# Samarit

**Computer Systems with Project Work (1DT003)**

Project proposal for team Ilumise:
Anton Augustsson, Anton Marhold, Douglas Gådin, Jacob Luup, Jakob Wallén, Jonas Teglund, Morgan Räsänen

Datum: 2021-03-26

# 1. Introduction

Some of the biggest victims of this pandemic are the elderly, since they cannot leave their homes without fear of getting infected. It is our intent to help them with their needs in a covid safe manner. We want to offer a platform where elderly people can get in contact with people who want to help. This can be with anything i.e. shopping groceries or picking up a package at the post office. On the other hand we have a lot of people that are losing their jobs during this pandemic and these people will get a platform where they can help people in need and make money at the same time. Although the latter isn't our main focus, it is still a problem that our platform will solve.
During this project we hope to learn app development, database management and interacting as well as creating API:s. Furthermore, we want to learn how to create a large system in a limited amount of time.

## Similar systems

There are several similar solutions, such as Foodora, ICA, Ryska posten, home delivery and Vollo. However most of these only provide services for specific niches, while our solution works any kind of services.
A company which provides a variety of services is Vollo. The difference between our system and Vollo is that anyone with our app can be a service provider. There is no recruiting process, which has the advantage of more service providers and where services can be performed at any time which is not possible with Vollo.
This however comes with an issue, how can our service providers be trusted? We solve this problem by verifying the service provider’s identity and linking it to an account. If a service provider acts unreasonably they could be banned, or reported to proper authorities. Therefore it should not be an issue.

## Challenges

The main challenges will be synchronizing and matching service requests to service providers. We are also using many new programs, languages and API:s. Getting everything to work properly together will be a challenge. We also have a lot of security concerns that may be taken advantage of. Making sure there is no way to cheat users will be another big challenge for us. As the elderly already have trouble with technology, it will be tough to implement a system they can easily use.

## Limitations

The biggest limitation we face is the short time frame. We also cannot use services that require payment fees since we have no funding. Furthermore, we have to consider the time to implement different features to see if we need to scrap some ideas or if we have more time left, to implement new ideas. Our lack of knowledge regarding some of the chosen technologies might introduce unforeseen complications during our project. There is much to learn before we can start producing something concrete.

# 2. Technologies & methods

## Maps and location

We want to implement a map where service requests are shown to service providers on a map. We will use google maps api or similar to implement this. The map will also allow us to filter orders by locations and only show requests local to service providers.

## Authentication

First and foremost we will implement authentication via google sign in. If we have time and motivation we will explore various other authentication methods, like social media, sign in with facebook, instagram etc. The reason why we want more is to provide a variety of authentication methods to make our application available to more people.
We will also require further authentication with BankID where we deem it necessary. We chose BankID because it is widely used for identification and authentication in our society. It is reliable, legally binding and most people have it.

## Languages and frameworks

We believe that the elderly in general are less likely to want to use smartphones. For this reason, we aim to create both a mobile application as well as a web page application, thus providing more than one choice of device. React Native fits this purpose since it offers great opportunity for cross platform development. For this reason, we have opted to use React Native as our front end language.

For the back end we will use node.js to build the Rest API that all clients can connect to. The reason node.js is suitable is that it is scalable and is made to handle http requests with efficiency and ease. Our main concern with node.js is that it is weakly typed, which means that it is easier to let bugs slip by that may be hard to pinpoint down the road. We believe that proper testing and documentation will avoid such situations. We plan to use JTest for testing.

### Database

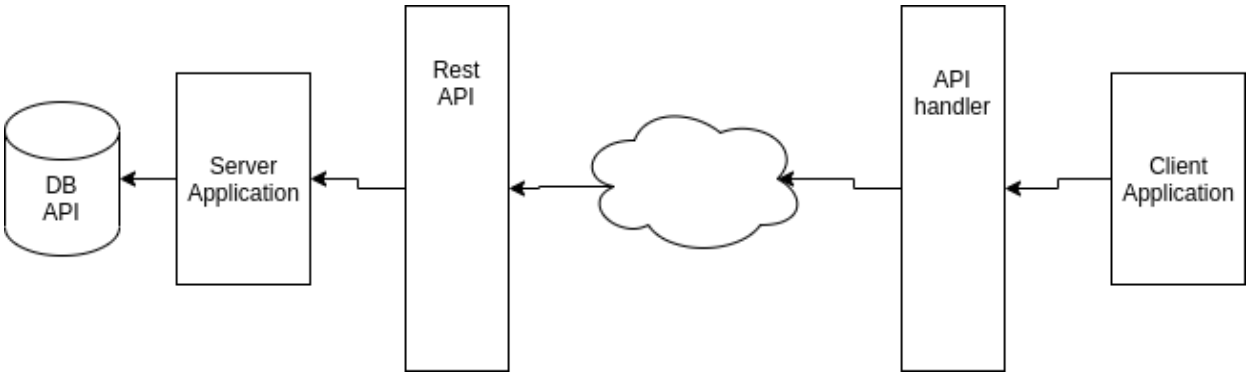
We choose MongoDB for our database because it is a flexible client/server database. With flexible we mean it can store almost any data we want. The data is stored in bson documents (JSON-style data structure), which we thought would have high compatibility with Node.js. The drawback of MongoDB is that it has large memory usage, which increases with data size. However, this will not result in issues since we will not save much information. We also considered firebase, but that would require us to use google servers which would compromise integrity. We did not want to handle that.

### Communication

The modules that will be used for live communication is socket.io which is a websocket framework for node.js. The reason we won't use Rest API for live update and chat functions is that we would have to use polling to constantly check if something has updated. However requests are useful in some scenarios when we for instance create a new request. Therefore a Rest API will be used for such communication.

## 3. System architecture

In a large system the biggest issue is to make a clear structure and readable code. Therefore we will create isolated modules to separate the functionality and create interfaces in order to create an abstraction of the other modules. This is achieved by dividing the system into five parts, on the client side: client application and api handler. On the server side we have a rest api, server application and a database api.



## Client

The client consists of two parts, the api handler and the client application. The api handler will be testable, however the client application will not provide an easy way to be tested with automatic tests. It should be tested by real persons. The client application will be written in react native in order to support cross platforms; android, ios and the web.

### Client application

There are two views. One for the requester and one for the service provider. When opening the application the requester view will be shown because most people will be a requester and it will be easier for the elderly. Then in the menu there will be a button for showing the servicer view. The application should be user friendly and suited for elderly. It should be very easy to navigate and take almost no time to request a service.

### API handler

The API handler provides an interface to make specific API calls. Not only for our own API, but also for the other API's that will be used. We separate this from the client application to create a modular system.

## Server

The server is composed of three parts, the DB interface, the server application and the Rest API.

- **Rest API**

  The module will have a single interface to access all functionality. It will create an abstraction of the client in order to provide a more sustainable system. The interface is lightweight since it will only refer to the server application and therefore will not contain logic which would complicate the interface. The interface consists of a rest api which will handle the http request. However rest api will not work wheel for live updates and therefore we will use a websockets. The interface handels the message that a client has sent then stores it in the database as well as redirected to the connected client.

- **Server application**

  Provide the functionality of the backend of the system. Responds to requests from the Rest API and Interacts with the DB interface.

- **DB interface**

  The DB interface simplifies interactions between the database and the server. It provides a level of abstraction inside the server to simplify database requests.

## Design patterns

### MVC

Model-View-Control is a commonly used pattern to develop a user interface. The view is the visual representation which will be built in the app with react native. Then the controller will control the view and extract the data from the model with the Rest API. The model is the server side. It provides a clear distinction of the client application to make it more sustainable.

### DTO

As remote interface calls are expensive, we want to make the most use from them. This is why we will use data transfer objects. Data transfer objects allow us to send serialized objects with more data.

### Remote façade

To gain access to the fine grained objects inside data transfer objects, we will implement remote facades. These will translate rough grained data transfer objects back to the fine grained objects that it was created from. This will be used by clients receiving data transfer objects from the server.

# 4. Issues

### Payments

The service who requested agrees with the service provider how to pay. This allows flexibility for the different types of services. Services like purchasing items such as groceries can cause financial issues for the service provider. Therefore the service provider agrees with the requester of the estimated cost and can then adjust depending on the resulting cost. The service provider gives the recite to the service requester and then adjusts accordingly. For other services, such as package delivery or mowing the lawn, both parties agree when and how much to pay. If the service requester does not pay, the service provider contacts us and we can charge the service requester, since it will be a legally binding contract when agreeing on the price. In order to make it legally binding we need booth approvals and the agreed price. All will be verified via the app. There will be an integration in order to take a small commission to make a profit for the app providers.

# 5. Project plan

- V11: - Kickoff
  - Starting with project proposal
  - idèas of project
- v12: - Done with the project proposal
  - Done with the group contract.
- v13: - Started with learning/implementation frontend and backend.
  - Specify interfaces, as well as a more detailed system architecture.
- v14: - Rapport - utkast 1
  - Started with login
  - Implemete 2 modes with basic functionality for frontend
  - Communication between nodes and service requests.
  - A database with functions to modify it and basic server functionality.
  - Some test for backend
- v15: **Milestone #1.**
  - Done with Login
  - Started with chat view for frontend
  - Done with communicating with nodes and service requests.
  - A server api to make most of the required functionality.
  - Most tests for the backend.
- v16: - Rapport - utkast 2
  - Done with chat
  - Starting with implementing location and maps the frontend
  - Storing where the requests for the service were made in the backend.
- v17: - Implementing maps and location
  - Robust backend with 100% test coverage.
- v18: **Milestone #2**.
  - Rapport - utkast 3
  - Done with location and maps.
- v19: - Improving the user experience.
  - Finalize frontend and backend
- v20: **Rapport - utkast 4**
  - Making and practicing presentations.
- V21: **Final presentation**.
  - Writing the final version of the raport
- v22: **Report**
- v23: **FINAL DEADLINES**

# 6. Conclusion

To summarize our proposal, we propose to develop a service application targeting an older audience, especially in urban areas that could need help with activities outside their home. This application would serve as a medium through which services can be ordered. A volunteer would take requests from people needing it for an agreed payment. This proposal is interesting because it solves a real issue that exists in today's society.
We will implement an application for both android and IOS using react native. We will use a mongoDB database and our server will be written in Node.js.