

Trung Nguyen

**SOFTWARE ARCHITECTURE & SOLUTION IN CITY LOGISTICS
PRODUCT SEGMENT COVERING PASSENGER LOGISTICS
PRODUCT PORTFOLIO OF ATTRACS OY AB**

**SOFTWARE ARCHITECTURE & SOLUTION IN CITY LOGISTICS
PRODUCT SEGMENT COVERING PASSENGER LOGISTICS
PRODUCT PORTFOLIO OF ATTRACS OY AB**

Trung Nguyen
Bachelor's Thesis
Spring 2021
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme

Author(s): Trung Nguyen

Title of the bachelor's thesis: Software Architecture & Solution in City Logistics Product Segment Covering Passenger Logistics Product Portfolio of Attracs Oy Ab

Supervisor(s): Patrik Friis, Lasse Haverinen, Hannu Rauhala, Toni Penttinen

Term and year of completion: Number of pages: 63

The subject of the thesis is to document my professional software development during my employment at Attracs Oy Ab as Software engineer. The scope of work covers the City Logistics Product of Attracs Oy Ab.

The main aim is to increase my fundamental knowledge of Golang, PostgreSQL, and Typescript in full-stack software development and apply the software development experience to contribute to the City Logistics Project's overall growth.

The software is a combination of many microservices, which I was responsible for developing with Golang and Typescript for the web and mobile application. The programming scope includes Client UI for web and mobile applications, design API for server, and Database Schema.

The works were tested in the internal software demo and the production environment, with features and bug fixes not limited to a single technology stack or microservice. With peer code review, agile methodology in software deliveries, the work scope resulted in good feedback from my colleagues and customers.

Keywords: Attracs Oy Ab, City Logistics product, Software Development, Golang, Typescript, PostgreSQL.

PREFACE

The thesis work was documented in Kokkola, Finland, during my employment at Attracs Oy Ab as a Software Engineer. The author chose the thesis topic after discussion with Attracs Oy Ab project managers.

Supervisors of this thesis were Patrik Friis, Hannu Rauhala, and Toni Penttinen as project managers at Attracs Oy Ab, with Lasse Haverinen, who acted as the tutoring lecturer, along with Heidi Hedström, who provided language review. The thesis literature is written with the supervision and direction of the supervisors to be fulfilled.

Kokkola, 08.03.2021
Trung Nguyen

CONTENTS

1 INTRODUCTION	6
2 ATTRACS OY AB	8
2.1 The employer company and the work environment	8
2.2 Attracs Oy Ab interests in City Logistics product	10
2.3 City Logistics product	11
2.3.1 The professional concepts of the City Logistics product:	11
2.3.2 The general software architecture of the City Logistics product:	13
2.3.3 Client development of City Logistics product and microservice	14
2.3.4 Server Development of City Logistics product and microservice	15
3 PURPOSE AND OBJECTIVES	16
4 DESCRIPTION OF WORK TASKS AND LEARNING	17
4.1 Thesis entry from the 31 st of August to 4 th of September 2020	17
4.1.1 Research and study	17
4.1.2 Theoretical background and technical information	18
4.1.3 Implementation	19
4.1.4 Reflection	20
4.2 Thesis entry from the 7 th of September to 11 st of September 2020	21
4.2.1 Research and study	21
4.2.2 Theoretical background and technical information	23
4.2.3 Implementation	24
4.2.4 Reflection	27
4.3 Thesis entry from the 14 th of September to 18 th of September 2020	29
4.3.1 Research and study	29
4.3.2 Theoretical background and technical information	30
4.3.3 Feedback service	30
4.3.4 Billing Microservice	31
4.3.5 Reflection	32
4.4 Thesis entry on the week from 21 st of September to 25 th of September 2020	33

4.4.1 Research and study	33
4.4.2 Theoretical background and technical information	33
4.4.3 Implementation	34
4.4.4 Reflection	36
4.5 Thesis entry on the week from 28 th of September to 2 nd of October 2020	37
4.5.1 Theoretical background and technical information	37
4.5.2 No-reply email implementation	38
4.5.3 Excel billing document implementation	39
4.5.4 Reflection	39
4.6 Thesis entry on the week from 5 th of October to 9 th of October 2020	40
4.6.1 Theoretical background and technical information	41
4.6.2 Research and study	41
4.6.3 Implementation	43
4.6.4 Reflection	44
4.7 Thesis entry on the week of 16 th of October to 27 th of October 2020	45
4.7.1 Research and study	45
4.7.2 Theoretical background and technical information	46
4.7.3 Implementation	47
4.7.4 Reflection	48
CONCLUSION	50
REFERENCES	52

ABBREVIATIONS

GPS: Global Positioning System

UI: User Interface

UX: User Experience

HTML: Hypertext Markup Language

API: Application Programming Interface

HTTP: HyperText Transfer Protocol

SQL: Structured Query Language

LOC: Line of Code

JSON: JavaScript Object Notation

RPC: Remote Procedure Call

npm: Node Package Manager

i18n: Internationalization and Localization

DRY: Don't Repeat Yourself

UUID: Universally Unique Identifier

DOM: Document Object Model

1 INTRODUCTION

Public transport is widely used in society today to serve everyone's travel needs. In most cities in Finland, public bus service or metro station is one of the most commonly used means of people's daily lives.

However, for a growing population in Finland, public transport is a huge barrier because of health and age limitations. According to Statistics Finland's statistics on the population, there were 874,314 people at the age of 70 or older in Finland at the end of 2019, of which 370,405 were men and 503,909 ladies (1.) This group of people has about at least 15 percent of the population share in the whole of Finland (1.). The number of people aged 70s or over has increased by 100,000 every three years (2.). From 2006 to 2016, around 16 percent of people from the age of 15 to 24 are reported not to study, work, or perform compulsory military services due to disability or chronic illness (3.). Some municipal offices in Finland look for other transportation means to serve their commuting needs in such a context.

Attracs Ab Oy is a 20-year-old company in the transportation and logistics software solutions, a fully owned subsidiary of Ahola Transport Ab Oy. With the company mission for efficient and profitable logistics, Attracs Ab Oy develops the City Logistics Product for Ahola Transport Ab Oy and FCG Finnish Consulting Group Oy's joint venture in incorporating private transportation with a smart digital solution to deliver a new intelligent city transportation solution for people with age and health limitation.

The goals of the thesis are to document my progress in working and learning during my employment at Attracs Ab Oy while being part of the City Logistics Product software development team. My work involves different technologies and programming languages to improve the software development team's software solution for the municipal offices in city logistics and human transportation. I will develop the software client and server with Golang, Typescript, and PostgreSQL to deliver the goals. With practical application,

results, peer code review, and customer feedback, I conduct my portfolio writing about my professional works every week for my thesis work.

The first part of the thesis will be an introduction portfolio of the company's current state, the work environment, and myself. This introduction will be about theoretical and professional concepts from the work results and my professional development and development needs. The second part will list the purpose and the objectives of the thesis writing. The third to the final part of the thesis entries will address my weekly work tasks, the work reflection against the theoretical background, and the solution from the reflection.

2 ATTRACS AB OY

2.1 The employer company and the work environment

Attracs Ab Oy is a 20-year-old company in the transportation and logistics industry. The company management team includes Tommi Hollström, Juha Åkerlund, and Toni Penttinen.

The work atmosphere is agile. The company uses project management tools, a version control system, and a communication application to decentralize software development and business responsibility. The responsibility decentralizes one person or group of persons. It opens to team members to obtain an effective way of delivering a particular task, from development to deployment and from an internal meeting to customer meeting.

The software development environment is similarly agile oriented in terms of development and task management. Tasks are organized weekly according to the customer meeting and assigned to the developers. Each assignment will be labeled with the Status (the Progress of the task), the Author (the person who is responsible for developing the task), and the Creator (the one who create the task). (13.)

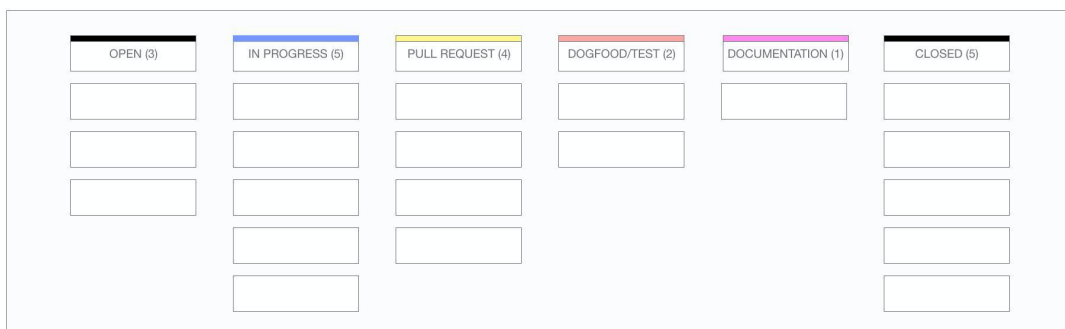


Figure 1 Tasks Management Process

From the development viewpoints, similar to industry-standard of Agile software development, the progress of the software development starts from local machine (14.), which changes the status of the task from Open to In Progress. Once the task is complete, the code will be made available as a pull request, which changes the status to Pull Request. (13.)

Each pull request will be examined by an assigned person, tested by company CI tools that run simple unit testing to assure quality before merging to the master branch of the application repository. If code quality is deficient, the assigned developer will request further code revision. Afterward, the master branch of the application repository inherits new code changes; the CI tools will automatically deploy the latest change code to the development version, which will change the development into nightly/test. (13.)

At this stage, the code is available in the development application version. The project manager will test the feature in the testing environment and product of the authored developer or the peers to ensure the developing code meets the business requirement. The development application will be similar to the client environment in many ways. The authored developer will request an adjustment if the submitted feature breaks the development application or does not deliver the business requirement. (13.)

Finally, when the code matches the specified standard, it will be ready for the next weekly application release to the client application for up-to-date feature change. The task status is Closed, and the developer can move on to the following tasks. (14.)

2.2 Attracs Ab Oy interests in City Logistics product

The City Logistics product is a software product of Attracs Oy Ab hereafter known as Attracs. The software development is for Ahola Transport Oy AB and FCG Finnish Consulting Group Oy's joint venture for delivering a smart digital transportation solution for municipal needs.

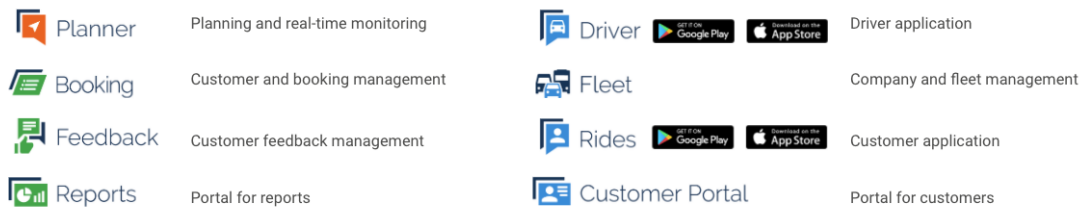


Figure 2 City Logistics product

The product aims are to provide a cloud-based dispatching system for municipalities to incorporate private transportation from taxi companies for a new means of public transportation. It provides a booking solution for taxi companies, with a booking system via mobile clients or a web portal. It also includes microservice planning software for route planning, real-time monitoring, automated/manual dispatching, and transportation assignment.

The service use of the software aims to provide an economical means of public transportation with a smart system to monitor booking action behaviors from the customer and driving experience from taxi companies. It also tailors the driving experience to fit with many customer profiles with a large diversity of health and age limitations. This customization ensures the pickup to drop-off experience is seamless in municipal agreement with the taxi companies. (15.)

2.3 City Logistics product

2.3.1 The professional concepts of the City Logistics product:

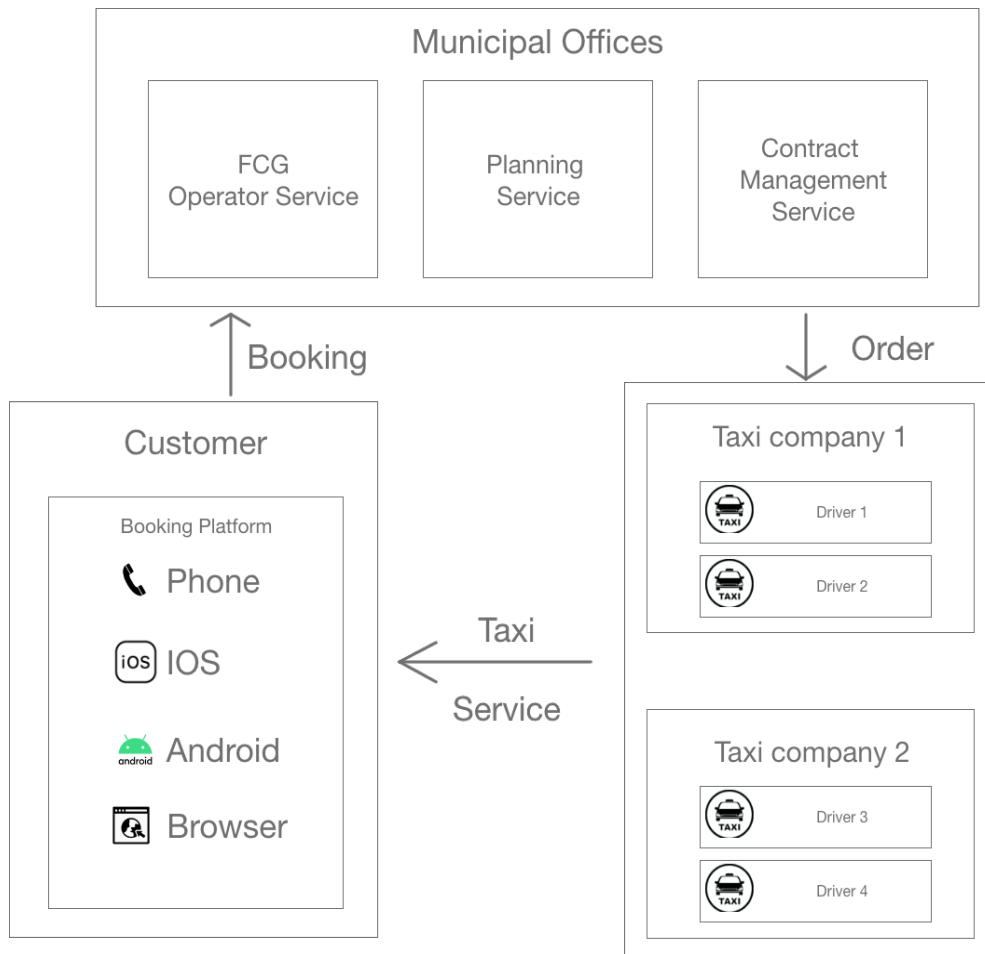


Figure 3 User flow of City Logistics product

The service is meant to be a digital platform for the municipality to support a group of people who cannot use public transportation due to health and age limitations by providing private transportation means through an agreement with taxi companies. The platform would support this special group of people to make transportation reservations, enabling the municipal office to arrange those reservations and bookings through an intelligent digital platform to create a

private vehicle to provide the transportation service for the one who requests the service.

The service customer can request a taxi reservation from the municipal offices through a browser, mobile application, or by making a phone call. This service is a part of the booking platform, a digital microservice provided for the municipal office to support the locality. The customer can use the booking through a mobile application, a web application, or a phone call to the office to make known the booking request.

The booking requests are then processed and handled by the digital planning microservice, where the office through the platform will order a vehicle from the taxi company to drive the booking. The planning service, operated through an intelligent algorithm and municipal operators, will look for any available vehicles from the taxi to drive the booking. The planning microservice algorithm will determine the available vehicles are suitable for the booking person by criteria of disability vehicle support (stair climber, ramp, low floor, rollator), general trip costs, availability, location at given booking time and other parameters like a vehicle CO2 emission. The planning service reduces the waiting time and provides a fair judgment on what kind of vehicle is the best fit for the booking.

When the platform processes the booking and finds a suitable vehicle for the booking, the taxi driver will receive a trip from the platform to drive the booking person. The taxi driver response to the offers by the driving microservice mobile application to keep track of the driver working hour, his acceptance to take on a booking and real-time monitor the status of the trip base on the GPS reading of the driver phone through the driver app (On customer being pickup, driven and dropped off). The driver mobile application ensures that there could be a driver available for the booking. It monitors the traffic route on whether the booking is delivered by a driver at all. After the booking is driven, the driver application will create a bill for the taxi company driver to send that trip invoice to the municipal office to pay for the trip cost. (15.)

2.3.2 The general software architecture of the City Logistics product:

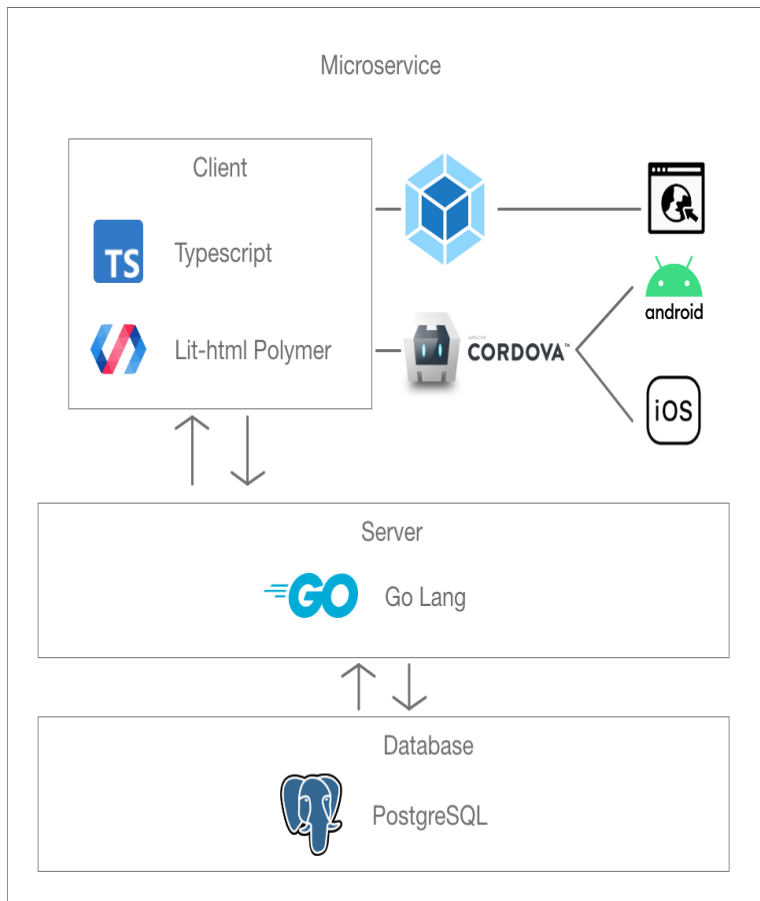


Figure 4 Technology Stack for A Microservice of City Logistics

According to Wikipedia, A microservice is not a layer within a monolithic application (for example, the web controller or the backend-for-frontend). Rather, it is a self-contained piece of business functionality with clear interfaces and may implement a layered architecture through its internal components.

The City Logistics product architecture is classified as Microservice Architecture (17.). Even though the understanding and definition for microservices in City

Logistics Oy are not definitive or distinctive, it can be characterized by the following traits:

- Isolation: Services in a microservice architecture interact over HTTP/2 network protocol using gRPC(gRPC Remote Procedure Calls). (11.)
- Productivity: Services are defined and focus around a business feature of the City Logistics product. (Figure 2.)
- Flexibility: Services may have different programming libraries or frameworks to fit best on its needs and purposes. (Figure 3.)
- Scalability: Services are scalable and thus autonomously developed and independent from others in deployment. (Figure 21.)
- Faster product development cycle: Services work independently, and components can be developed, modified and deployed individually.

2.3.3 Client development of City Logistics product and microservice

Client development or frontend development is a practice of converting data to a graphical interface through the use of HTML, CSS, and JavaScript so that users can view and interact with that data. (19.) For the City Logistics product, the business service is providing for both web and mobile platforms, using TypeScript as the client programming language across microservice.

TypeScript is a strict syntactical superset of JavaScript that has optional static typing and is compiled down to an idiomatic JavaScript. Using the type system from TypeScript brings benefit to client development in writing predictable and readable code that eliminates typos or type assumption mistakes. The interface APIs of the data set are often self-explanatory and discoverable through type declaration as opposed to using JavaScript. Thus, unit testing in client development will spend more time focusing on testing feature-capability over type checking. (19.)

lit-HTML is a client-side JavaScript library that supports TypeScript. It let the developer writes HTML templates in TypeScript and renders or re-render those templates with data to create and update the User Interface. lit-HTML is efficient and considered extremely fast for its rendering UI mechanism. Unlike Virtual DOM libraries, the library only updates the User Interface parts that change and do not re-render the entire view. Because lit-HTML is not a framework and focuses on efficiently creating and updating DOM, it is combined with Apache Cordova to build a hybrid mobile application for Android and IOS from one single codebase (5.) (20.).

Apache Cordova is a mobile application development framework. In the City Logistics product, Apache Cordova (or Cordova) allows standard web technologies – HTML5, CSS3, and idiomatic JavaScript for cross-platform development with access to device’s capabilities such as network status, GPS, and location.

2.3.4 Server Development of City Logistics product and microservice

Go is a statically typed, compiled programming language designed at Google. Go is syntactically similar to C, but with memory safety, structure typing. The language is often referred to as Golang because of its domain name, golang.org. (21.) The language consists of:

- A syntax and environment utilizing patterns more common in dynamic languages.
- Fast compilation.
- Native binaries produced without external dependencies.
- Remote package management (get) and online package documentation.

Because of the lightweight characteristics for fast compilation without dependencies, the microservices' server is written in Golang.

PostgreSQL is an open-source object-relational database system that uses and extends the SQL language combined with many features that store and scale the complicated data workloads. Some storage features, such as Indexes, Schemas, and Data Types, are often utilized in creating and maintaining a robust and performant database infrastructure. Attracs' own customized library is used for connecting to PostgreSQL from the Golang server.

3 PURPOSE AND OBJECTIVES

My purpose and objectives for the thesis work are as following:

- To get familiar with Attracs Ab Oy microservice technology stack.
- To familiarize with client-side development with lit-HTML and TypeScript on the library to create web and mobile application features through webpack and Cordova.
- To get familiar with server-side development, using Golang and PostgreSQL.
- To make a sizeable code contribution to the microservices that I will work on, from a small to medium and large-scale task.

4 DESCRIPTION OF WORK TASKS AND LEARNING

4.1 Thesis entry from the 31st of August to 4th of September 2020

On the 31st of August, after a month of working at Attracs Oy Ab, I started my thesis work. At this point, I was moderately familiar with the fundamentals of internal company technical infrastructure and workflow and had become accustomed to the technologies and the digital solution delivery. The first four weeks helped me develop more insights and expertise about the client and the server programming languages. This experience allowed me to be more prepared to start my first thesis week.

4.1.1 Research and study

I have earlier received some UI-related problems with the customer mobile app. The issue was about making the UI more convenient and intuitive for our customers to make a booking. The solution was from the community perception of intuitive UI and UX research on how the user will use the mobile screen device. (Figure 6.)

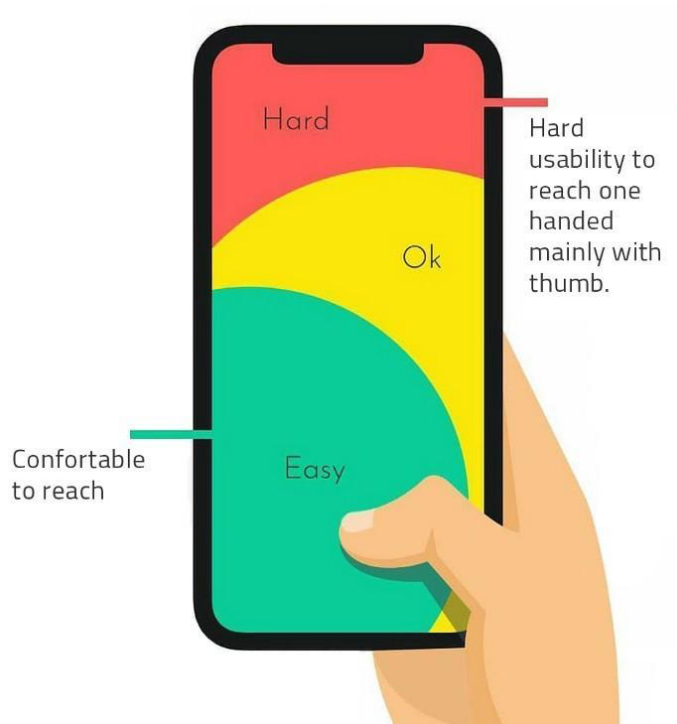


Figure 5 UX Study on Mobile Device Screen (4.)

In our case, we wanted the thumb motion to make a booking in our UI without having to outreach their one-hand holding. According to a research (4.), 75% of mobile app users use their smartphone in one-hand holding position and thus have limited comfortable screen interaction in the screen area. By placing the booking input at the bottom of the screen and pushing them gradually to the top by the completion status, we achieved screen comfort for the use case of one-hand holding, left-hand interaction, and two-handholding position.

4.1.2 Theoretical background and technical information

The User Interface is written in an HTML templates in TypeScript by lit-HTML to render the UI with data. lit-HTML has two main APIs: `html` and `render()`.

- The `html` template tag is used to write templates.
- The `render()` function is used to render a template to a DOM container.

A lit-HTML expression does not trigger any DOM to be created or updated. It is only a description of DOM, called a TemplateResult. The `html`` or `TemplateResult` needs to pass to the `render()` function to trigger the UI rendering. (5.)

Another concept of lit-HTML for making dynamic UI change in the mobile application is called directives. Directives are function that changes the appearance or behavior of a UI or a DOM element.

Internally, lit-HTML uses the Part interface to represent the dynamic DOM associated with a binding. A directive has access to the Part associated with its binding. For example, it can find the current value of the part and set a new value for the part. Directives in lit-HTML shares a common usage and understanding with other client-side libraries and framework directives and component states in:

- Modifying the behavior of an element.
- Responding to events that change the directives' behavior.
- Asynchronous directives.
- One binding type directive.

Apache Cordova is used to creating a hybrid app using Cordova CLI on Android and IOS platforms. Cordova is a native wrapper application where the lit-HTML and TypeScript code is embedded to create a native shell application. This native shell application is the platform web view component that loads the web application and gives the developer the ability to create and publish native applications that can be submitted to each of the platform's app stores. (23.)

4.1.3 Implementation

The implementation was a twofold process: to update the component data and to clear the UI. As a lit-HTML web component, the component has its local state

data to contain information (5). In this UI component, the local state data was the search field typing data and other address data. Once the user fills the input data, the local state's component binds them to the mobile application UI and updates the interface.



```
reset() {
  //Note: Reset the search request
  this.request = new BookingRequest();

  //Note: Reset the search field completion
  this.progress = new FieldProgress();

  //Note: Clear the search field completion
  let dummy_addr = new tms.v1.Address();
  this.fillAddressInput('from', dummy_addr);
  this.fillAddressInput('to', dummy_addr);
  this.fillAddressInput('waypoints', dummy_addr);
  this.fillTimeInput('time', '');

  this.timeNow = true;
}
```

Figure 6 Clear Search Field Mock Code

Thus, in the reverse logic, once the local component data was cleared on the condition that all the search fields are complete, the local information was reset to have an empty string and update the object google map prediction as an empty object. (Figure 7).

I also worked on other UI tasks to build a button to clear the mobile app search input. The issue occurred as the need to undo a booking exhibit gradually—the current interaction in removing search fields one-by-one to be a poor, labor-intensive user experience.

4.1.4 Reflection

The implementation in this week includes:

- Using lit-HTML to render HTML template elements.

- Listening to events to change the component state and directives.

By using the lit-html to render HTML template from listening to component states, the application was able to perform the clear search function as intended.

The concepts of render() and directives in lit-HTML is noteworthy similar to how other common client-side libraries and framework manipulate web component UI. This application of the concepts and the skills learned from it is beneficial in improving the basic understanding of web development regarding DOM manipulation and how DOM manipulation is used in client-side libraries and frameworks to create a good UI behavior that could lead to a good UX use case.

In the current task, the immediate UX benefits from the concepts and knowledge are to serve the feature for the visually impaired and elderly people. Since this feature of the application is expected to be simple and performant, the knowledge serves in providing good UX and good accessibility for this user group.

4.2 Thesis entry from the 7th of September to 11st of September 2020

On the second week of the thesis work, I received a task to programmatically document billing data from a certain period and sending the documents as an attachment email to a specified address. Besides the billing task, I also received some assignments about booking microservice concerned about UI data display and behavior.

4.2.1 Research and study

According to Figure 2 and Figure 4, the architecture of the City Logistics product is built on the microservice concepts. Therefore, the children applications of the City Logistics are in compliance with this principle. Each application is personalized for a business logic aspect of the City Logistics product.

The Booking and Billing application that I was working on are an independent application. The Booking service focus on providing different platforms for making transportation booking to different customer audience. The Billing service provides a systematic process of trip billing documentation and calculation base on customer group, customer service, and municipality. To conclude, the Billing service provides management of all billing data, and the Booking service focus on the monitor of booking and customer data.

Regarding the task of making excel documentation of billings information, it was implemented on the Billing service. The implementation happened of client and server in the Billing service codebase and required knowledge of lit-HTML client library, Excelize Golang package, SendGrid Golang package, and PostgreSQL database access in Golang server.

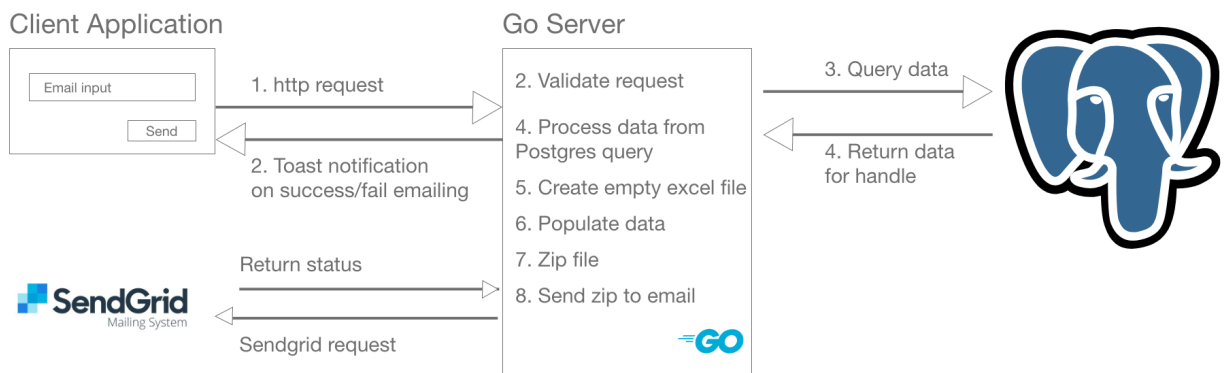
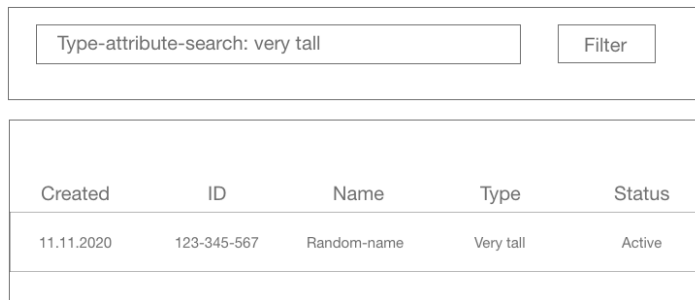


Figure 7 User flow of billing document and automated emailing system

Figure 7 describes my study and research in designing and coding the Excel billing documentation for the Billing service. The figure shows different technologies and their interaction in a user flow for making the excel billing documentation. The interaction starts from the client application with information about making billing documents and sends them to the Golang server. The Golang server receives the request and, through its mechanism and process,

generates the excel billing documentation with the Excelize Golang package, PostgreSQL database access, and SendGrid email service.



The image shows a search interface. At the top, there is a search box containing the text "Type-attribute-search: very tall" and a "Filter" button. Below the search box is a table with the following data:

Created	ID	Name	Type	Status
11.11.2020	123-345-567	Random-name	Very tall	Active

Figure 8 Search engine UI feature

For the remaining tasks that concern the general management of customer data, the coding implementation happened in the Booking service codebase. The need for search engine improvement led me to design the additional searching feature in the UI client of the Booking service client. The improvement was about filtering the search result of the search from the use of special commands in the search box. When the user uses the command to search, the result will be filtered through. (Figure 12.)

4.2.2 Theoretical background and technical information

Go programs are organized *Go packages*. A *Go package* is a compilation of source files in the same directory that are compiled together. Functions, types, variables, and constants defined in one source file are accessible to all other source files within the same package. (24.)

A Go repository typically has only one module found at the root of the repository. A file named *go.mod* there indicates the module path: the import path prefix for all packages within the module. Each module's path not only serves as an import path prefix for its packages but also indicates where the go

command should look to download it. An import path is a string used to import a package. A package's import path is its module path joined with its subdirectory within the module. For example, the module `github.com/360EntSecGroup-Skylar/excelize` contains a package in the directory `excelize/`. That package's import path is `github.com/360EntSecGroup-Skylar/excelize`. Packages in the standard library do not have a module path prefix. (24.)

Excelize is a library written in pure Go and providing a set of functions that allow writing and reading from XLSX files. The library reads and writes XLSX files generated by Microsoft Excel™ 2007 and later. Basic usages from Excelize to implement the billing documentation includes (25.):

- `NewFile`: a function to create a new file with type *File* by default template. (26.)
- `NewSheet`: a function to create a new sheet by given worksheet name. When creating a new XLSX file, the default sheet will be created when you create a new file. (26.)
- `SetCellValue`: a function to set a value of an excel cell.
- `SaveAs`: a function to create or update to an xlsx file at the provided path. (26.)

SendGrid is a cloud-based SMTP provider that allows you to send email without having to maintain email servers. SendGrid manages all of the technical details, from scaling the infrastructure to ISP outreach and reputation monitoring to whitelist services and real-time analytics. SendGrid also provides client libraries in Golang to integrate the email system with an application. (34.)

The `database/sql` is a Go package for using SQL databases from Go. (27.) The package can provide usage such as:

- Database connection.
- SQL query execution.
- Dealing with NULL value.

Package http is a Go package that provides HTTP client and server implementations. The HTTP implementations include some of the common HTTP requests such as Get, Post, Put, and Delete. (28.)

4.2.3 Implementation

The implementation of the Excel billing documentation happened in the Billing service. The implementation required changes to the client and server of the Billing application. In the client of the application, the application was implemented to display a form UI. This form UI registered the user interaction and triggered the generation of the Excel billing document in the Go Server. The Go server received the trigger event from the client through an HTTP request, process the event, and complete its task.

```

import { LitElement, html, customElement, property } from 'lit-element';
import { globals } from './utils/globals';

@customElement('email-form')
export class EmailForm extends LitElement {
  @property()
  email: string = '';

  async sendEmail = () => {
    let req = {email: this.email} as EmailRequest;
    fetch(`${globals.host}/api/v1/excel/email`, {
      method: 'POST',
      body: JSON.stringify(req),
    })
  }

  render() {
    return html`
      <div class="layout horizontal center">
        <input
          label="Sending email "
          name="password"
          autocomplete="off"
          placeholder="Sending email"
          .value=${this.email}
          ?readonly=${!!this.email}
        ></input>
        <button @click=${this.sendEmail}>Send Email</button>
      </div>
    `;
  }
}

interface EmailRequest {
  email?: string;
}

```

Figure 9 lit-HTML client component

The Lit-HTML provides HTML template and directive to let the developer program an UI that could function with some behavior. The form in the client was created by using this provision from lit-HTML to listen to user interaction of typing and form submission (.5). After registering the user interaction for creating a billing document in the client side, the UI form component calls for

the server Excel billing document through HTTP API request to complete the user request.



```
func MakeExcelZip(r *http.Request, w *http.Response) (*bytes.Buffer, string) {
    billings, err := DB.Billing(r)

    f := excelize.NewFile()

    for _, iter := range billings {
        render_style_if(style_bold, i == 0, fmt.Sprintf("%s %s", iter.uuid, iter.name))
        render_style(style_align_right, iter.Value)
        render_style(style_align_right, iter.Vat)
        render_style(style_align_right, iter.Total)
        newline()
    }

    buf, err := f.WriteToBuffer()
    if err != nil {
        httpperr.Error("Failed to generate Excel bytes").BackendError(err).Write(w)
        return nil, ""
    }

    filename := fmt.Sprintf("%s-%s-%s.xlsx",
        time.Now().Format("2006-01-02-1504"),
        string.Replace(billing.Name, " ", "-"),
        string.Replace(billing.excel, " ", "-"),
    )
    return buf, filename
}
```

Figure 10 Document Excel in Server-side

The Go server of the Billing service completes the user request by generating an Excel document through an API request. This API was completed the user request by asking the Go server to create the document from the Billing service database data.

This API listened for the customer request from the client side, using the net/http Go package. The net/http package provides an http handle for the Go server to know the context of the API request. By reading the API request context, the http handle tells the Go server to generate the Excel Document with the context received from the client side. (28.) The context of the API, in this case, lets the server know the Excel document needs to contain the billing data.

To get the billing data and to generate the Excel document for the customer request, the Go server uses database/sql package to retrieve the billing information and renders the Excel document from the data by Excelize Go package. (26.) (27.) (Figure 10.) The compilation for the Excel document, once complete, returns a byte value for the server. Using this byte value, the Server sends the document via email to complete the user request.

The implementation for the search engine improvement happened in the Booking service application. The implementation required creating a Typescript function to filter the result from the search through a typing command. This typing command would support Finnish, English, and Swedish language. For example, suppose the search feature operated in English base on the keyword type-attribute-search. In that case, it would be tyypi-haku in Finnish and Swedish, typ-sökning. (Figure 13.)



```
get_filteredSearch() {
  if (
    this.fitler.startsWith('type-attribute-search: ') ||
    this.fitler.startsWith('tyypi-haku: ') ||
    this.fitler.startsWith('typ-sökning: ')
  ) {
    const search = this.fitler.replace(/(type-attribute-search|tyypi-haku|typ-sökning): /,
    '').toLocaleLowerCase();

    return this.data.filter((d: Data) => {
      return (d.type || []).some((t) => {
        return (
          (t || '').toLowerCase().includes(search)
        );
      });
    });
  } else {
    return this.data.filter(d: Data => d.name.toLowerCase().includes(search))
  }
}
```

Figure 11 Search function with lit-HTML

The Typescript function was created to validate the search string from the typing event and to filter the search result using a JavaScript global function Array.filter(). This function is triggered through the directives concepts of Lit-HTML, allows simultaneous reading, validation, and filtering to achieve the results.

4.2.4 Reflection

Since the works were involving development in different microservice, the microservice concept became clearer for me in practical use and theoretical understanding. I exposed myself to different documentation and implementation within the City Logistics and observing the depth of technology being utilized and optimized to enhance the core features. The Booking and Billing service each had different responsibilities to handle different workloads and performance.

The Excel billing documentation task in the Billing application taught me about the codebase of its service and how the same technologies stack is applied and used across the City Logistics product. The usage of lit-HTML concepts such as directive and HTML template was used in implementation for the client in the development of the feature. It resulted in fast development and interaction for UI component behavior. The benefit of this lightweight client library in the Billing application is performance since the API could contain two thousand bills per day and twenty thousand bills per week. The reading and rendering by using directive and lit-HTML template ease the DOM usage for data binding and rendering.

The usage of the Go Excelize package fits the Billing service context because it provides a comprehensive and programmatic way to render thousands of bills in a systematic manner. During the implementation, I was able to test the performance of the Excel documentation on the local machine data and internal company data. The compilation process for data with two hundred bills from my local data and nine hundred bills from the company internal data was relatively lightweight and fast.

The application of the feature in the future in the City Logistics product is practical for documentation with Excel need. The branches of this application and need are typically conveyed and resulted in official documentation and presentation. Thus, to have a bigger impact of the feature, the accuracy of the

information and the performance of the rendering is the key factor in this development.

For the Booking service, the improvement for the search engine was useful for the customer data use. In terms of UX, it provides a faster way of searching for results. This feature is useful in the Booking microservice because of the data management and search need. The filter search for specific data type uses the specific command and supports a multi-language filter for the same data presentation.

4.3 Thesis entry from the 14th of September to 18th of September 2020

I worked on a different business sector of the City Logistics product this week. This business sector is about monitoring the quality of the transportation service and complaints of the quality from the customers. It is called the Feedback service. The Feedback service is a self-contained application because it has its own database, server, API routes, and client application. And thus, as a microservice, the implementation code about Feedback service is focused only on the management of service quality. (Figure 2.) I also continued further my Excel Billing work in a different branch of applicability for the different billing documentation needs.

The concept of microservice in software development was clearer to me. The Feedback microservice also has its own codebase, similar to the way Booking and Billing microservice have. This codebase of the Feedback application contains a similar technology structure to other services as observed and described in Figure 4. Yet, having the same technologies stack that personalizes for the data structure of the Feedback business strategy, the Feedback application, and the City Logistics in general benefits in productivity and quality of code implementation and server performance. (17.)

4.3.1 Research and study

Feedback microservice, in comparison with other services, was young and lack of applicability. Because it was newly developed, the code quality and features are relatively simple and not yet reliable for commercial use. The tasks I received for the Feedback service were to make an additional refinement to improve its commercial value before the presentation and demo to our customer. This refinement for the Feedback application required the use of the Vaadin component in our client UI presentation with Lit-HTML. It also required me to be familiar with more advanced programming concepts about this application, such as its data structure and the use of Enums in software development.

The Billing application also received an additional request for Excel billing documentation use. Because of this need, I continued my knowledge and skills learned from the previous week of developing the Excel billing documentation into a new branch of billing need.

4.3.2 Theoretical background and technical information

Vaadin is a framework for developing reactive client-side web apps. Views are web components built with Typescript and lit-HTML. The framework is lightweight and performant due to its compatibility with the lit-HTML library. It also provides UI components encapsulate with useful functionality and a read-made theme for data management usage.

Grid is the Vaadin component that is used in the implementation of the feature for improvement in data presentation. The UI components are compatible with lit-HTML and TypeScript, provides a definitive and functional way of presenting Array data for client UI. Each item in the items array assigned to the <vaadin-grid> represents a single row in the grid. (29.)

Enums, in computer science, is called enumerated type. It is a data type consisting of a set of named values called elements, members, enumerals, or

enumerators of the type. Enums allow a developer to define a set of named constants. Using enums can make it easier to document intent or create a set of distinct cases. TypeScript provides both numeric and string-based enums. (30.)

4.3.3 Feedback service

To present the service quality and complaint data in the Feedback client, the UI was constructed by rendering the Grid UI provided by the Vaadin component. This Grid UI contains useful data assertion features such as sorting, filtering, and reading of data context. The feature usage of the Grid component helps the developer to create UI with creative behavior that fits the management context of the Feedback application.

By building a UI component from the use of the sorting and filtering function of the Grid component, the UI was able to display a graphical reading of the service complaint data. These functions helped the implementation in reading the complaint data structure and in constructing the UI behaviors and styling.

The assignment in the Feedback codebase also extended my general understanding and improvement of the data structure of the service. I was learning and adding enums data type for the data structure to improve the definitive reading of the complaint data structure. This improvement resulted in faster rendering and development for the lit-HTML, enforced with strong typing.

4.3.4 Billing Microservice

I spent my following half of this week for the billing microservice to continue with a new Excel billing documentation. The feature was about generating an excel document with all the billing data from a month.

The general implementation of the feature is similar to previous Excel billing documentation implementation. It required the client of the Billing service to provide an UI where the customer could interact with. When the customer interacts with the UI, the client sent the API request for the server for generating the Excel documentation with a month data. The technology concepts involved

were the use of Go package net/http and database/sql for database processing. The Excel compilation was used by the Go Excelize package for the reliability in rendering and compilation of large data amounts.

The difference to this Excel Billing documentation feature was about the amount of data being processed and compiled by the Go server. Since the amount of data is a month of bills, the API was required to provide meaningful information of time range so that the server could access the PostgreSQL database and proceed with the compilation. Within the API, the server application would query from the PostgreSQL database for the data. The data, upon which being fetched and serialized into an interface, was used to generate the billing document's content programmatically.

When the generating content was complete, the Go server named the file and returned the buffer data of the excel file. The buffer and the file name were then set as a server response attachment with the body of binary and spreadsheet type to make the file downloadable upon the API call.

4.3.5 Reflection

My thoughts on the work being done during the week are reflective of the thesis theme. The microservice architecture divides the City Logistics core feature into different codebases that utilize the same technology stack to achieve a different technical purpose and commercial results.

My work this week involved the development of the Feedback service helped me observed the benefits of the City Logistics code organization. I was able to recycle the concepts of lit-HTML in the technical context of the Feedback application. The common knowledge of lit-HTML, such as directives, was exhaustively utilized in the Feedback client application. The feedback information needed to be presented as a listing, and my implementation for lit-HTML was adapting well to the data structure.

My knowledge and skill of TypeScript in client development were also heightened by advanced concepts and applicability. I was able to use and test my understanding of enums in TypeScript to develop a type-safe data type for the client application. This type-safe usage of enums was developed more in the feedback data structure, and it prevented crashes from unexpected value and unknown type.

The enums data type usage in the feature is useful in general software development for designing a good data structure. The future extension for enums applicability is for using the constant value of the enums in comparison situations. Since the value and type of the enum are unchanged and immutable, the comparison for client feature and function is reliable to produce a reliable Boolean comparison result.

For my implementation of the Excel billing documentation in the Billing service, I was able to re-use the previous knowledge with a new branch of application and need. The usage of the feature for the current implementation was heavy-loaded due to the time range difference. I was able to produce a desirable performance for a month data from my local machine data and the company internal data. The struggle of the implementation was to have complete accuracy in the billing, as the documentation was for official use. I was able to produce a complete accuracy in my local machine, but I met with some duplication data problem with the company internal data.

4.4 Thesis entry on the week from 21st of September to 25th of September 2020

During this week, I spent my time developing new features for the feedback microservice. My task was about making a correct display of summary information regarding the complaint and rating of our City Logistics product.

4.4.1 Research and study

Since the tasks I received were mainly about service quality management and monitors, the implementation was coded in the Feedback service codebase.

Creating a display of rating and complaint summary information required implementation in the client UI and Go server of the Feedback application. The code implementation required computation for the summary data from the database, and displaying that computation results in the client UI through an API. The computation was handled by using the provided function from PostgreSQL, and the application read the computation results by using Golang struct data type and TypeScript interface data type.

4.4.2 Theoretical background and technical information

A TypeScript Interface is a structure that defines the contract in the TypeScript application. It defines the syntax for classes to follow. Classes that are derived from an interface must follow the structure provided by their interface. An interface is defined with the keyword `interface`, and it can include properties and method declarations using a function or an arrow function. (31.)

A Golang struct (short for "structure") is a collection of data fields with declared data types. Golang has the ability to declare and create its own data types by combining one or more types, including both built-in and user-defined types. Each data field in a struct is declared with a known type, which could be a built-in type or another user-defined type. Structs are also considered as a template for creating a data record, like an employee record or an e-commerce product. (32.)

The `AVG()` is an aggregate function in PostgreSQL. The `AVG()` function allows calculating the average value of a set. `COALESCE` is a conditional expression function. The `COALESCE` function evaluates arguments from left to right until it finds the first non-null argument.

4.4.3 Implementation

In the Typescript application, I developed an API function that handled the GET HTTP method for the component and a customized interface that matched the JSON structure from the Golang interface structure. When the API function was called, the parsing of the value by name would be matched. The API would be used in the parent component to fetch the server's summary data when the user went to the dashboard view of the application or login and was redirected to the dashboard. The parent recognized the category of the current view, passing through the children component the summary data and the current view enums so that the children component could render the summary of the data UI base on which category of the dashboard was being viewed.

The API, being recognized in the server, would request it to make a set of serialization in order to obtain the summary data, which is of the following:

- The total and average rating of the booking microservice in the previous week and month,
- The total and average of positive and negative feedback of the system in the same duration
- The total and average amount of policy violation during the same period and how long the violation claim being processed.

From the HTTP handler, the server resolved the summary data from two sources, from the internal microservice database and through RPC connection API. When fetched from these sources, the resolved summary data would be compiled into an interface and serialized into JSON data to send to the client for API consumption.

For the internal microservice database source, the backend server requested PostgreSQL to send the total and average counting of the negative and positive feedback and violation through the SQL script and scan the result to the summary data interface.


```

func (db *Database) GetSummary() (Summary, error) {
    stmt := postgres.SF("SELECT")

    stmt.Statement += fmt.Sprintf(
        "(SELECT count(*) AS total_critical_data_month FROM data WHERE category in (%d, %d) AND EXTRACT(MONTH FROM created) = EXTRACT(MONTH FROM NOW)::date - interval '1 month')",
        enums_data.Critical,
        enums_data.Serious)

    stmt.Statement += fmt.Sprintf(
        "(SELECT count(*) AS total_critical_data_week FROM data WHERE category in (%d, %d) AND EXTRACT(WEEK FROM created) = EXTRACT(WEEK FROM NOW)::date - interval '1 week')",
        enums_data.Critical,
        enums_data.Serious)

    stmt.Statement += fmt.Sprintf("(SELECT COALESCE(avg(rating), 0) AS system_average_month FROM system WHERE EXTRACT(MONTH FROM created) = EXTRACT(MONTH FROM NOW)::date - interval '1 month')",)

    stmt.Statement += fmt.Sprintf("(SELECT COALESCE(avg(rating), 0) AS system_average_week FROM system WHERE EXTRACT(WEEK FROM created) = EXTRACT(WEEK FROM NOW)::date - interval '1 week')",)

    summary := Summary{}

    err := stmt.QueryDestError(db.postgre, &summary)
    if err != nil {
        Log.Error("GetSummary() Error - %s ", err.Error())
        return Summary{}, err
    }
    return summary, nil
}

```

Figure 12 Server computes PostgreSQL query

In Figure 14, The Go code demonstrated how the server requested the data from the SQL script and scanned the results to the Go summary data interface. Most of the functionality, such as filtering and calculating the total and average amount of data, was used by the SQL function to enhance the overall readability and reduce the performance load of the Go Server calculation, even though the load was not subtly stressful. However, the scanning could fail because the average AVG() function calculation from PostgreSQL could return nil value when the calculation value was empty, so using COALESCE() acting as the safety net returns 0 edge case happened. Furthermore, since the time value of the summary data was the timestamp, the filter was being compared and handled inside the PostgreSQL database. The comparison extracted the month and week values of the data. It validated with the previous month and week value from the current timestamp to filter for the expected data within the filtered time range.

4.4.4 Reflection

The task for the week happened in the Feedback service with the need to display the summary information about the service of City Logistics product. To complete the task, I created a PostgreSQL computation query to get the summary data from the Feedback service database. The summary data is read by the Go server and served to the client through an API request. The client UI consumes the API with the UI component to display the dynamic data from the database.

The success of the task is about having an accurate computation for the summary data. To produce an accurate result, I chose to make the computation in the PostgreSQL database and return the results for the server for API use. By using an aggregate function and conditional expressions to create the computation query, the results were crash-proof for the Golang server and fast.

The applicability of the implementation is the knowledge of making computation data in the PostgreSQL database over the Go server. Computation in PostgreSQL is faster in Golang because of the large amount of information in a month time range. The computation cost presents when the data amounts to twenty-thousand units. In the Golang server implementation, the computation executes in the server once the data is fetched from the database. When using computation in the PostgreSQL database over in Golang server, the server can skip the computation and only be concerned about reading the data.

The other success point of the feature was also to construct the API for the server to serve the data for client UI consumption. I was able to use Golang knowledge with Go package `net/http` and `database/sql` to achieve the results. The data in the client was able to render dynamically with computation results from the database.

4.5 Thesis entry on the week from 28th of September to 2nd of October 2020

In the fifth week, I continued my programming work on the Feedback service and Billing service. My work on the Feedback during this week was about making an automation no-reply that will be sent according to the boolean status of the Feedback data. For the Billing service, I developed language support for the Excel document billing.

4.5.1 Theoretical background and technical information

Dynamic email template is an HTML emails with design and appearance. The adoption of dynamic email provides a way to display informative data over regular text-only emails. SendGrid provides basic instruction in using a dynamic email template (34.):

- Create an API key.
- Design the dynamic template.
- Send the email using the SendGrid Go package.

Environment variables are variables in the local machine system that describe the machine environment. Aside from 'built-in' variables, environment variables can also be created independently. Since they are bound to the environment, they are great for things such as API access tokens. The variable can be set to one value on the development machine and another in the production environment without having if-statements or special config files. (33.)

4.5.2 No-reply email implementation



```
func sendEmail(feedback_uuid, language, template string) error {  
    email_body := CriticalEmailTemplateBody(feedback_uuid, language, template)  
    template_id := feedbackEmailTemplateID(language)  
  
    Log.Info("Template ID: %s", template_id)  
  
    m := mail.NewV3Mail()  
  
    m.SetFrom(&email_body.Sender)  
    m.SetTemplateID(template_id)  
    m.AddPersonalizations(&email_body.Personalization)  
  
    c := sendgrid.NewSendClient(Config.Sendgrid.Key)  
  
    return checkSendGridError(c.Send(m))  
}
```

Figure 13 Email functionality with SendGrid

First of all, I integrated SendGrid into the application server. The integration work would produce a no-reply email template in English, Swedish, and Finnish. These email templates would include dynamic data placeholders. The server uses this dynamic data placeholder to populate feedback information from the client. Afterward, on each API request sent to our server of the feedback information, the server would read the context of the API. If the reading from the API results in the data information with a Boolean status as truthy, the server will send an email automatically. (Figure 15.)

The integration also involved a setup of the SendGrid service into our application. This setup is a way to secure fetching and storing the API key and API template in order to securely use the SendGrid service. To make the setup functional, the API key was stored as an environment variable, and the API template was stored inside a YAML file. Putting API credentials in an environment variable prevents exposure risks. This sensitive data is only

available for use by the fetching mechanism developed in the server. (Figure 15.)

4.5.3 Excel billing document implementation

The implementation of the Excel billing document for this feature happened in the Billing service codebase. This implementation is similar to the previous billing document implementation. The bill was generated in the server through an API by using the Go package Excelize. The data of the bill is fetched from the database by Go package database/sql to populate into the Excel billing document.

Since the billing needs to support multi-language formats, additional changes were needed in the billing API. The API received in the Go server provides context about the language format for the billing document. From this language format context, the server generates the language content of the document by using the i18n (internalization) package, a client and server library to support multi-language content. This package allows the rendering of the billing content in the language that the server requested. (9.) (10.)

4.5.4 Reflection

My work for the week including the development tasks for both the Feedback and Billing service. The Feedback service task was able to complete by having a no-reply email system being integrated into the service. For the Billing service, the improvement for the Excel billing document was able to support multi-language content.

The no-reply email system implementation in the Feedback service produced a favorable result. The key factors of the result come from developing the server to have a secure integration to use the SendGrid service and to send the no-reply email from the Boolean status of the data structure. The integration was achieved securely by developing accurate and dynamic reading for the API key from the Golang server through the environment variable. The setup and

integration also resulted in supporting the no-reply email content available with different language need and with data being dynamically populated to the email content.

The implementation in the Billing service is about creating language-support Excel billing documents. This implementation was achieved by having additional improvement for the Excel billing API. The API received from the Golang server was able to provide a meaningful context of the human language need. This need is read by the Golang server to generate the content according to the API context by using the i18n (internationalization) package. The end results are the Excel billing documents are available in English, Finnish and Swedish language.

The applicability of the feature lay on the ability to provide multi-language content from the Golang server. By successfully using the i18n package in the Golang server, the server could provide dynamic language content. This applicability is meaningful because the branches of potential development could be the support of providing a language-based PDF billing document and report or having a Word document report with language support.

4.6 Thesis entry on the week from 5th of October to 9th of October 2020

For this week, my task was about creating a two-way data interaction between the Feedback service and the Billing service. Specifically, The Billing service needs to display Feedback information from the Feedback database. In addition to displaying the data, the Billing service could create new data for the Feedback service in the Billing client. The implementation required the use of gRPC, Google protocol buffers, and Google protocol buffers.

4.6.1 Theoretical background and technical information

Remote Procedure Call

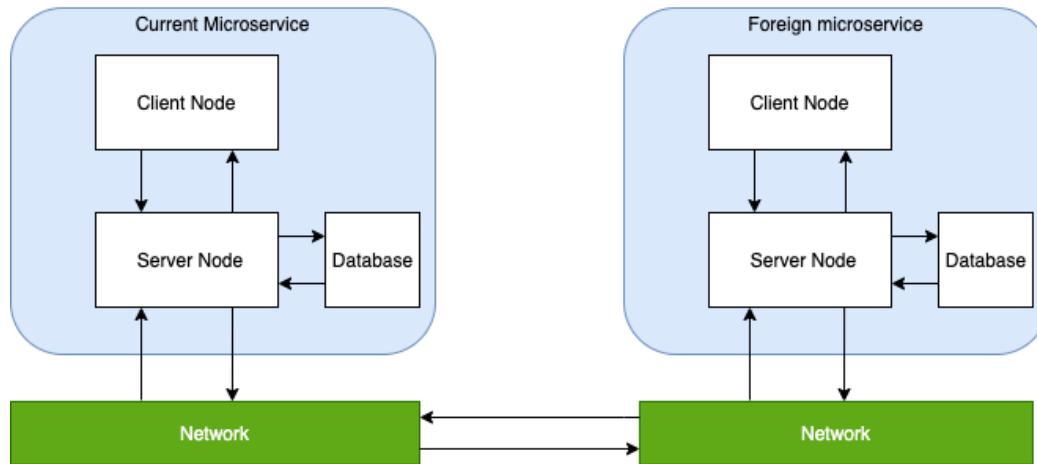


Figure 14 Remote Procedure Control

From official Golang documentation (6.), RPC is a go package "provides access to an object's exported methods across a network or other I/O connection. After registration, exported methods of the object will be accessible remotely. A server may register multiple objects (services) of different types, but it is an error to register multiple objects of the same type".

The other concept of development that I must be acquainted with during the development of the feature was the Google protocol buffer. In general, it is Google way of serializing structure data in a small, fast, and simple mechanism (8.). The developer defined the characteristics and behaviors of the data through a protocol syntax was. The data definition was then used to generate a source code of the defined structured data to and from various data languages.

4.6.2 Research and study

The Billing client requires database access to the Feedback service to display data and to make changes to the Feedback data source. This database access to the Feedback database is provided by a protocol context from the RPC

connection. This protocol context could be a request for retrieving data or for creating new data. The protocol context includes the Golang struct or the data structure of the Feedback service. This structure is generated by the protocol definition of Google Protocol Buffers. (11.)

The development of the gRPC function includes:

1. Create the RPC connection of the Feedback service in the Billing service.
2. Serve the data by retrieving and creating requests by using the RPC function.



```
server := application.New_Server(  
    rpc,  
    mhttp.NewTwirpHooks(Framework.JWT.TwirpAuthenticationValidator),  
)  
  
a.Router.HandleFunc(server.PathPrefix(), a.TwirpWrap(server,  
    Framework.JWT.AuthenticateTwirp))
```

Figure 15 Creating RPC Server Simplified Code

The Feedback service provides the gRPC connection for Billing client use to receive the Feedback database access. The Billing service receives the Feedback data in the server and serves the Feedback data in its client application. The client application uses the Feedback data from the API to render UI interaction.

In the sending data implementation, the data first received in the Billing server and compiled as a caller request. The request is sent to the Feedback service server after being compiled in the Billing server. The Feedback server

processes the Billing data with validations and stores data in the Feedback database.

4.6.3 Implementation

The implementation of the task started with writing a protocol definition for the RPC connection. The protocol definition describes the Feedback service data structure. The definition also includes the type of database connection to the Feedback service and provides to the Billing service GET and CREATE change to the Feedback service database. Once the writing of the protocol is complete, the Google Protocol Buffers generate the definition into Golang code.

The Billing server uses the definition in the Golang code to read the Feedback service data structure. It also allows the Billing service to make database changes in the Feedback service through this Golang code definition. The Billing server reads the Golang definition of the Feedback data and serves the function to its client through an API. The client of the Billing service receives and makes changes to the Feedback database through this API. (Figure 19.)

```
service MicroService {
  rpc CreateData(DataReq) returns (DataRes);
}

message DataReq {
  int32 data_index = 1;
}

message DataRes {
  string uuid = 1;
  string text_data = 2;
  google.protobuf.Timestamp created = 7;
  google.protobuf.Timestamp updated = 8;
}

message Data {
  string uuid = 1;
  string text_data = 4;
  google.protobuf.Timestamp created = 7;
  google.protobuf.Timestamp updated = 8;
}
```

Figure 16 Protocol Buffers Process Simplified

4.6.4 Reflection

Feedback and Billing service are two self-contained applications. Because of using gRPC, the Billing service does not need to create a database about Feedback data. The Feedback data is provided to the Billing service through gRPC data streaming. This data streaming lets the Billing service have access to the Feedback database and data structure.

This access ensured the code quality because the Feedback data structure access provided to the Billing service is consistent in data type and value. Since the Google protocol buffers create this API definition for client and server in TypeScript and Golang, a software solution was not limited to a use case and the schema of the database tables.

The task was successful in providing this gRPC access for the Billing service. In the implementation, I was able to create a bidirectional steaming of Feedback

data for the Billing service. This bidirectional streaming allows the Billing service to connect its client to the Feedback server and database.

Another practical use of the protocol is its support for the English, Finnish, and Swedish languages. With the use of `i18n` (internalization) (9.) (10.) in the client, the protocol helped to define the enums as a constant variable. Translation works and data flowed from the client to the database were no longer relying on a string but a number by value. The use could be to interpret language text with `i18n` for the client UI display and translate the text value regardless of the language base on the enums generated and serialized by the protocol files.

The practicality of protocol had been very useful in reducing the data structure's guesswork and reducing unnecessary DRY code, allowing the developer to be more focused on development.

4.7 Thesis entry on the week of 16th of October to 27th of October 2020

This week was the final week of the thesis work and writing. For the weeks, my task was about making a code quality improvement on the Feedback service. The improvement for code quality was implemented in the client and server with TypeScript and Golang programming language.

4.7.1 Research and study

The leading reasons for the refactoring work had been the accumulation of technical debts in the codebase and the database structure. The requested change also affected many APIs that both client and server feature of the microservice application depends. The previous week, a meeting was being held to discuss the software implementation's technical limitation from the database to client use. A decision for change was made with the presence of a lead developer and a project manager.

The columns of the database, for example, are storing data as text values for reference data instead of UUID, which increased the difficulty in indexing data for microservice and database tables. In the database table, data of different

users were stored in one table of the database, magnifying the problem of magic numbers. Also, some database tables were redundant and could be merged as a property of a table as JSONB data type (12.) to reduce the work of data indexing.

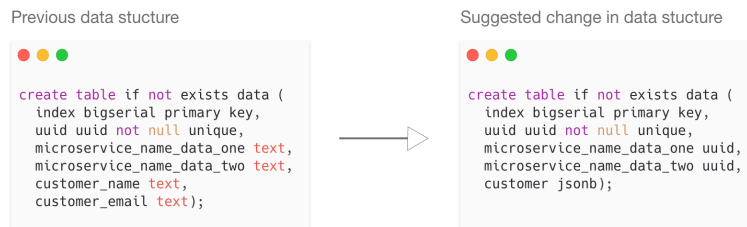


Figure 17 Database structure improvement

In other places of the microservice codebase, server-side feature for using data was also primarily affected and impacted by data structure definition and database functions tagged by the data structure. The impact touched many API functionalities, changing the way of how the backend server functioning. The change from the API, in turn, led to the change of the client code.

The change of improvement to the database of the microservice and any cascading functionality was acceptable. The application was yet released for commercial use, so implementing change did not impact City Logistics Product.

4.7.2 Theoretical background and technical information

JSONB is a data type of PostgreSQL and is stored in a decomposed binary format. Storing JSON data in PostgreSQL database as JSONB type provides fast processing but adding conversion overhead to input. JSONB also supports different operators such as getting object by key, by index, and at the specified path. (12.)

A database index is a data structure that improves the speed of data retrieval operations on a database table at the cost of additional writes and storage

space to maintain the index data structure. Indexes are used to quickly locate data without having to search every row in a database table every time a database table is accessed. Indexes can be created using one or more columns of a database table, providing the basis for both rapid random lookups and efficient access of ordered records. (35.)

4.7.3 Implementation

The implementation work started with the database changes of the Feedback service. The changes in database structure were defined by the business strategy of the Feedback service. The change would include indexing columns, time record columns, and Feedback service columns. The indexing columns are Index and Uuid data type, and the time record columns are timestamp data type. (Figure 21.) The Index and Uuid data types provide a reference method in making queries for a single or cluster of information. The timestamp record column provides a systematic filter for retrieving the data on the timestamp unit.

The table also contains other central information about the table. This information is defined as the central data column of the table. The value and data type of these columns are determined by the technical application and business strategy of the Feedback service. For the dynamic column information, the integer data type is used because the column provides Enums interpretation in the Go server and TypeScript client with a constant value. For static column information, JSONB data type is used to store information from the Go server as a snapshot. This snapshot of JSON data provides the read-only replication of information.

The change in the Feedback service database also resulted in changes for the API of the Golang server and TypeScript client. The changes in the Golang server and client API were about the modification to the table data changes in data type. The Feedback database table provides Uuid and index data types for indexing and reference. The retrieval and modification of the data from the Go server were able to make fast and performant queries on this data type as a

result. This improvement on reading the database results in the server provides a robust way for client applications to consume the data with consistency.

4.7.4 Reflection

The work of this week was able to connect to the overall theme of the Thesis topic. Services of the City Logistics are developed in the microservice architecture to provide an optimal technical solution for the services' aspect. Therefore, a self-contained development environment of services includes a personalized collection of database tables, client applications, and server API.

The change and improvement introduced in the Feedback service were able to bring the technical infrastructure to focus on the Feedback business strategy. The technical changes include database tables, server API, and client implementation.

```
http.Handle("/api/v1/datas", http.ValidateGET, authenticateAPI, authorize, handleData)

func handleDatas(r *http.Request, w *http.Response) {

    filter := &Filters{
        Status:      DataStatusList{},
        Pagination:   r.QueryBoolDefault("pagination", false),
        TimeSlot:     r.QueryInt("timeslot", 3),
        CustomerUuid: r.Query("customer_uuid"),
        Limit:        r.QueryInt("page_limit", 25),
        Cursor:       r.Query("cursor"),
    }

    if status, err := r.QueryIntList("status", ","); err != nil {
        httperr.BadRequest("invalid query").BackendError(err).Write(w)
        return
    } else if len(status) > 0 {
        for _, s := range status {
            filter.Status = append(filter.Status, dataStatus(s))
        }
    }

    data_paginations, err := DB.Datas(filter)

    if err != nil {
        httperr.BadRequest(err.Error()).Write(w)
        return
    }

    w.WriteBodyJSON(data_paginations)
}
```

Figure 22 HTTP Handlers for data list

One example of the efficiency from the infrastructure change is the stability in developing Golang server API. The use case for this example is about data

listing for API consumption. The changes in database level provide consistency in reading data structure information across the Feedback application with good indexing and time record. With the data type change that supports dynamic data such as Enums, the client implementation with TypeScript and Golang, therefore, are type-enforced and predictable. This predictability also prevents server and client applications from crashing because of reading a contradict type value.

CONCLUSION

The thesis started on the date 31st of August and ends on the date 27th of October, lasted about eight working weeks of my employment at Attracs Oy Ab.

During my thesis work, I developed my programming language skills of Typescript, Golang, PostgreSQL and various other libraries as part of my development work and responsibility at Attracs Oy Ab. I became more integral with the development team in developing the client and the server on four different microservices of Attracs Ab Oy City Logistics Product and Android, IOS, Web application powered by Server-Side development.

I was involved with several server feature tasks, some web and mobile application changes, and bug fixes for the on-production-deployment microservices. These tasks were my introduction to Attracs City Logistics Software Development. I gradually started to pick up the Go syntax for the server-side development and progressively adapted to the Attracs development standard on Go and Typescript development standards.

Even though features had been used in the production environment, I was later corrected by a senior colleague that there is room for server code improvement in terms of readability and performance-wise. This correction resulted in some code change to increase the developer experience of the microservice repository for the team. My microservice tasks were about one month and a week working on bug fixes to medium and large-scale features, where it happened on the server and client-side.

I was also active in the development of one internal deployment microservice. The responsibility ranges from the design and planning to implementation of Golang server-side API, Typescript User Interface and client functionality, and the data structure PostgreSQL database, including the software architecture of the microservice with Google Protocol Buffers.

The development work for this microservice ended on a good note with feature weekly developed and demo for the municipal client on our nightly deployment. It resulted in several ten different features, from small to large, being delivered agilely with agile results.

Starting with little knowledge and skills in Go, the thesis works ended with productive work produced with newfound experience. The results were fruitful in Go development on the server-side, Typescript for the client development, and the server interaction of Go to the PostgreSQL database. More high-level programming work on the general design of data structure microservice feature.

REFERENCES

1. Statistics Finland, Number of persons aged 70 or over 874,000. Date of retrieval 20.04.2021
https://www.stat.fi/til/vaerak/2019/vaerak_2019_2020-03-24_tie_001_en.html#:~:text=According%20to%20Statistics%20Finland%27s%20statistics,by%20100%2C000%20in%20three%20years.
2. Findicator 2020, Age Structure of population. Date of retrieval 01.11.2020
<https://findikaattori.fi/en/table/14>
3. Statistics Finland, 2017, Inactive population, Date of retrieval 01.11.2020
https://www.stat.fi/til/tyti/2016/13/tyti_2016_13_2017-04-12_kat_004_en.html
4. Mobile Usability
<https://medium.com/@cayetanogros/mobility-usability-dc0bc61d05c1>
5. Writing templates 2017, lit-HTML guide, Date of retrieval 05.11.2020
<https://lit-html.polymer-project.org/guide/writing-templates>
6. Package RPC 2020, Golang documentations by Google, Date of retrieval 16.11.2020
<https://golang.org/pkg/net/rpc/>
7. Protocol Buffers, Protocol Buffers documentation by Google, Date of retrieval 16.11.2020
<https://grpc.io/docs/what-is-grpc/introduction/>

8. Protocol Buffers, Protocol Buffers documentation by Google, Date of retrieval 16.11.2020
<https://developers.google.com/protocol-buffers>
9. Internalization and Localization, Wikipedia, Date of retrieval 16.11.2020
https://en.wikipedia.org/wiki/Internationalization_and_localization
10. i18n, i18n npm documentation, Date of retrieval 16.11.2020
<https://www.npmjs.com/package/i18n>
11. Defining services, Protocol Buffers documentation, Date of retrieval 17.11.2020
<https://developers.google.com/protocol-buffers/docs/proto3#services>
12. PostgreSQL, JSON Functions and Operators, Date of retrieval 30.11.2020
<https://www.postgresql.org/docs/9.6/functions-json.html>
13. What is Agile Software Development, ClickUp, Date of retrieval 27.01.2021
<https://clickup.com/blog/agile/agile-software-development/>
14. Agile Software Development, Wikipedia, Date of retrieval 27.01.2021
https://en.wikipedia.org/wiki/Agile_software_development
15. Henkilöliikenteen välityspalvelut, FCG Finnish Consulting Group Oy, Date of retrieval 27.01.2021,
<https://www.fcg.fi/kestava-elinymparisto/kaupunkisuunnittelu-ja-kehitys/henkiloliikenteen-valityspalvelut>
16. Package excelize, GoDoc, Date of retrieval 27.01.2021
<https://godoc.org/github.com/360EntSecGroup-Skylar/excelize>

17. Microservices, Wikipedia, Date of retrieval 14.02.2021
<https://en.wikipedia.org/wiki/Microservices>

18. Typescripts, Wikipedia, Date of retrieval 14.02.2021
<https://en.wikipedia.org/wiki/TypeScript>

19. Front-end Web Development, Wikipedia, Date of retrieval 14.02.2021
https://en.wikipedia.org/wiki/Front-end_web_development

20. Apache Cordova, Wikipedia, Date of retrieval 14.02.2021
https://en.wikipedia.org/wiki/Apache_Cordova

21. Golang programming language, Wikipedia, Date of retrieval 14.02.2021
[https://en.wikipedia.org/wiki/Go_\(programming_language\)](https://en.wikipedia.org/wiki/Go_(programming_language))

22. Apache Cordova, Wikipedia, Date of retrieval 14.02.2021
<https://lit-html.polymer-project.org/guide/creating-directives>

23. What is Hybrid App Development, Chris Griffith, Date of retrieval 14.02.2021
<https://ionic.io/resources/articles/what-is-hybrid-app-development>

24. How to write Go Code, Godoc, Date of retrieval 16.02.2021
<https://golang.org/doc/code#Organization>

25. Excelize, 360EntSecGroup-Skyler, Date of retrieval 16.02.2021
<https://pkg.go.dev/github.com/360entsecgroup-skyler/excelize>

26. Excelize, 360EntSecGroup-Skyler, Date of retrieval 16.02.2021
<https://github.com/360EntSecGroup-Skyler/excelize>

27. SQLInterface, Golang Wiki, Date of retrieval 19.02.2021
<https://github.com/golang/go/wiki/SQLInterface>
28. SQLInterface, Golang Wiki, Date of retrieval 19.02.2021
<https://golang.org/pkg/net/http/>
29. Grid, Vaadin Fusion, Date of retrieval 19.02.2021
<https://vaadin.com/components/vaadin-grid/html-examples>
30. Enums, The TypeScript handbook, Date of retrieval 19.02.2021
<https://www.typescriptlang.org/docs/handbook/enums.html>
31. TypeScript Interfaces, Tutorial Teacher, Date of retrieval 20.02.2021
<https://www.tutorialsteacher.com/typescript/typescript-interface>
32. Golang struct, Golang programs, Date of retrieval 20.02.2021
<https://www.golangprograms.com/go-language/struct.html>
33. How to set environments variable, Dominik Kundel from Twilio, Date of retrieval 20.02.2021
<https://www.twilio.com/blog/2017/01/how-to-set-environment-variables.html>
34. How to send an email with dynamic transactional templates, SendGrid, Date of retrieval 20.02.2021
<https://sendgrid.com/docs/ui/sending-email/how-to-send-an-email-with-dynamic-transactional-templates/>
35. Database Index, Wikipedia, Date of retrieval 20.02.2021
https://en.wikipedia.org/wiki/Database_index

