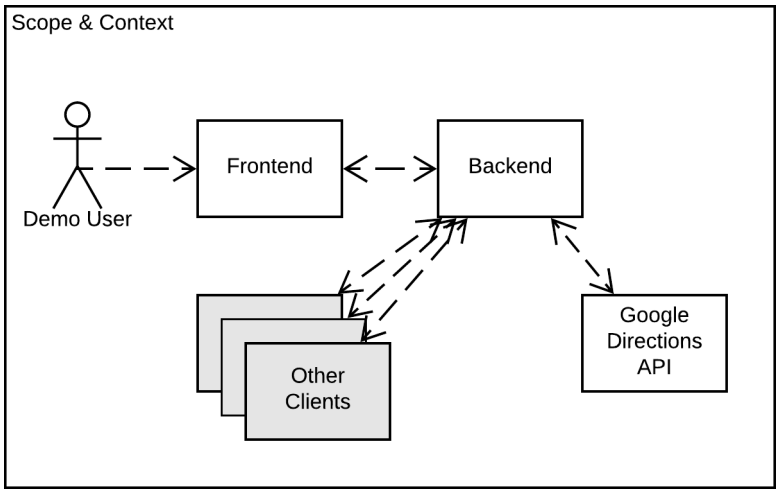


Figure 9: Screenshot of the *Swagger* interface based on the documentation. When the API key is specified, it is possible to send the request with the "Try it out" option.

3 Building Block View

3.1 Whitebox Overall System



Demo User Someone using the frontend.

Frontend A user can use the frontend. The frontend requests and receives a route from the backend.

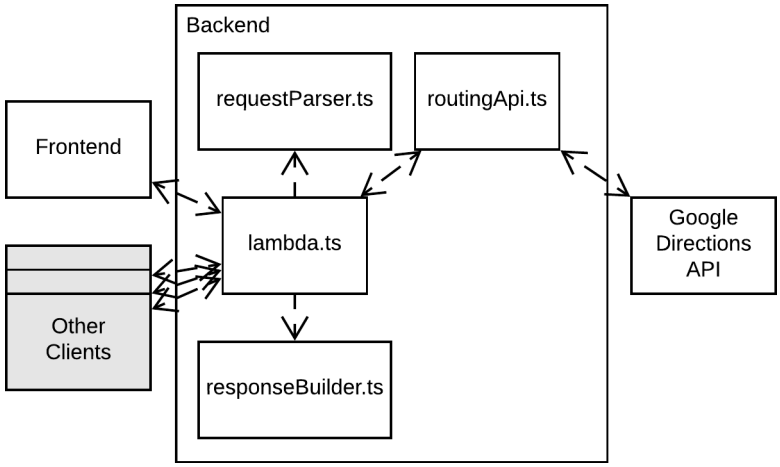
Other clients Other clients can also request a route, this is how Truck Route will be used in the future.

Backend The backend server returns optimized routes based on the data received. For the route generation itself, *Google Directions API* is used. For a more exhaustive explanation of the process on the server refer to section 4.1.

Google Directions API *Google Directions API* is used to generate a route based on waypoints.

3.2 Building Blocks – Level 1

3.2.1 Backend



lambda.ts The lambda (named after the *AWS Lambda compute service*) is one function which coordinates:

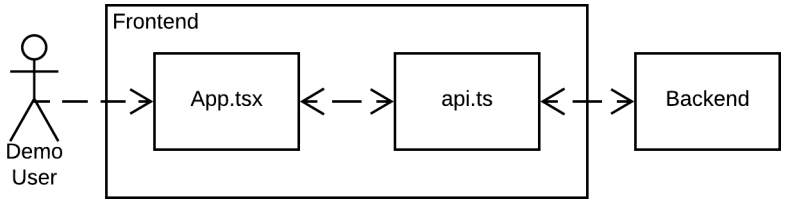
1. The parsing of the request
2. The validating of the request
3. The processing of the request (calculation of routes)
4. The building and returning of a response with the result or an error message

requestParser.ts The request parser is used to parse and validate if the request is parsable.

routingApi.ts The routing API is responsible for the communication with the *Google Directions API*. It also builds solutions from the routes which are returned by the *Google Directions API*. The solutions contains the request, the response and key figures

responseBuilder.ts The response builder returns an object which is processes able by the *AWS API Gateway*, based on a body object and a status code.

3.2.2 Frontend

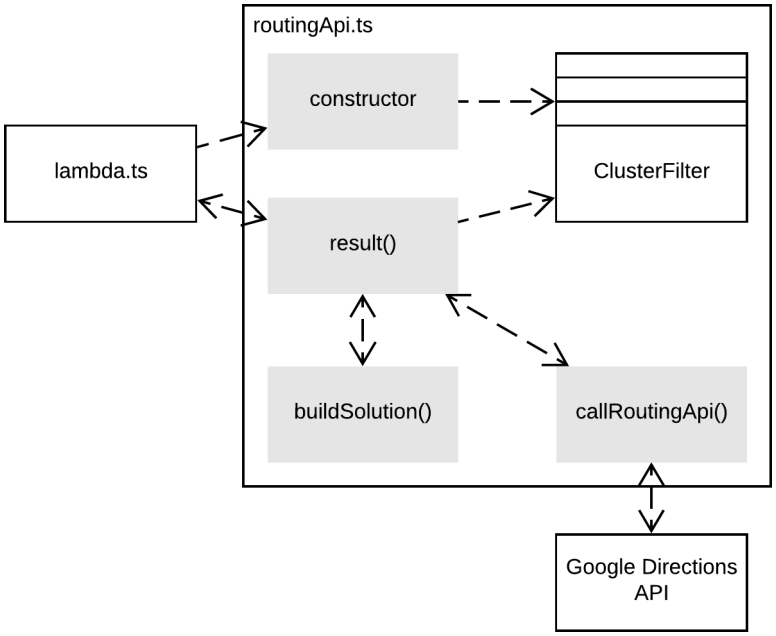


App.tsx This is the main React component. It receives input from the user and displays the selector and, if there are any, routes.

api.ts When the user triggers a request to the backend it is handled with the api.ts. api.ts extracts the request parameters from the selected data and fetches the results from the backend. Afterwards it parses and returns the parsed results.

3.3 Building Blocks – Level 2

3.3.1 Backend / Routing API



As a whole, the routing API class is responsible to encapsulate the *Google Directions API*. This allows for a complete replacement of the *Google Directions API* by another routing service.

constructor The constructor generates the cluster filters.

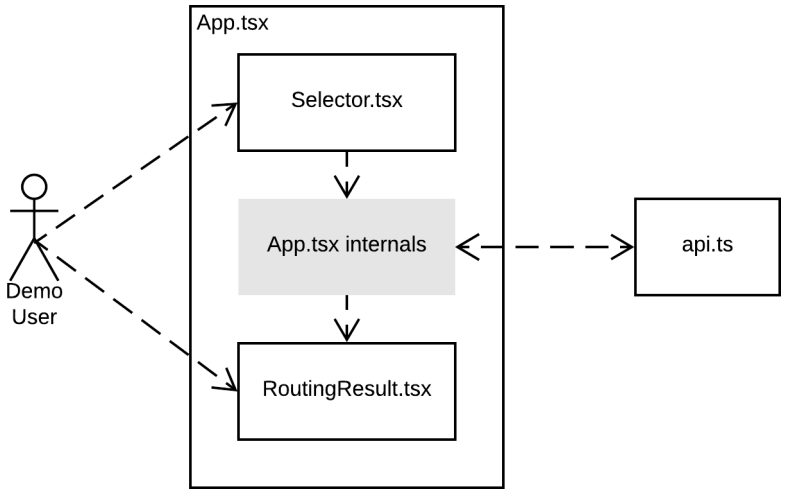
ClusterFilter Cluster filters are used to get a subset of clusters, basically deciding which container clusters should be emptied.

result() The result function uses the subsets of clusters created by the cluster filters and calls the *Google Directions API* for each of them. It then builds solution objects using *buildSolution()* and returns them.

callRoutingApi() This helper function uses the *Google Maps* node.js client[15] via another function which handles the authorization, to fetch a route from the *Google Directions API*.

buildSolution() The buildSolution function calculates key figures from the request objects and the response of *Google Directions API*. It returns an object containing request, response and key figures.

3.3.2 Frontend / App.tsx



Selector.tsx This React component enables the user to select:

- Container clusters, from which Truck Route may select a subset from.
- A truck, which defines the starting point of the route and how much volume and how many kilograms can be carried.
- A content type, at the moment either glass or garbage. Based on the content type, a recycling station is selected.

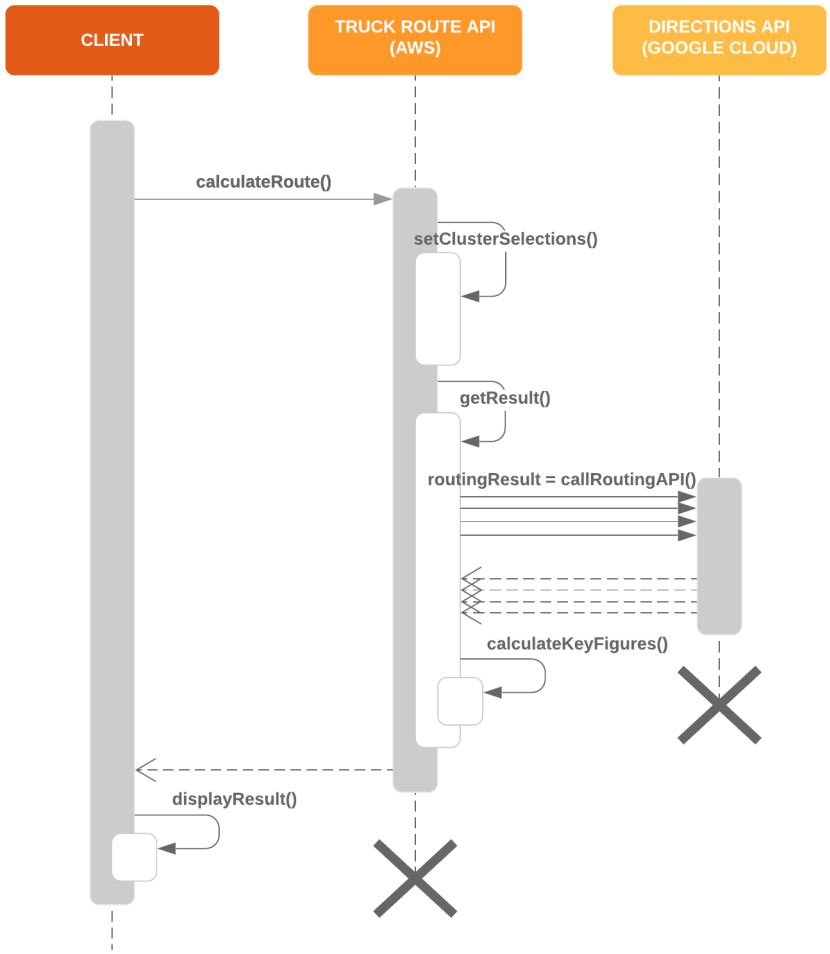
App.tsx internals The main component has a handler *onCalculateRoute(...)*. This handler is executed by the selector when the button *Calculate Route* is clicked. The handler then calls the backend over api.ts and stores the result in its state. For each result it renders a RoutingResult component.

RoutingResult.tsx This component renders a routing result the user wanted to calculate. The user can switch between multiple algorithms/stategies and compare the algorithm results.

4 Runtime View

4.1 Scenario: Get Route and Display It in the Frontend

This is the main scenario which was implemented as part of this thesis. There are three actors: The frontend, the backend on AWS and *Google Directions API*. The diagram shows the communication and internal calls of the frontend and backend. For simplicity and better understanding some calls have been renamed.



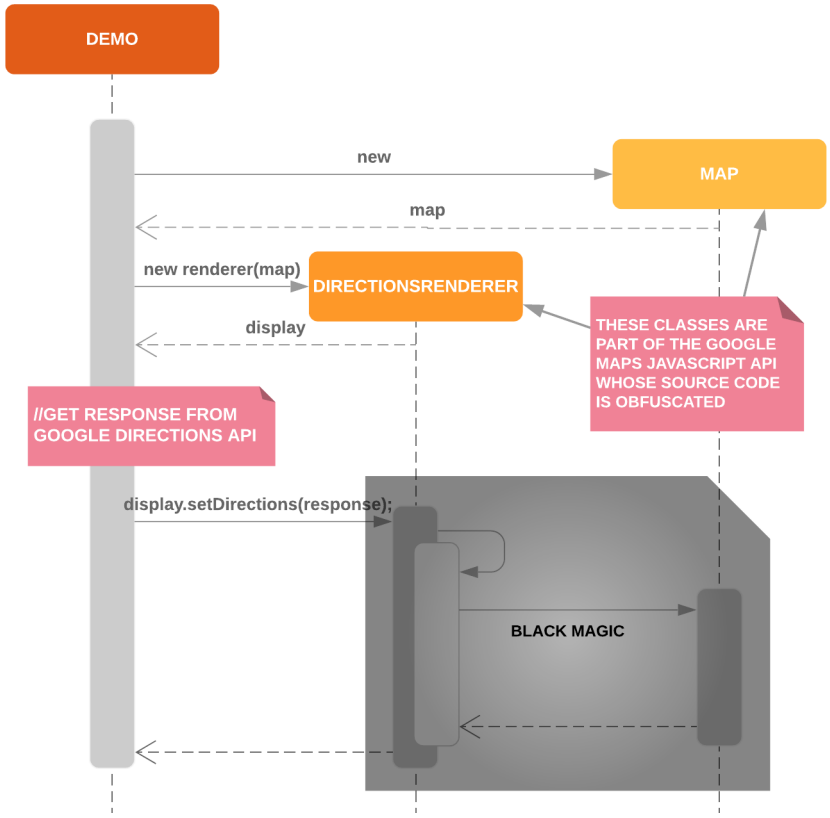
4.2 Scenario: Visualize Received Route in Google Maps

While trying to visualize the routes we received from the backend we hit an unexpected roadblock. Even though the route was passed to the React Google Maps Component[16], it was not displayed.

To understand why, a comparison with the *Google Directions API* prototype was made. We discovered that the browser SDK and the Node.js SDK for *Google Maps* are not compatible. This was confirmed in the response to a *GitHub* issue. Statement of the core developer Stephen McDonald[17]:

> The front-end and back-end services aren't intended to work together in the way you're trying to do.
>
> At a guess you might be able to manually patch the back-end route data with the missing attributes that the front-end API expects.

Based on his guess that we might be able to patch it manually we reverse engineered the frontend API. For a better understanding, the flow used in the reverse engineering part is shown in the following scenario:



To request and display a route with *Google Directions API* the *Google Maps JavaScript API client library*[18] was used. The code of this client library is

heavily obfuscated, which means it is generally hard to understand what it does. Nevertheless we made some discoveries through debugging and reverse-engineering. We found out that new properties are added to the result object before displaying it. To fix the visualization of the route, we transformed the result object of our API to fit the expectations of the client library. The code to transform the result can be found in the Figure 10.

```
import * as googleMaps from '@google/maps';

export default function getDirectionRoute(
  directionResult: googleMaps.DirectionsResponse
): google.maps.DirectionsResult & { request: { travelMode: string } } {
  return {
    ...directionResult,
    routes: directionResult.routes.map((route: any) => ({
      ...route,
      bounds: new google.maps.LatLngBounds(
        route.bounds.southwest,
        route.bounds.northeast
      ),
      overview_path: google.maps.geometry.encoding.decodePath(
        route.overview_polyline.points
      ),
      legs: transformLegs(route)
    })),
    request: { travelMode: 'DRIVING' }
  };
}

function transformLegs(route: any) {
  return route.legs.map((leg: any) => {
    leg.start_location = new google.maps.LatLng(
      leg.start_location.lat,
      leg.start_location.lng
    );
    leg.end_location = new google.maps.LatLng(
      leg.end_location.lat,
      leg.end_location.lng
    );
    leg.steps = leg.steps.map((step: any) => {
      step.path = google.maps.geometry.encoding.decodePath(
        step.polyline.points
      );
      step.start_location = new google.maps.LatLng(
        step.start_location.lat,
        step.start_location.lng
      );
      step.end_location = new google.maps.LatLng(
        step.end_location.lat,
        step.end_location.lng
      );
      return step;
    });
    return leg;
  });
}
```

Figure 10: We see in the code the decoding of a polyline[19] encoded string and the creation of location objects to satisfy the *Google Maps JavaScript* client library.

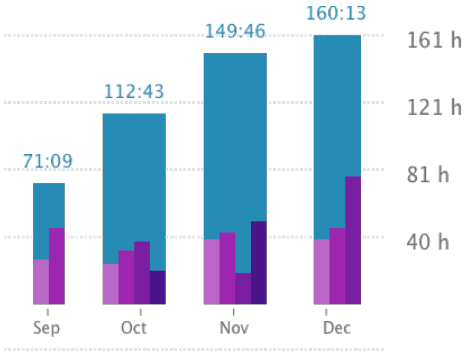
5 Project Management

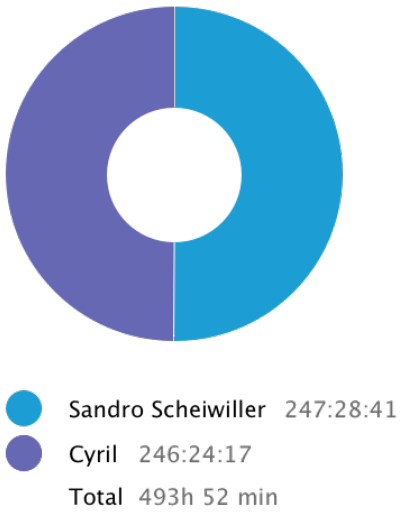
5.1 Milestones

Because we were working agile, we mostly did without milestones, expect a few which were set by the school.

Number	Description	Date planned	Date is
1	Kickoff	17.09.2018	18.09.2018
2	Abstract delivery	18.12.2018	18.12.2018
3	Project delivery	21.12.2018	21.12.2018

5.2 Time Evaluation





5.3 Quality Measures

This chapter defines the measures and instruments used to ensure the quality of the project.

5.3.1 Documentation

The individual documents are stored in a shared *OneDrive* folder "SA". The benefits are that *OneDrive* has a document modification history to ensure that no documents are lost. Furthermore the main documentation file of this project is written in an online Latex editor called *Overleaf*[20]. *OverLeaf* has also a modification history. Writing the documentation in Latex ensures a high quality and a consistent layout.

5.3.2 Project Management Workflow

We use scrum to manage the project and as project management software we use *Zube*[21]. A lightweight and flexible software which is provided as a service. We established our own workflow as seen in Figure 11.

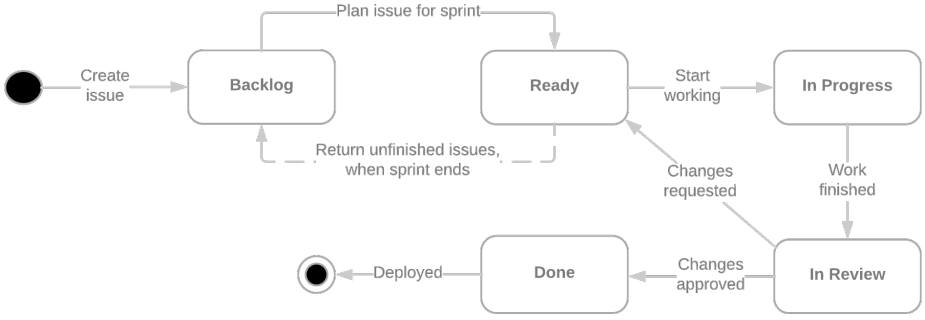


Figure 11: Our Project Management Workflow.

5.3.3 Development

The source code is hosted on *GitHub*:

- Demo: https://github.com/cyrilkyburz/truck-route-frontend
- API: https://github.com/cyrilkyburz/truck-route

Workflow We use our own version of *GitFlow*[22] which is a subset of it. Namely we use the *Feature Branch Workflow*. We do not have hotfixes since the project is not in production and we use the release system integrated in *GitHub* combined with semantic versioning[23].

5.3.4 Definition of Done (DoD)

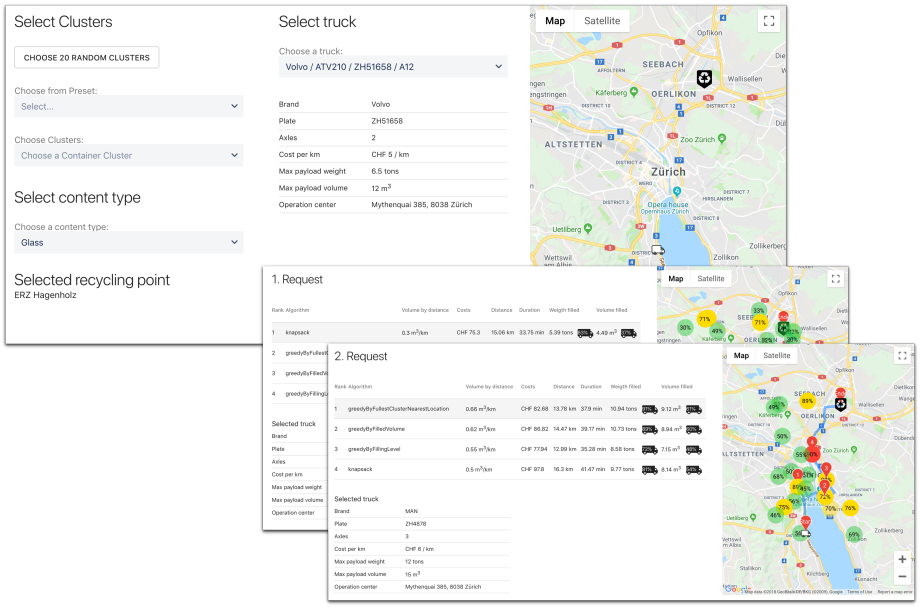
We use our own DoD to reach our desired quality. The DoD is a common understanding, when something is done.

The following points have to be fulfilled:

- Our code is managed in a git repository. For features code reviews are done when the code wants to join the develop or master branch.
- 4 eyes principle for critical code and infrastructure.
- Dependencies and libraries are evaluated to be stable and future-proof.
- 100% test Coverage for the API
- Our code is linted and we have the same style of writing the code.
- We use TypeScript the proper way and try not to use any or generic types.
- Our git repositories have working build pipeline for macOS, Windows and Ubuntu.
- Our Continuous Integration is always green for develop and master branch. This means the project builds, tests, lintings and deployments are working.
- Our git repositories have meaningful "ReadMe" files explaining the local setup and the build pipeline.

6 Results

The result of this thesis are two applications both live and released. The Truck Route Demo features a powerful user interface for showcasing our Truck Route API. The API is production ready and able to handle up to 4000 requests per second. For every request multiple optimized routes are calculated and the requester can decide which one fits his needs the best. The outcome of both applications combined can be seen below.



6.1 Implementation of Functional Requirements

All use cases were implemented:

- UC 01: Calculate route
- UC 02: Select and view filter category
- UC 03: Get route
- UC 04: View Route

6.2 Implementation of Non-Functional Requirements

All non-functional requirements are fulfilled. Special attention went to performance and scalability.

6.2.1 Performance

The API speed we measured varies in a range between 200ms and 800ms as seen in the Figure 12. The average time is 250ms. The goal was to achieve an API speed of less than 4s, and we succeeded, as our measurements clearly show.

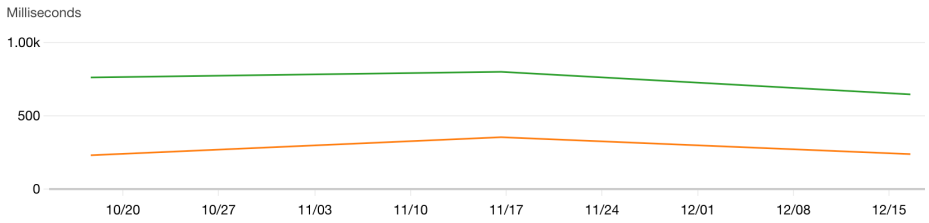


Figure 12: Performance measures from the *AWS CloudWatch*. The yellow line shows the average time and the green line the worst case.

6.2.2 Scalability

The *AWS Lambda function* allows us to handle up to 1000 concurrent API requests. Based on the average time used by our API we should be able to handle 4000 request per second if needed.

6.3 Cloud Pricing Calculator

To calculate how the costs of running the application would be influenced by the amount of requests we made a custom excel sheet. As seen in Figure 13, the user can input multiple values and gets displayed a bill.

We found out that most costs occur using the *Google Directions API*.

6.4 Outlook

Live Track AG is currently developing a prototype, part of which is going to be the *Truck Route API*. Based on the feedback to the prototype *Live Track AG* will then decide further actions.

If the decisions falls to move forward using Truck Route, there are several features that could be needed. An easy to implement feature would be the support for all types of recycling containers such as aluminum. A more challenging feature would be to calculate multiple routes for an entire day.

6.5 Acknowledgments

We would like to thank our supervisor, Mirko Stocker, for his continued support throughout the thesis. The weekly meetings with him were invaluable in keeping us on track.

We also thank our industry partner, Bernhard Zindel, who trusted us to experiment with new solutions and was always there for questions.

Truck Route Calculation Sheet

Program information	
Number of calculations / month	4000
Calculations in millions (rounded up)	0
Calculations in millions (rounded down)	1
Average response size (KB)	62.5
Total response data transfer (GB)	244.140625
Average request size to Directions API (KB)	2.5
Number of algorithms	4
Each algorithm more means an additional request to the Google Directions API.	
Total request data transfer (GB)	39.0625

AWS Lambda	
Average time billed (ms)	400
Memory used (MB)	256
Price per 1s	0.00000417
Total compute time (s)	1600
Total compute (GB-s)	400
Free tier compute time	*400000*
Billable compute (GB-s)	0
Compute costs	$ -
GB used for requests to Google	40
Data transfer costs	$ 3.51

Bill		
Google Directions API	$	160.00
AWS		
AWS Lambda		
Requests	$	-
Computing time	$	-
Data Transfer	$	3.51
AWS API Gateway		
Requests	$	3.70
Data Transfer	$	21.96
AWS CloudWatch	$	-
Total	$	189.17
per request	$	0.05

AWS Gateway	
Price / M requests	$ 3.70
Price for requests	$ 3.70
GB used for response	245
Data transfer cost	$ 21.96

Google Directions API			
Pricing:	*SKU: Directions Advanced*		
Number of queries			**Price ($)**
1	*to*	*100000*	*0.01*
100001	*to*	*500000*	*0.008*
16000			$ 160.00

Figure 13: The cloud calculation sheet. The user can input the values in the cells with an orange background. The values with the grey background and the bill get updated according to the input.

7 Attachments

We have included the following documents as attachments:

Personal Reports Personal reports of the thesis authors

Glossary Common vocabulary of this thesis

Task Definition The task definition we got at the start of the thesis

Decisions The decisions document which we updated with new decisions during the entire thesis

Copyright and Usage Agreement The agreement to give our industry partner the right to use our code

7.1 Personal Reports

7.1.1 Cyril Kyburz

For my student research project thesis, it was particularly important to me that:

- it was a challenge
- it had an industrial partner
- it is needed in the professional world

The cooperation with our industrial partner Bernhard Zindel and our supervisor Mirko Stocker harmonized from the very beginning. The discussion of ideas and solutions in the weekly meetings were the basis for a challenging and at the same time enjoyable thesis.

We had the free choice of technology and were able to use our preferred programming language and framework to get the work done. Namely TypeScript and React. It quickly became clear that Sandro would be responsible for the frontend and I for the backend. But with an improvement compared to our engineering project, where we completely split the work. We took care that either of us developed for both applications and stayed up to date. That allowed me to learn the most from both worlds. Resulting in learning new libraries (for example: *Google Maps SDK*) and improving existing skills with new techniques like advanced TypeScript syntax and React features.

With the chosen technologies we were able to implement a, in my opinion, good solution which is able to optimize routes. In particular, the architecture with the stateless AWS Lambda function appeals to me. The clear flow makes it is easy to understand and test. I would definitely choose this architecture again.

In summary I can say that this thesis is a full success and that I am happy to be on the team with Sandro. I am for sure taking a lot of good memories with me.

7.1.2 Sandro Scheiwiller

For me personally, it is very important that my projects are useful and that I can develop my skills. This project has delivered both: Whether *Live Track AG* directly uses the project or not, they have gained a lot of valuable information. And I personally learned a lot, both technology related and project related.

From the beginning I felt very confident in how the project was going to turn out. The meetings were productive and our advisor and our industry partner gave us clear goals we could work to meet. These goals were very stable, which saved us from having to throw away unused work. Because of this I later realized the value of having an extensive analysis phase.

I usually do put my focus on the result, not on the technologies used, but it can make an real difference regarding developer experience. In a (meanwhile faraway) past I used to discount the value of adding a typing system to

JavaScript, but meanwhile I would certainly miss it, even in smaller projects. The integrated development experience is just so much better. So I was very happy using TypeScript. A welcome surprise was the addition of TypeScript support from the Facebook team to the React app base kit, which made adding TypeScript support to the frontend much easier than before.

Here I also want to thank Cyril for setting up continuous integration, testing, deployments and everything else that enabled us to put our focus on coding.

Overall I am very happy with the final result. Cyril and I have accomplished a lot and, as far as I am concerned, avoided both over-engineering (too much generalization) and writing spaghetti code (not enough generalization).

7.2 Glossary

API or Backend If not special mentioned, API stands for our developed backend application

AWS Amazons cloud

(AWS) API Gateway An AWS service that enables the maintaining, monitoring and securing of the backend

(AWS) CloudFront Content delivery network (CDN)

(AWS) CloudWatch Service for performance monitoring and logging

(AWS) Lambda The AWS Lambda is a serverless computing platform where ourbusiness logic is running

(AWS) S3 Simple Storage Service is an object storage servic

Container Cluster Multiple containers combined

Container Cluster Subset Selected clusters ready for the Google Directions API

Frontend If not special mentioned, frontend stands for our developed frontend application

Google Directions API Routing service from Google solving the traveling salesman problem

UI User Interface

UX User Experience

References

[1] Inc. Amazon Web Services. *Serverless Applications Lens*. 2017. URL: https://d1.awsstatic.com/whitepapers/architecture/AWS-Serverless-Applications-Lens.pdf.

[2] Google. *Developer Guide*. URL: https://developers.google.com/maps/documentation/directions/intro.

[3] Microsoft. *Calculate a Truck Route*. URL: https://msdn.microsoft.com/en-us/library/mt814923.aspx.

[4] Google Directions API users and developers. *Ability to influence Directions (e.g. "avoid" / "roadblock")*. 2008. URL: https://issuetracker.google.com/issues/35816642.

[5] Google. *SKU: Directions Advanced*. URL: https://developers.google.com/maps/documentation/directions/usage-and-billing#directions-advanced.

[6] Microsoft. *Truck Routing API*. URL: https://www.microsoft.com/en-us/maps/truck-routing.

[7] Microsoft. *Help Choosing the Right License*. URL: https://www.microsoft.com/en-us/maps/licensing/options.

[8] Ryan Dahl and other contributors. *deno. A secure TypeScript runtime built on V8*. URL: https://github.com/denoland/deno.

[9] Travis. *travis. Continuous integration service*. URL: https://travis-ci.org/.

[10] Tim Caswell. *nvm. Node version manager*. URL: https://github.com/creationix/nvm.

[11] Facebook. *yarn. Package manager for Node.js*. URL: https://yarnpkg.com/en/.

[12] Facebook. *git. Version-control system for files*. URL: https://git-scm.com/.

[13] Facebook. *Jest. Delightful JavaScript Testing*. URL: https://jestjs.io/.

[14] OpenAPI Initiative. *open-api. API description format for REST APIs*. URL: https://www.openapis.org/.

[15] Google. *Node.js Client for Google Maps Services*. URL: https://github.com/googlemaps/google-maps-services-js.

[16] Tom Chen. *react-google-maps. React.js Google Maps integration component*. URL: https://github.com/tomchentw/react-google-maps.

[17] Stephen McDonald. *react-google-maps-issue. Node.js Google Maps Compatibility Issue*. URL: https://github.com/googlemaps/google-maps-services-js/issues/161.

[18] Google. *Maps JavaScript API*. URL: https://developers.google.com/maps/documentation/javascript/tutorial.

[19] Google. *Encoded Polyline Algorithm Format*. URL: https://developers.google.com/maps/documentation/utilities/polylinealgorithm.

[20] Overleaf. *overleaf. Online LaTeX editor*. URL: https://www.overleaf.com.

[21] Zube. *zube. Project managment software*. URL: https://zube.io/.

[22] Atlassian. *git-flow. GitFlow*. URL: https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow.

[23] Tom Preston-Werner. *semver. Semantic versioning*. URL: https://semver.org/.