CHALMERS | 

GÖTEBORGS UNIVERSITET

Miniproject: Distributed systems
Software Architecture Document

Group 5

Emanuel Dellsén - emanuel.dellsen@gmail.com

Hannes Ringblom - gusringbha@student.gu.se

Adam Klevfors - gusklevad@student.gu.se

Niklas Möller - gusmolnia@student.gu.se

Karl Westgårdh - karl.westgardh@gmail.com

Table of Contents

Conceptual design of architecture and the architectural style for Visual Transportation Support System

The Visual Transportation Support System is designed following the paradigm of the Publish-Subscribe architecture style as well as Pipe-Filter. A component named “TravelRequestProvider” serves as an independent generator and producer of travel request and publish these messages through the pipe to the second filter which then generates a topic based on the data received through the pipe. A component called “Visualiser” register an interest in topics and use the data received through these messages to draw lines on the map. Thus, the architecture is event-driven and operates in real time.

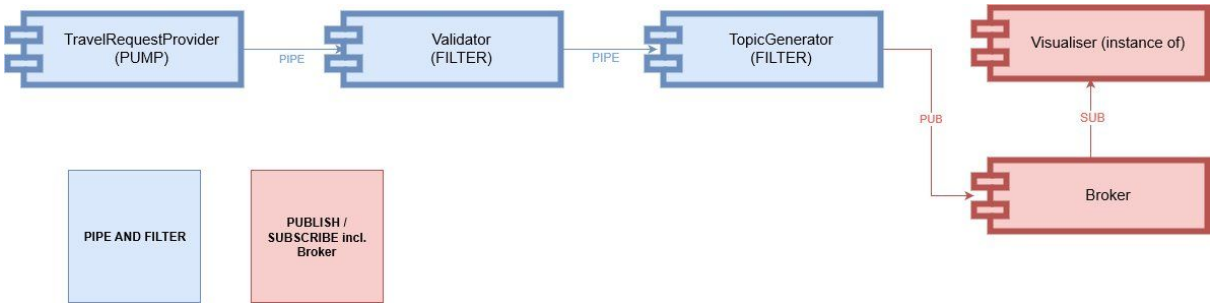


Diagram 1

Mapping the conceptual design onto implementation

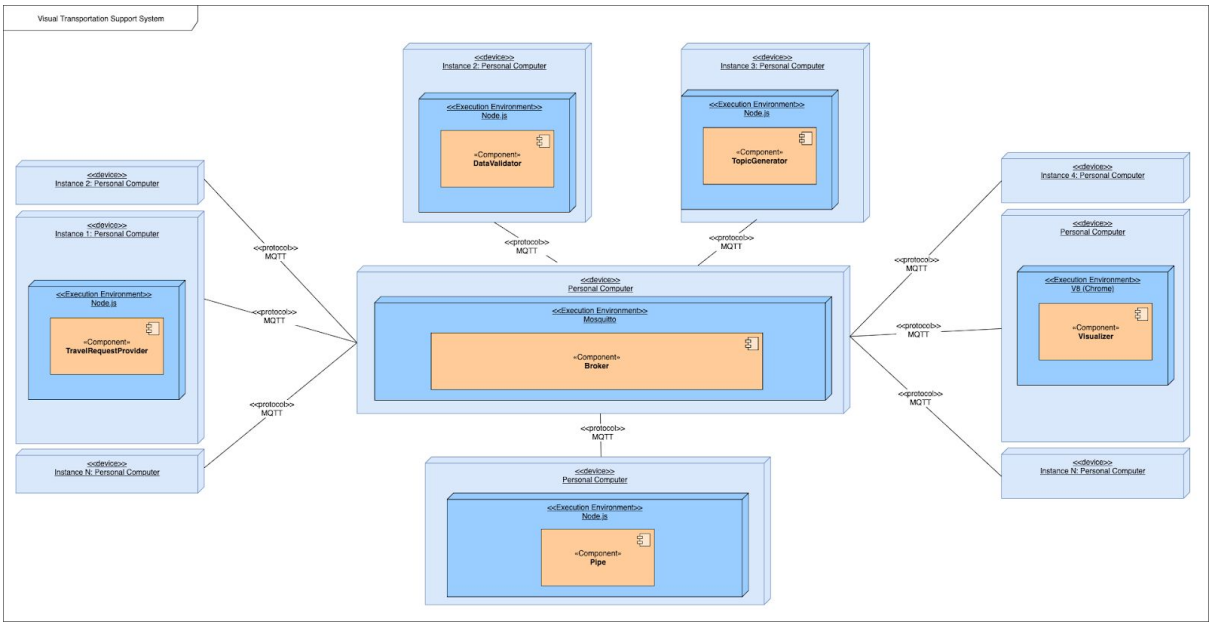


Diagram 2

As seen in the Deployment Diagram above, the system is running on personal computers that are communicating over the MQTT protocol. The system is able to scale, adding both several “TravelRequestProviders” as well as “Visualisers”. Only one broker is initially provided but several can be used.

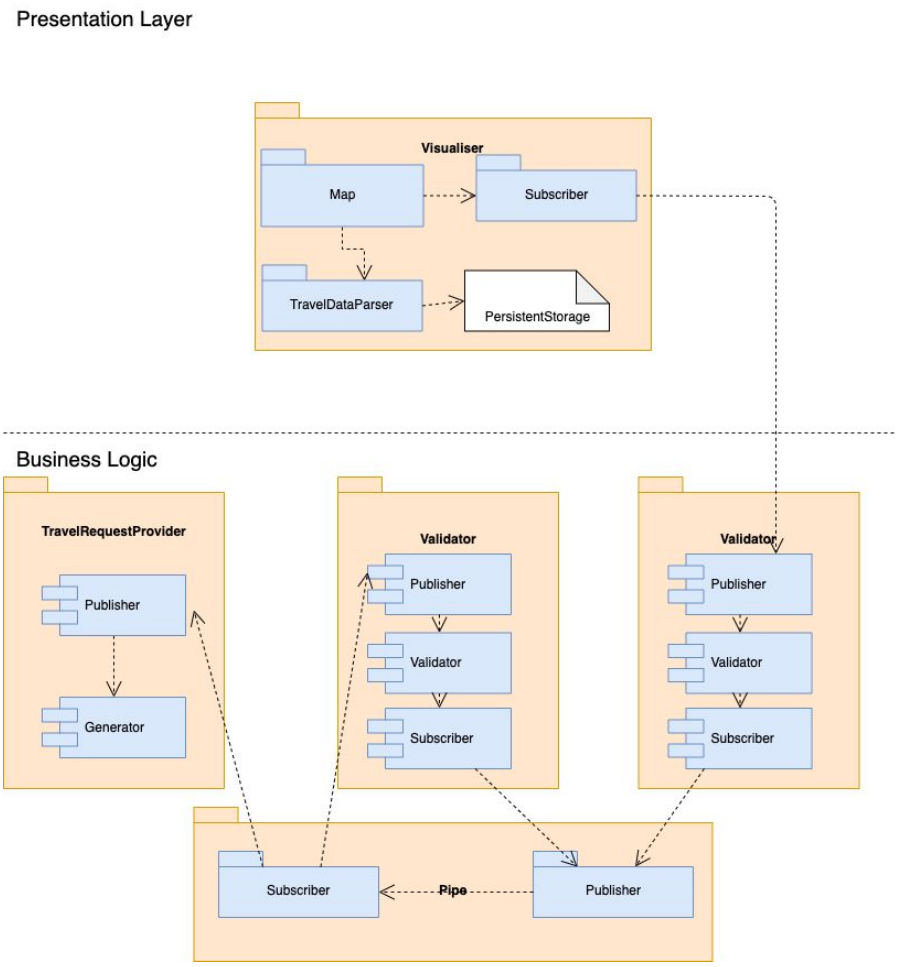


Diagram 3

In the package diagram above, the respective repositories/components can be mapped as independent packages that can be placed within different layers: *Presentation Layer* and *Business Logic layer.*

Each of the components contains sub-components with different responsibilities, interacting as communicated in the Component Diagram below:

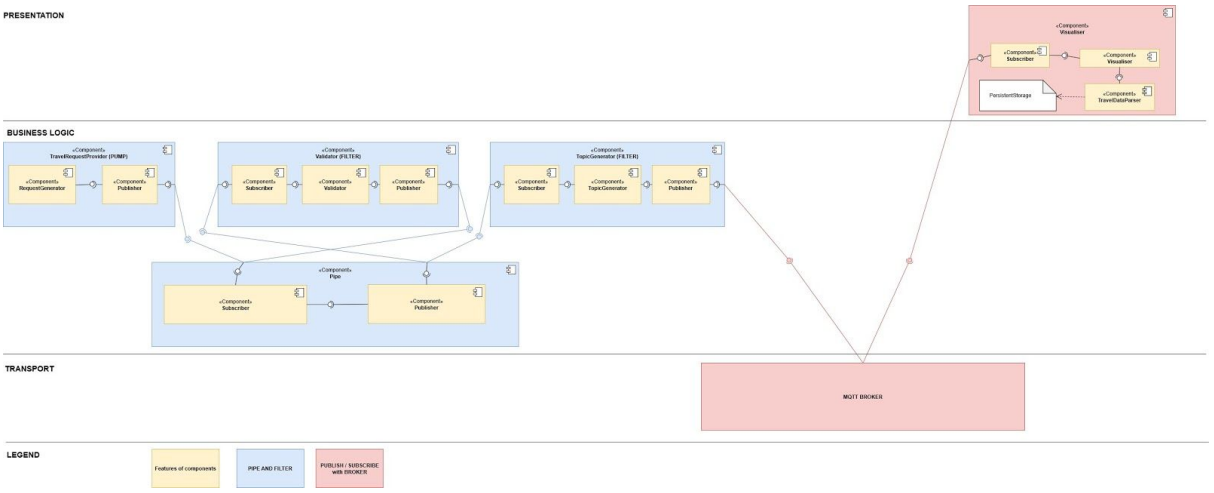


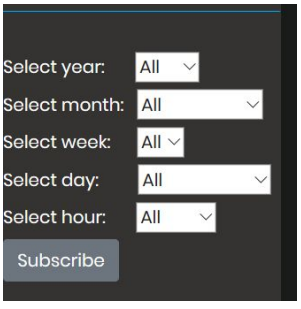
Diagram 4

By observing diagram 4 we can clearly see how the design takes into account two technologies, one being the “Broker” which is a part of the Eclipse Mosquitto Open source MQTT broker [1]. The broker component is the heart of the “publish and subscribe” architecture style as described in the section above.

The system has a Topic Generator-component that creates a tailored topic for the Visualizer to subscribe to. This topic is based on the particular departure time of the request, nested in alignment with the MQTT-protocol:

"json/year/month/week/weekday/time",

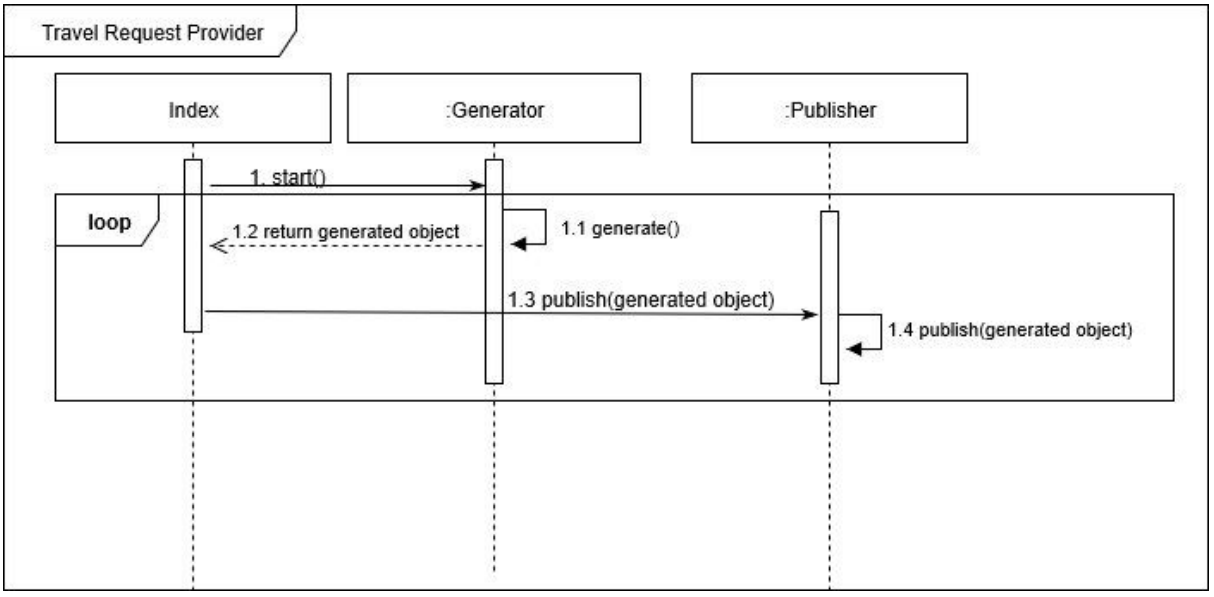
This gives the visualizer the possibility to subscribe to either all data or to a particular timespan[2]. This can be helpful if the data load is very big, as well as if two different screens are interested to show travel requests from different time spans in real-time.



Showing how the Visualizer gives permission to choose timespan for subscription

[1] https://mosquitto.org/
[2] For more info, see: https://www.hivemq.com/blog/mqtt-essentials-part-5-mqtt-topics-best-practices/

Sequence Diagrams

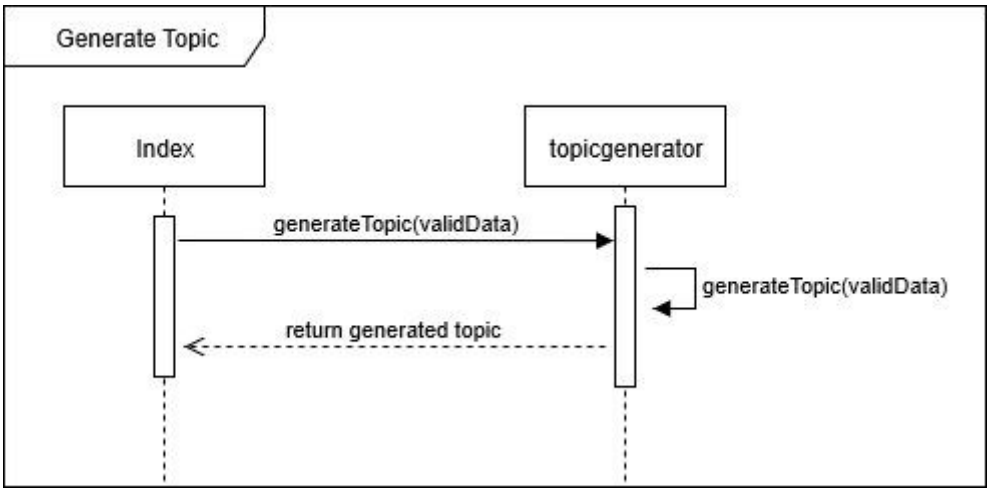


The Travel Request Provider - generating and publishing a travel request

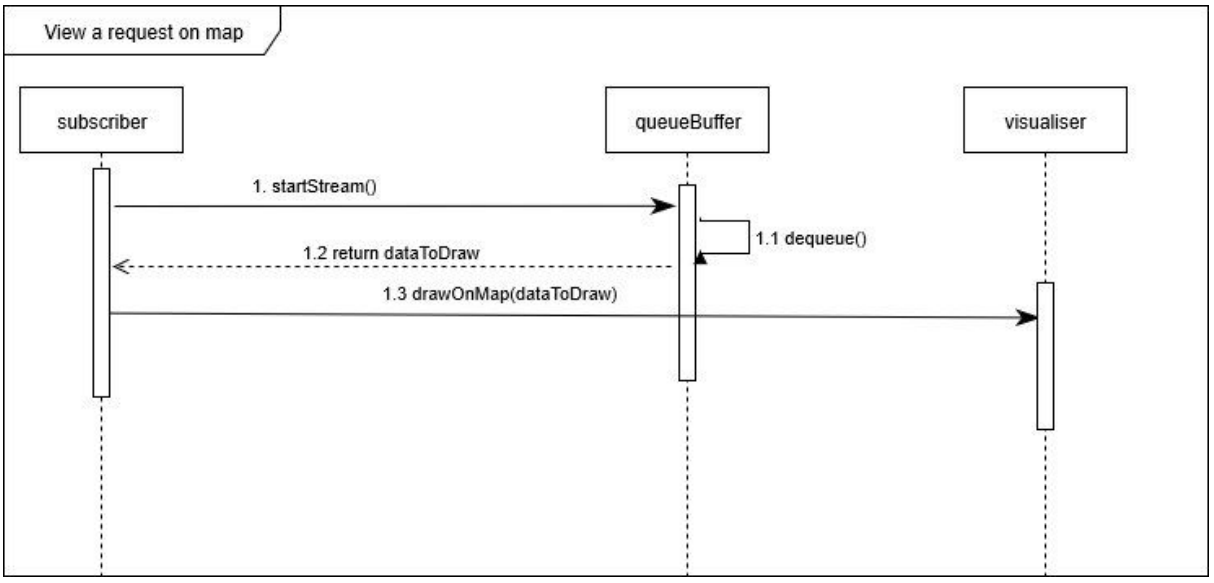
```
try {
    setInterval(function() {

        var generatedMsg = myGenerator.generate(reqId);
        reqId++;
        try {
            console.log('TravelRequestProvider publishing: '+generatedMsg)
            publisher.publish(generatedMsg);

        } catch (error) {
            console.log(error.message)
```

Code snippet from index.js in TravelRequestProvider. The setInterval maps to the loop in the diagram above[3].

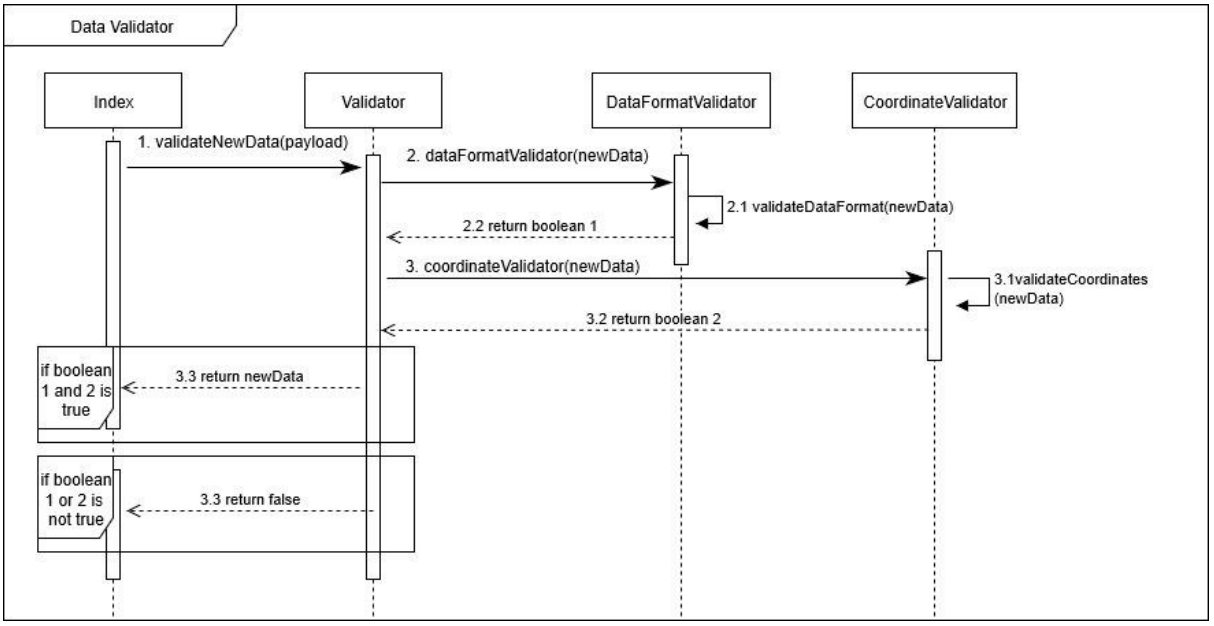
[3] The updating of requestID has been left out of the diagram



Generating a topic in TopicGenerator



Visualizer-component drawing a request as a line on the map



Data Validator - validating that a request has the right format and that the coordinates are within our given area.

Reflections about architectural choices

The initial requirements stated that the chosen architecture should not be based on Client/Server[4]. The solution in this project has been to use the *Eclipse Javascript Paho Client-library* for receiving MQTT-messages straight into the browser. The library is using WebSockets to communicate with our broker[5].

Another architectural decision has been to not rely on any external API in order to visualise public transportation data. Instead we have gathered raw data from Västtrafik containing transportation routes of busses, trains, ferries and trams[6]. These routes has been aligned with their respective travel times and modified by us into a GeoJSON-format[7].

The advantage of not using an API is that the Visualizer does not become constrained to a particular limit of data-points to process during a minute, which otherwise is the case with public transport API's[8]. In our system, we can with advantage receive more than 30 data points/minute.

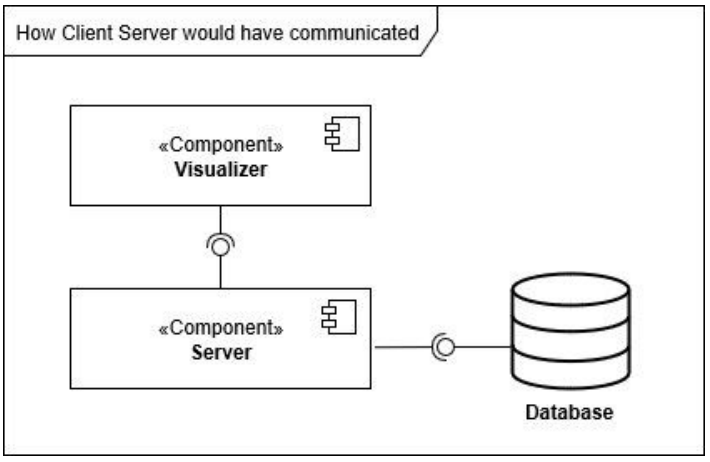
A design decision has been made to store this data in the visualiser component.

Showing commit of travel data - with file size 237 KB

This decision has mainly been found on the initial requirement to not rely on a client/server architecture. With such a design, a database would have been

[4] See the Kickoff slide: https://git.ita.chalmers.se/dobslaw/dit355-material-2019/raw/master/kickoff.pdf
[5] https://www.eclipse.org/paho/clients/js/
[6] https://github.com/vasttrafik/open-data-gtfs
[7] See the IO-chart for more info: https://git.chalmers.se/courses/dit355/2019/group-5/documentation---group-5/tree/master/IO-Schemas
[8] See https://www.trafiklab.se/api/resrobot-reseplanerare

implemented. However, since the data is only a few hundred Kilobytes, it has been considered fine to keep it in the client without making it to “thick”.



For a database to exist, a server would also have been necessary

The trade-off when not using Client/Server and a database is that the data will be multiplied with the amount of visualizers being active. Let us say our fictional “travel-data-analyst” would want to start the visualizer on three different machines, being able to analyze data over three screens. Then our data would exist in three nodes instead of one which can be considered redundant.

Another trade-off to keep the data in the client is that the scalability of the system becomes affected. Let's say that our system would need to visualize public transportation routes over entire Sweden, the data-size would have increased. However, since this is a system for the greater Gothenburg Region[9] this has not been considered a risk for the current scope.

[9] See the assignment:
https://git.ita.chalmers.se/dobslaw/dit355-material-2019/raw/master/groupAssignment.pdf

Appendix

Output schemas

```
{
    "list":[{
      "_comment" : "Output for RequestGenerator",
      "topic to publish under": "json/newData",
            "deviceId": "randomId",
            "requestId": "randomId",
            "origin":{
            "latitude":"latitude",
            "longitude":"longitude"
            },
            "destination":{
            "latitude":"latitude",
            "longitude":"longitude"
            },
            "timeOfDeparture":"YYYY-MM-DD HH:MM",
            "purpose":"purpose"
    }]
}
```

1. *Format of data published by TravelRequestGenerator*

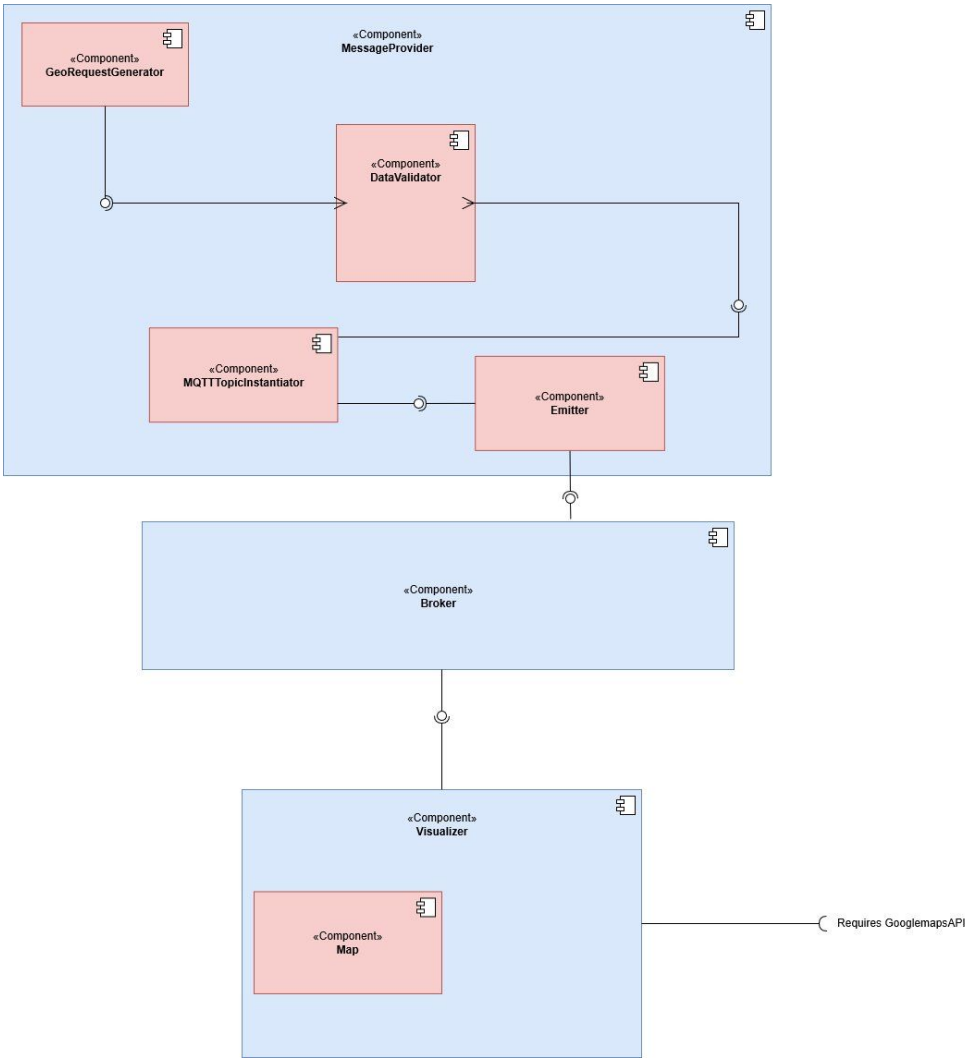
```
{
    "_comment": "Output",
    "topic to publish under": "json/validData",
    "deviceId": "randomId",
        "requestId": "randomId",
        "origin":{
        "latitude":"latitude",
        "longitude":"longitude"
        },
        "destination":{
        "latitude":"latitude",
        "longitude":"longitude"
        },
        "timeOfDeparture":"YYYY-MM-DD HH:MM",
        "purpose":"purpose"
        }
    }
}
```

2. *Format of data when the data has been validated - topic has changed*

```
{
    "_comment": "Output",
        "topic to publish under": "json/year/month/week/weekday/time",
        "deviceId": "randomId",
            "requestId": "randomId",
            "origin":{
            "latitude":"latitude",
            "longitude":"longitude"
            },
            "destination":{
            "latitude":"latitude",
            "longitude":"longitude"
            },
            "timeOfDeparture":"YYYY-MM-DD HH:MM",
            "purpose":"purpose"
}
```

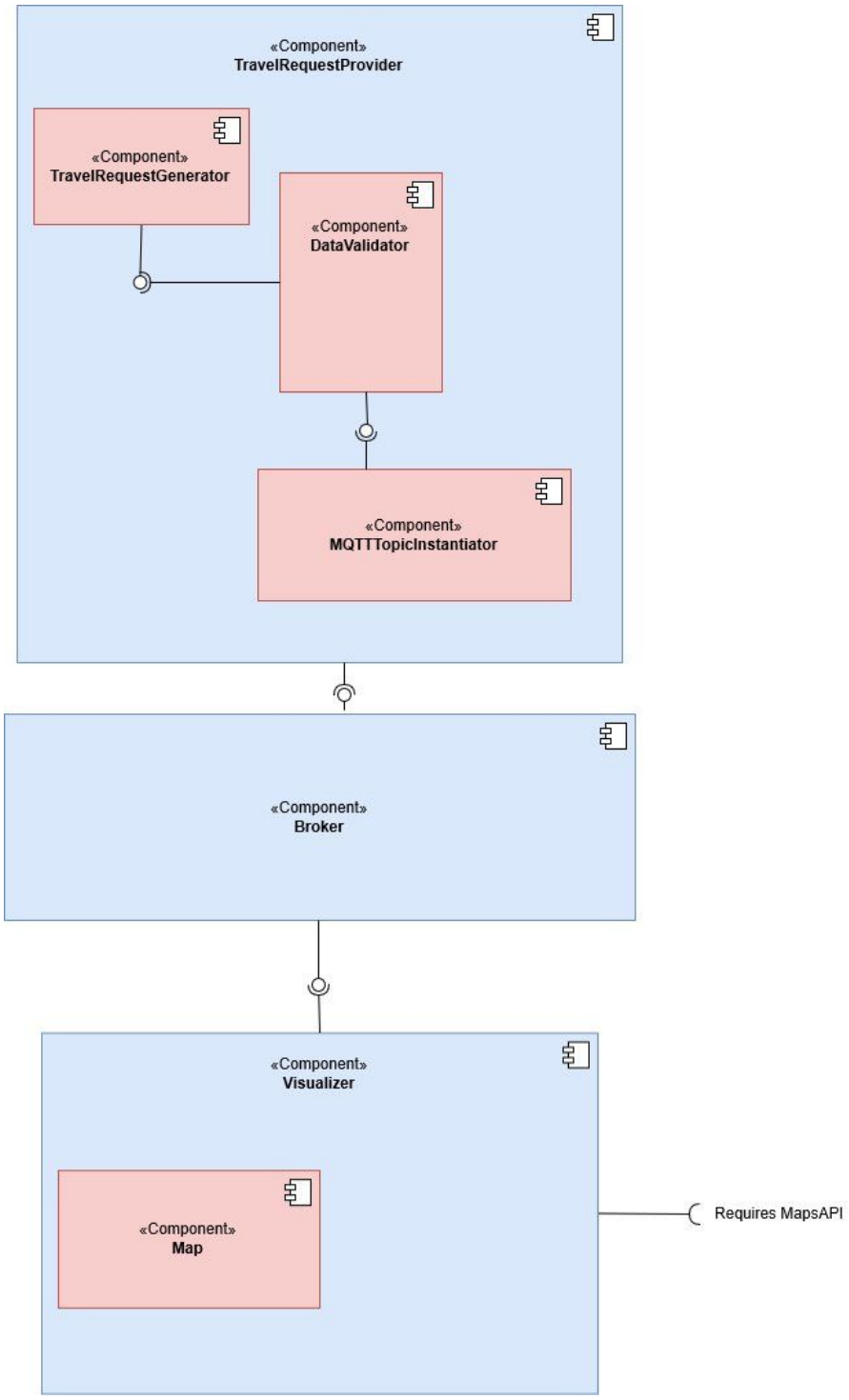
3. *Format of output data of Topic Generator - a customized topic has been generated*

Tracing how the system has evolved



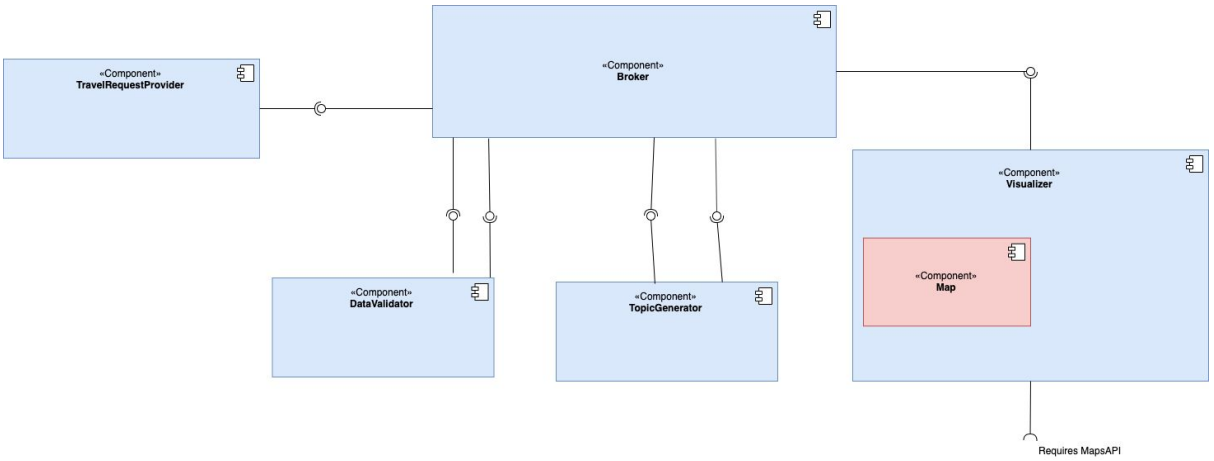
1. 15 nov.

The system has only two major components - a “Message Provider” and a “Visualizer”. Considering to use Google Maps API.



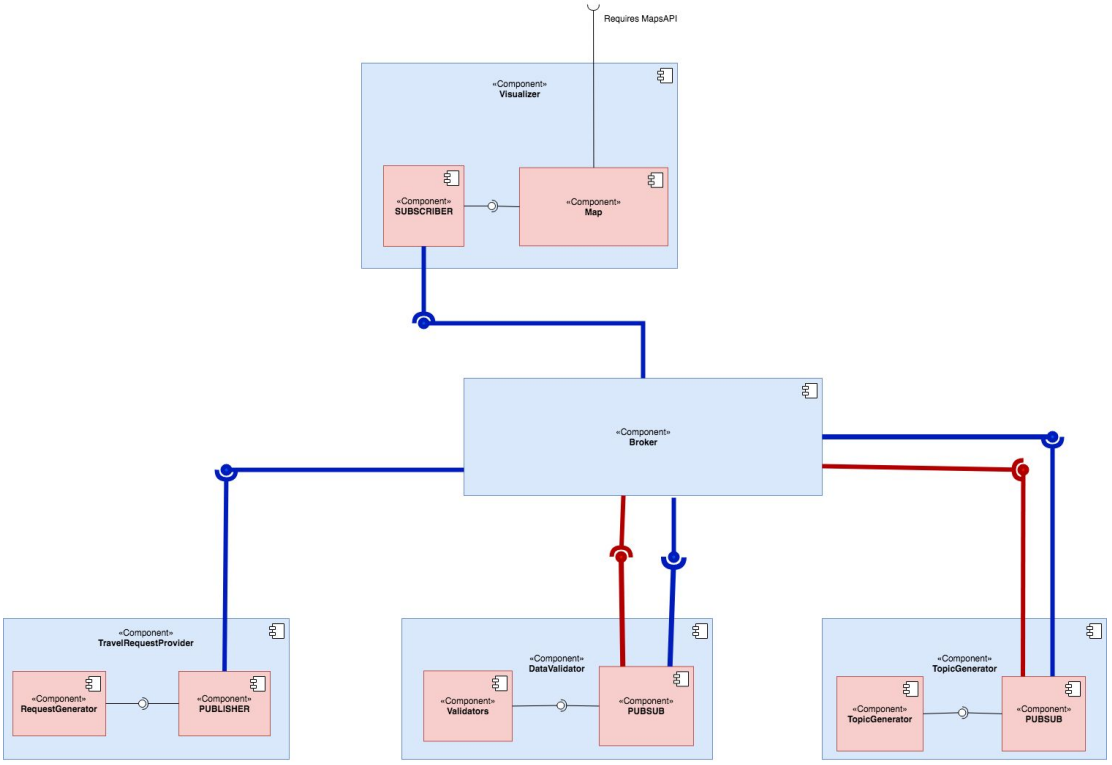
2. 18 nov

The Emitter sub-component is removed since we don't consider it valuable



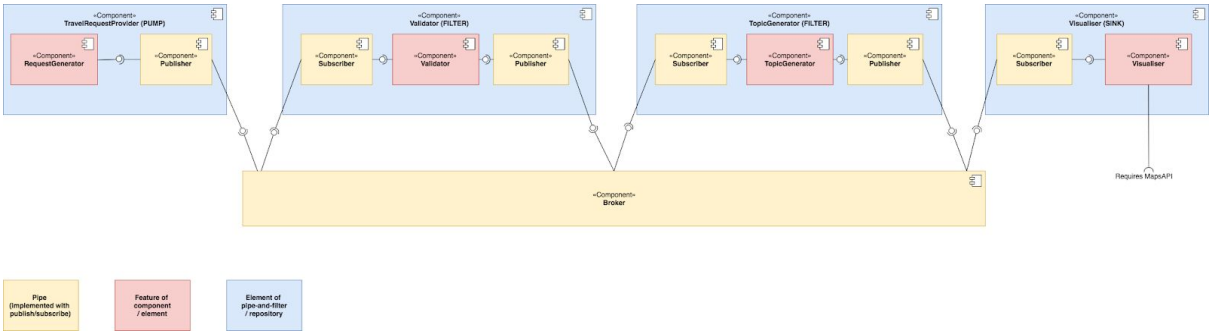
3. 20 Nov

The different sub-components are decoupled



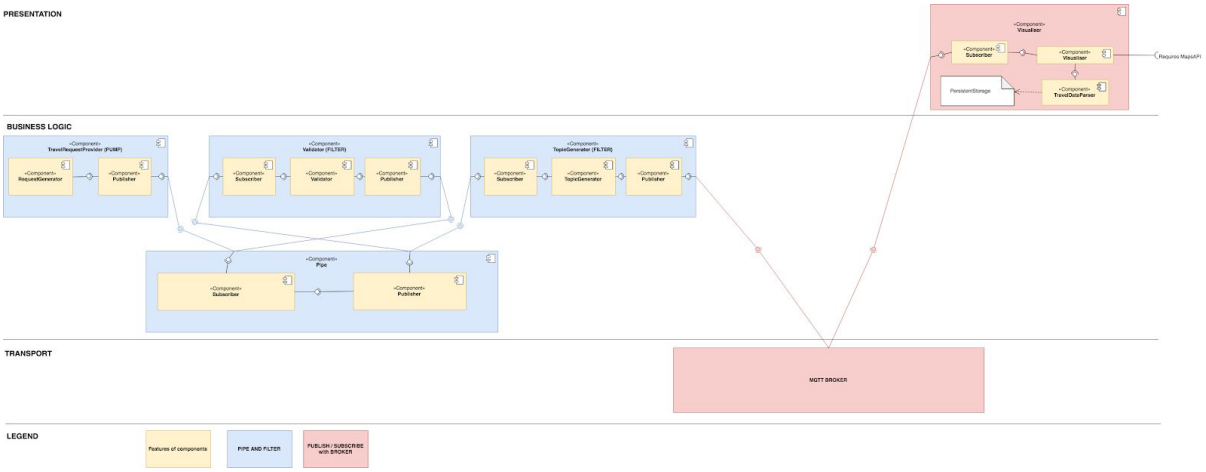
4. 1 Dec

Sub-components are shown and thoughts about styles start to take place



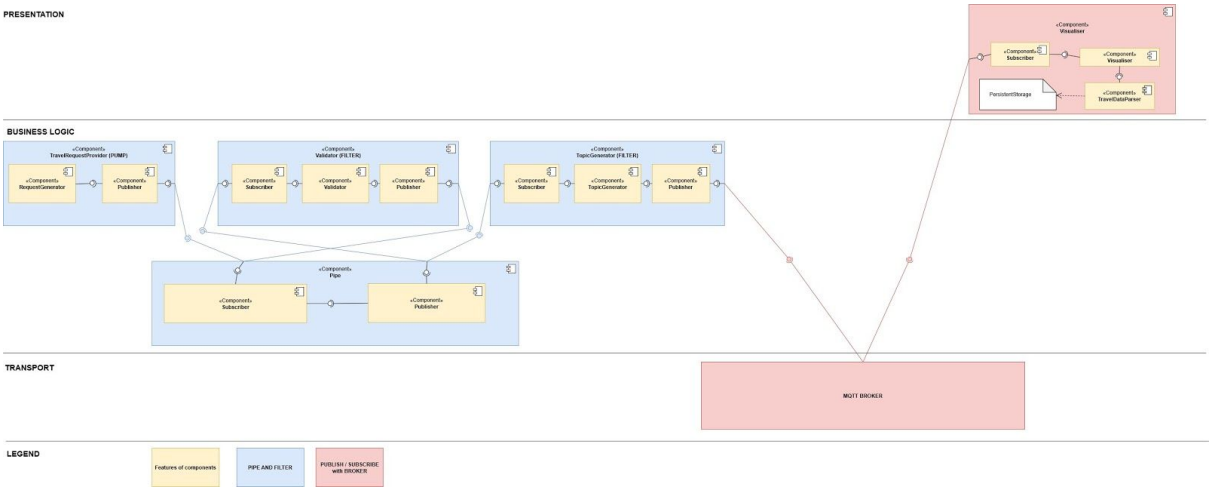
5. **7 Dec**

   Considering the Broker our pipe



6. **11 Dec**

   New version with a pipe component, as well as showing the different sub-components within the Visualizer. Mapping to layers as well.



7. **18 dec**
   Final version - the Maps API has been removed since it was never implemented