TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Ilja Gužovski 152999IAPM

WEB-BASED LOCATION TRACKING PLATFORM ANALYSIS AND IMPLEMENTATION ON THE EXAMPLE OF CONSTRUCTION COMPANY

Master's thesis

Supervisor: Jekaterina Tšukrejeva
MSc
Co-supervisor: Jaak Henno
Phd

Tallinn 2020

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Ilja Gužovski 152999IAPM

VEEBIPÕHISE ASUKOHA JÄLGIMISE PLATFORMI ANALÜÜS JA ARENDUS EHITUSFIRMA NÄITEL

Magistritöö

Juhendaja: Jekaterina Tšukrejeva
MSc
Kaasjuhendaja: Jaak Henno
Phd

Tallinn 2020

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Ilja Gužovski

09.12.2019

Abstract

The goal of following Master's thesis is to perform an analysis and implementation of a location tracking platform and as a result develop an in-browser tracking application for a medium-sized construction company which is privacy concerned, safe, easy to update and does not rely on Google Play Store or Apple App Store – since all tracking functionality will be represented on an ordinary web page.

Various activity and location tracking solutions have become a significant part of different business infrastructures in recent years. Currently, there are many solutions on the market, quality and properties of each can vary greatly. One of the construction companies offered to integrate such infrastructure into their business processes. However, it appeared that there are no existing activity trackers that could completely comply with company internal privacy policies and given functional and non-functional requirements.

The thesis will propose a solution that attempts to create a platform different from existing tracking solutions because it runs in an end-user mobile phone browser and does not require installing an app.

In the first part of the work, the focus will be on gathering requirements from the host construction company. After that work proceeds with researching tracking systems, comparing, analyzing as well as selecting needed frameworks, libraries and architectures needed for implementation.

In the second part of the work, the main focus will be on documenting the implementation and testing process. Finally, all testing results will be evaluated, analyzed and a conclusion will be given.

As a result of this Master's thesis, the MVP of a mobile tracking platform will be developed, the MVP will be tested and comparison with existing tracking systems will be created.

This thesis is written in English and is 104 pages long, including 8 chapters, 41 figures and 17 tables.

Annotatsioon

Veebipõhise asukoha jälgimise platformi arendus ja analüüs

Käesoleva töö eesmärk on analüüsida ja luua askoha jälgimise platform, mis töötab veebipõhiste tehnoloogiate toel. Töö lõppeesmärk on luua brauserisisene GPS jälgimisrakendus, mis on privaatne, lihtsasti uuendatav, ohutu ja ei sõltu Google ega Apple App poest – kuna ise rakendus on tavaline veebilehekülg, mis töötab otse veebilehitsejas. Rakendus luuakse keskmist suurusjärku ehitusfirma jaoks.

Mitmesugused aktiivsuse ja asukoha jälgimise lahendused on viimastel aastatel muutunud erinevate äride tähtsaks osaks. Turul on mitmeid erinevaid jälgimisrakenduste lahendusi, mis on erinevate omaduste, hinna ja kvaliteediga. Antud ettevõte esitas soovi integreerida sellist süsteemi oma äri protsessi. Ilmnes, et pole ühtegi olemasolevat jälgimissüsteemi, mis vastaks täielikult ettevõttesisestele privaatsusnõuetele ning funktsionaalsetele ja mittefunktsionaalsetele nõuetele.

Töö käigus valmib lahendus, mis on erinev olemasolevatest jälgimislahendustest, kuna see töötab kasutaja mobiiltelefoni brauseris ja ei vaja rakenduse installimist.

Töö esimeses osas keskendutakse rakenduse nõuete kogumisele. Tööd jätkatakse jälgimissüsteemide uurimisega, raamistike, teekide ja arhitektuuride võrdlemise, analüüsimise ning valimisega.

Töö teises osas keskendutakse põhiliselt arendus- ja testimisprotsessi dokumenteerimisele. Lõpuks hinnatakse ja analüüsitakse kõiki testimistulemusi ja tehakse järeldus.

Töö tulemusena töötatakse välja mobiilne jälgimisplatvorm MVP, testitakse seda ja luuakse võrdlus olemasolevate jälgimissüsteemidega.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 104 leheküljel, 8 peatükki, 41 joonist, 17 tabelit.

List of abbreviations and terms

AMD	Asynchronous Module Definition
API	Application programming interface
CJS	CommonJS
DOM	Document Object Model
ES6	ECMAScript 6
Employee	An individual who works under a contract for client company
GPS	Global Positioning System
HP	(a company) Hewlett-Packard
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
JS	JavaScript
JSON	JavaScript Object Notation
Loader	(used in vehicle context) Type of tractor
MDN	Mozilla Developer Network
MIT licence	Permissive free software license originating at the Massachusetts Institute of Technology
MVP	Minimum Viable Product

R&D	Research and development
REST	Representational State Transfer
SDK	Software Development Kit
SPA	Single page application
Sensor	(used in context as) HTML5 Mobile device sensor
State	variable storage. In our case those are browser JavaScript variables.
TDD	Test-driven development
UMD	Universal Module Definition
W3C	World Wide Web Consortium
Webpage	A document which can be displayed in a web browser. In our context webpage supports JS and HTML5 Sensor API
Websockets	Communications protocol, providing full-duplex communication

Table of contents

Author's declaration of originality 3
Abstract 4
Annotatsioon 6
List of abbreviations and terms 7
Table of contents 9
List of figures 12
List of tables 14
1 Introduction 15
1.1 Background and motivation 15
1.2 Problem statement 15
1.3 Goals 17
2 Background and analysis of a problem 18
2.1 Problem background and explanation 18
2.2 Functional requirements 25
2.3 Non-functional requirements 29
3 Evaluation and analysis of existing solutions 31
3.1 Spyzie 32
3.2 SpyFone 34
3.3 Traccar 35
3.4 Google maps 37
3.5 mSpy 39
3.6 FollowMee 41
3.7 Comparison of existing solutions 43
4 Analysis of implementations and architectures 48
4.1 Analysis of different implementations for mobile app development 48
4.2 Analysis of needed sensors and their API-s 52
4.2.1 Environmental Sensor API-s: 52
4.2.2 Useful but not mandatory API-s that could be used: 53
4.2.3 Data storing API-s 54

4.3 Analysis of sensors compatibility for different browsers 54
4.4 Analysis of possible implementation problems and limitations 57
4.5 Analysis of web mobile development performance 58
4.6 Conclusion: web mobile development 58
4.7 Analysis of different architectures 59
4.8 Conclusion 61
5 Implementation 62
5.1 Databases 62
5.1.1 CouchDB and PouchDB 63
5.1.2 CouchDB 63
5.1.3 PouchDB 65
5.1.4 Postgres 66
5.2 Backend platform and framework 66
5.2.1 NodeJS 67
5.2.2 FeathersJS 67
5.3 Frontend framework 70
5.3.1 React 70
5.4 Component integration and communication 73
5.5 Use cases 74
5.5.1 User signup 74
5.5.2 User login 75
5.5.3 User location/sensor tracking (FR-1, FR-2, FR-3, FR-4, FR-6) 77
5.5.4 User location monitoring by admin (FR-7) 79
5.5.5 Set starting point and destination of user (FR-8) 80
5.5.6 Admin sees long term navigating history (FR-9) 82
5.5.7 Admin creates group of users (FR-10) 83
5.5.8 Admin sees group members on the map (FR-11) 86
5.5.9 Removing user data (FR-5) 88
6 Application testing 89
6.1 Functional requirements testing 89
6.2 Non-functional requirements testing 94
6.3 Testing results: conclusion 96
7 Conclusion 98
8 Summary 100

9 References 101
Appendix 1 – Platform source code 104

List of figures

Figure 1 Example of current problematic situation (1) 21
Figure 2 Example of current problematic situation (2) 22
Figure 3 How IT system could solve above mentioned business problem (1) 23
Figure 4 How IT system could solve above mentioned business problem (2) 24
Figure 5 Spyzie screenshot 32
Figure 6 SpyFone screenshot 34
Figure 7 Traccar screenshot 35
Figure 8 Google Maps screenshot 37
Figure 9 mSpy screenshot 39
Figure 10 FollowMee screenshot 41
Figure 11 Browser usage chart (Source: StatCounter Global Stats for October 2018) .. 55
Figure 12 New CouchDB database creation code snippet 63
Figure 13 New CouchDB user creation code snippet 64
Figure 14 Granting access of particular database to user - code snippet 64
Figure 15 CouchDB admin UI screenshot 65
Figure 16 Creating new PouchDB instance code snippet 66
Figure 17 Inserting into record into PouchDB code snippet 66
Figure 18 User service implementation code snippet 68
Figure 19 Interaction with backend through FeathersJS client library – code snippet ... 69
Figure 20 Tracking page source code 72

Figure 21 Deployment diagram 73
Figure 22 User signup sequence diagram 74
Figure 23 Signup page UI 75
Figure 24 User login sequence diagram 76
Figure 25 Login page UI 76
Figure 26 User location tracking sequence diagram 77
Figure 27 Monitoring page UI 78
Figure 28 Monitoring page UI – monitoring/sync successful (2) 78
Figure 29 User location monitoring sequence diagram 79
Figure 30 Monitoring page UI (Users marked as markers) 80
Figure 32 Send notification modal UI 81

Figure 31 Set starting point and destination of user sequence diagram 81
Figure 33 Admin sees long term navigating history sequence diagram 82
Figure 34 User history monitoring UI 83
Figure 35 Admin creates group of users 84
Figure 36 Create group page UI 85
Figure 37 Admin sees group members on the map 86
Figure 38 Show user group members page UI (group users are marked with markers) 87
Figure 39 Removing user data sequence diagram 88
Figure 40 Delete user data (Revoke rights button) 88
Figure 41 Code coverage report 96

List of tables

Table 1 Requirement FR-1 25
Table 2 Requirement FR-2 26
Table 3 Requirement FR-3 26
Table 4 Requirement FR-4 26
Table 5 Requirement FR-5 27
Table 6 Requirement FR-6 27
Table 7 Requirement FR-7 27
Table 8 Requirement FR-8 28
Table 9 Requirement FR-9 28
Table 10 Requirement FR-10 28
Table 11 Requirement FR-11 29
Table 12 Features comparison between Spyzie, Spyfone, traccar, Google Maps, mSpy, FollowMee 45
Table 13 Comparison between various mobile application development approaches 49
Table 14 Comparison between mobile web approach and business requirements 50
Table 15 Availability of sensors for different browsers 56
Table 16 Functional requirements testing results 93
Table 17 Non-functional requirements testing results 96

1 Introduction

The main goal of the thesis is to perform an analysis and implementation of a location tracking platform that will be powered by web-technologies. As a result, an MVP of an in-browser tracker for a construction company will be created, which could be used as a basis for further development. The developed tracker will have a set of unique properties, which are not present in other solutions: it does not depend on Google or Apple stores, it is easy to install and update, and it respects the privacy of the users.

To achieve this goal, the analysis of requirements, goals, as well as analysis of different solutions, architectures, implementations, tools, and libraries will be performed. Gathered knowledge should provide an optimal solution for fulfilling the goal.

1.1 Background and motivation

Tracking user location is an increasingly popular tool in modern logistics, construction, and service providing business. Companies such as Uber, Taxify, Lyft, Gett, Wings, and others use GPS and activity tracking in their apps to meet customer and service providers [1]. The dominant part of those solutions is based on the idea of separate apps for Android and iOS [2] [3]. However, the approach of separate iOS and Android apps has its drawbacks: the high cost of maintaining two nearly identical apps [4], issues with updating, versioning and installing of those apps, privacy concerns, and total dependence on Google Play Store and Apple store. Some parts of those problems could be addressed with the use of the safe in-browser environment, which is mostly independent, safe [4], easy to update, install and use.

1.2 Problem statement

Construction company needs a vehicle coordinating solution for its call center. The problem in the current situation is that most of the time there is no information about when and where the loaders, drivers and their trucks are. Each delay of the truck or

employee and failure in coordination/communication leads to time and money loss. To prevent communication failures, the company wants to analyze the existing problem, gather requirements and develop a simple application which should tell where the construction truck or employee is.

Current work will analyze and explore existing solutions and approaches for solving current problematic situation happening in the host construction company: Analysis of existing readymade solutions, constraints and approaches used in the software development for creating a GPS and activity tracker, as well as all the needed steps to implement this kind of software: analysis of functional and non-functional requirements, comparison of different implementations, documentation of implementation and implementation itself.

Web-technology based tracker has some great advantages, compared to building native Android and iOS applications:

At first, building separate native apps is considered to be less cost-effective than one unified app that suits all operating systems [4].

Secondly, it is far easier and cheaper to maintain everything on the same stack of technologies (mainly ordinary web technologies - HTML, JavaScript, SQL, Java, etc).

Thirdly, there are more web developers available on the market than Android or iOS developers. Those reasons were taken into account in the company, for whom I am implementing the solution. The company did not want to spend money on a separate Swift, Android or React Native developers, and decided to experiment with a web-based approach.

As a result, an MVP solution for tracking the user location and activity will be implemented. Solution will be secure, privacy concerned, independent from Apple and Google stores, scalable and highly configurable. Created MVP could be adapted and used as a basis for other projects and companies coming from other or similar business domains: such as peer-to-peer ridesharing, taxi cab hailing, food delivery, bicycle-sharing systems, logistics and many others.

1.3 Goals

Before starting the development the current business situation will be analyzed, exploration of the problem will be performed and all functional and non-functional requirements will be gathered.

After acquiring the requirements, the analysis of existing solutions on the market will be created. And a decision, if there is a need to create a dedicated application specifically for analyzed requirements, will be given.

Work proceeds with an analysis of architecture and different approaches on how such applications could be developed and analyze all needed dependencies and components for implementing the platform.

During the implementation phase, the Agile-based methodology elements will be used: Test Driven Development, timeboxing and early delivery of valuable software. However, comprehensive documentation of the development and testing process will be created as well.

As a result of a development process, an MVP solution for a construction company will be created. The MVP will solve company problems with the call center and will optimize loader and truck working flow. For testing the prototype, a suite of unit, functional and manual tests will be created and executed.

Finally, a conclusion how well application complies with all functional and non-functional requirements will be given and described with most important features that do the browser-specific tracking has, compared to ordinary native app-based solution.

2 Background and analysis of a problem

2.1 Problem background and explanation

The client for whom this solution is developed is a road construction company located in Tartu. The turnover of this company is over 10+ million euros. The company is engaged in the provision of services in the construction of roads. The company consists of departments each connected to counties and brigades and each brigade connected to the department and construction project. The company has many subcontractors. The company strategy is to expand in the southern part of Estonia and to optimize its costs.

Due to the growing and competitive nature of the company it became important to optimize logistic and transferring of construction assets, such as construction materials, vehicles, tools, and human resources. For managing and coordination was formed a "call center" - a small department, that is responsible for renting of construction vehicles, transferring them to place on time, transferring of building materials and solving problems between vehicle renting companies, subcontractor companies, and customer. Most of the call center job is done via phone, SMS, email, excel and open-source ERP system Odoo.

The problem in the current situation is that most of the time there is no information about when and where the loaders, drivers and their trucks are. Tracking and asking of the truck loaders is time-consuming and inefficient for the call center, but the location of each worker and construction vehicle is important.

Moreover, each delay of the vehicle or employee and failure in coordination/communication leads to time and money loss. To prevent communication failures, the company wants to develop a simple application that will tell where the construction truck or employee is.

At the moment, the customer company is willing to improve its IT infrastructure and reduce the number of things stored in excel and papers. At first it was proposed to use

some SaaS service as Vismo or Spyzie, however, it was decided that it is important not to store worker's location data in third party services. After that, it was offered to create simple Android and iOS apps, which will ask for a location.

However, the client did not want to pay for the maintenance of the nearly identical versions of the same app. Solutions which could produce applications for both Android and iOS at once from the same code base (such as Cordova and React native) were discussed as well, however, were questioned because of updating and installing issues - it was clearly stated that application needs to be easy to use, install and update on any phone. This requirement was brought in by previous experience - most of the employees had experienced problems with installing/updating or even opening of apps (such as Google maps, for instance) on their smartphones.

In general, the company needed a solution that should be easy to use, support and train. After a long discussion with the technical director of the company, it was agreed that the product could be a simple HTML5 page that will poll position and other needed sensor data from the employee phones and send it to the server for further analysis. To ensure the correct working of application it will be installed on mobile devices given by the company and usage of this application will be not mandatory for employees. The application will not track user activity outside of the range of working hours and could be closed employee just by closing the page.

Looking from the technical side ordinary HTML page satisfies all given requirements: It is very easy to update (just by refreshing the page), it can query all needed sensors for our purposes, I do not need to develop separate applications for iOS and Android, end-users do not have to download anything from the Apple or Google Play Store and majority of people know how to use the smartphone for accessing the web page, so the ease of use, learning curve and user interaction with an app is kept to minimal. In addition to that, other sensors could be added and a larger scale of developers could improve and maintain the application: at present, there are far more web developers, who know JavaScript than Swift or Java developers with good knowledge of Android. [5] On the backend sensor data will be processed and analyzed to give the current location and activity of the employee.

The problem with the interaction of call center and client/subcontractor/host company official is the following: Usually truck driver knows where to go and where to transmit the construction material but it happens quite often that there is nobody on the construction site to accept the materials: either it is some official from subcontractor firm or another brigade from other company or official of client. So instead of efficient use of construction vehicle, the truck driver waits for an unknown amount of time. It is a very common situation that authorized person is missing or forgot about the upcoming construction vehicle or assets. Without his signature vehicle cannot be unloaded or used. Daily this puts stress on a business: rented construction vehicles are waiting, construction materials are not transmitted, working hours of a construction vehicle are used inadequately, call center cannot forecast how much assets could be transmitted, which results in an inadequate billing of the rented vehicles and puts high stress on a call center: usually at peak hours people are calling when they have problems with paperwork and “nothing is moving”.

Below I have created diagrams which illustrate one particular problematic situation in the call center that happens almost daily:

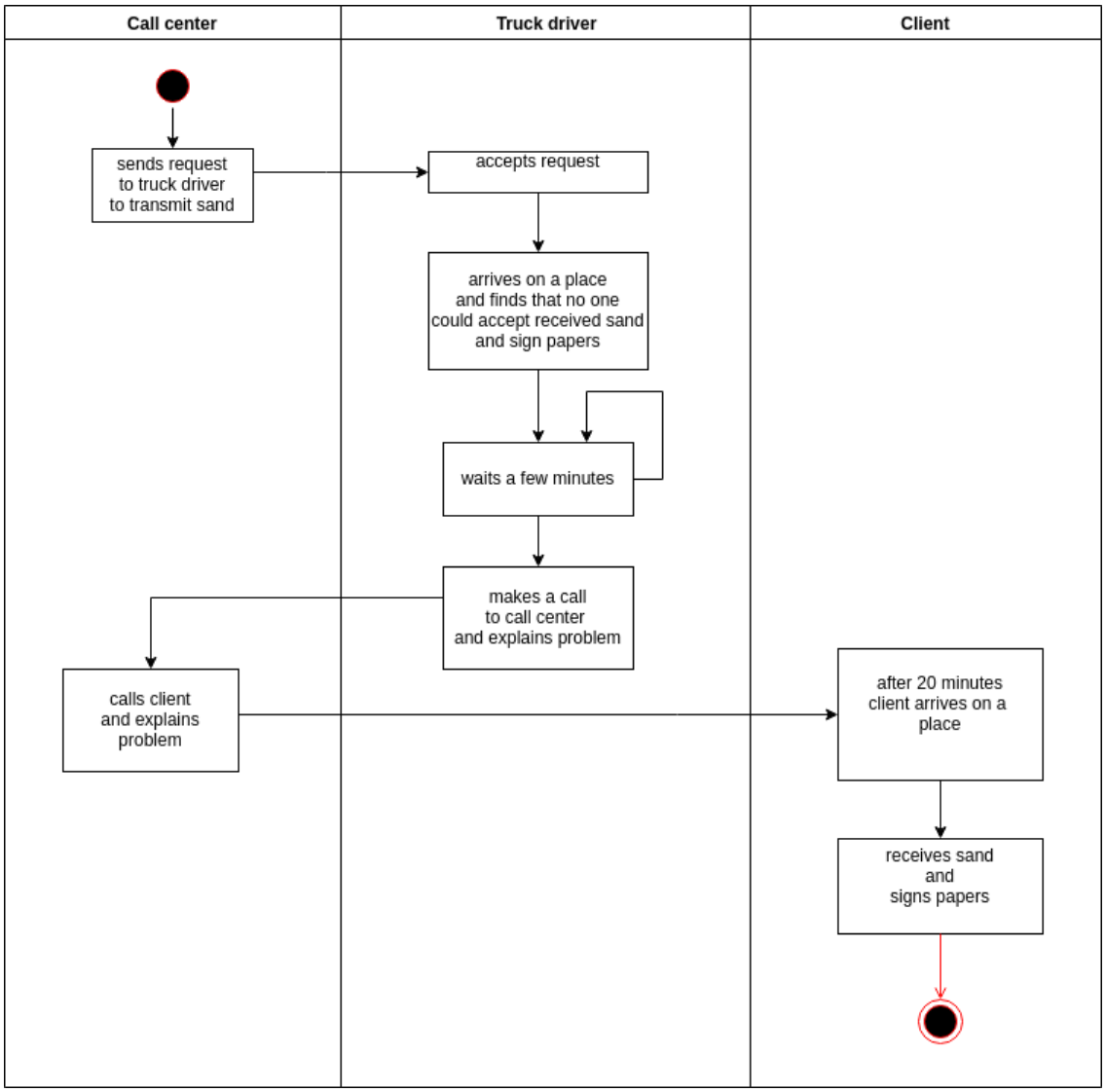

Figure 1 Example of current problematic situation (1)

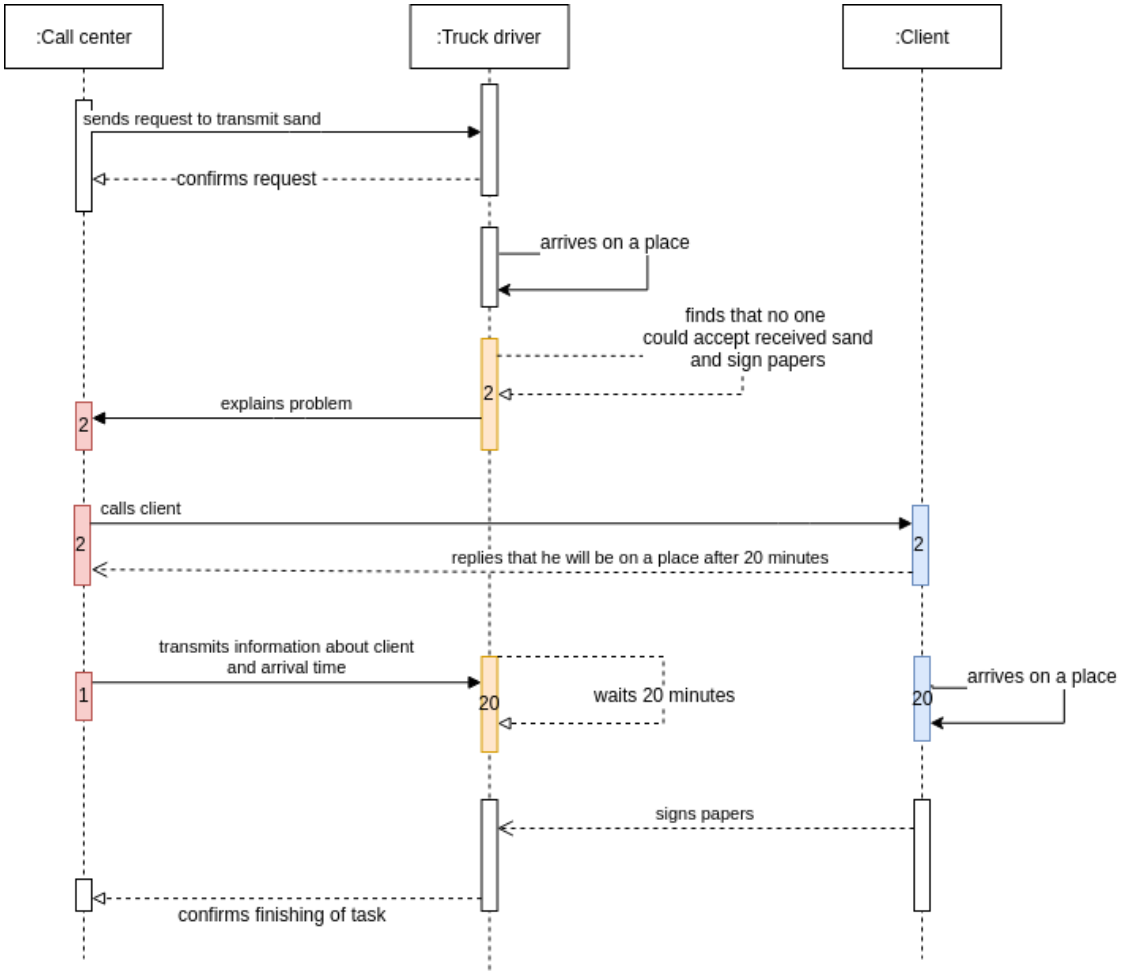

Figure 2 Example of current problematic situation (2)

The graph above illustrates failure in communication – the client forgets about the upcoming truck, so instead of being accurate on time in the correct place, he is somewhere else. This results in the idle standing of the truck and truck driver. The truck is rented on an hourly basis and the truck driver receives a salary. Obviously this situation results in a money loss, which is not good for the business. This routine situation (which happens many times during the day) results in 5 minutes of work done by call center (steps which took additional time are marked by red color), 22 minutes of work done by truck driver (marked with orange), and 22 minutes of work done by client (marked with blue).

As a result, the host company and client are losing time and money. All colored steps could be eliminated and as a result, we will see a much simpler and less time-consuming business process. Note that truck vehicle is also standing, which result in additional renting and amortization costs.

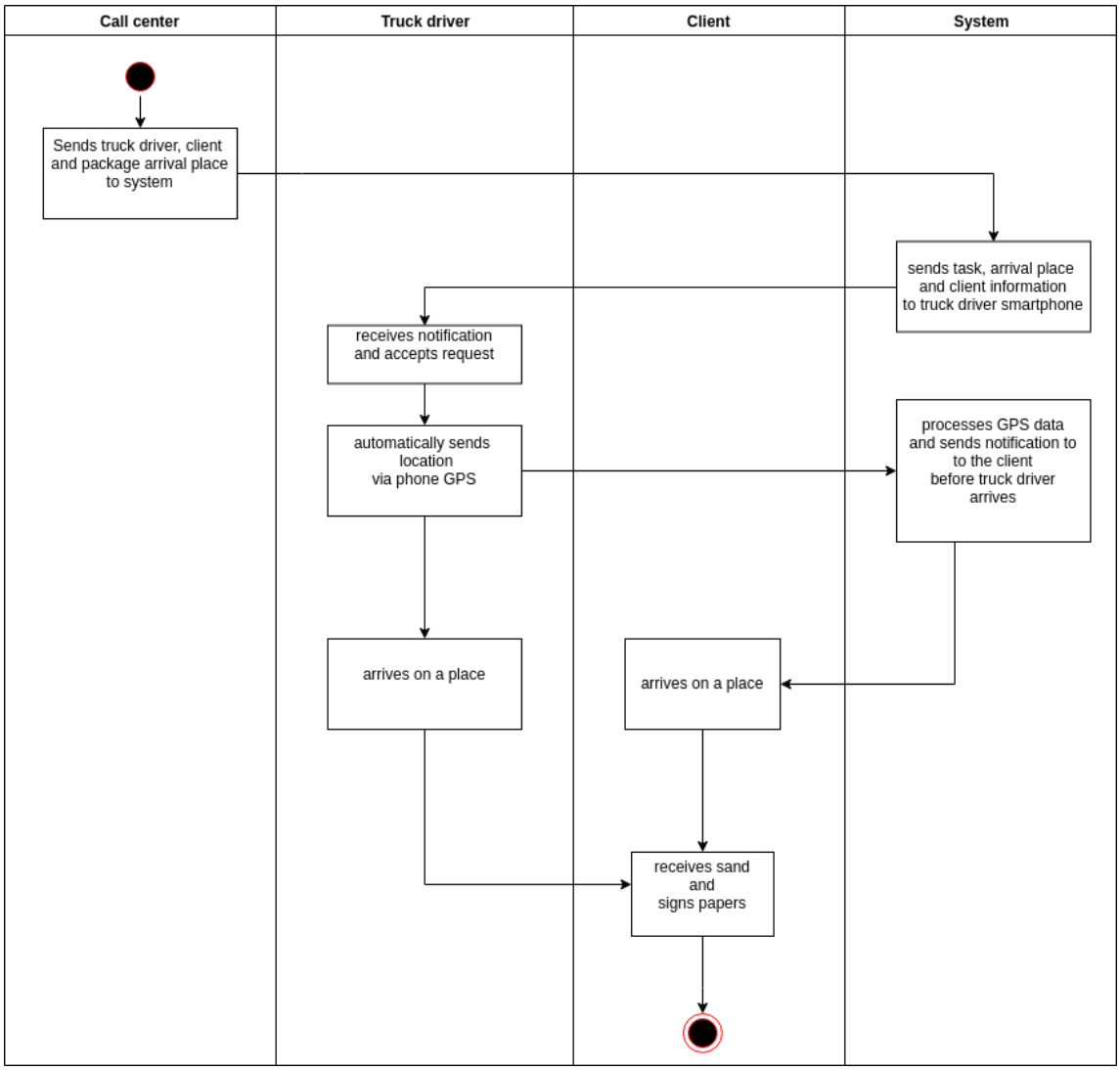

Figure 3 How IT system could solve above mentioned business problem (1)

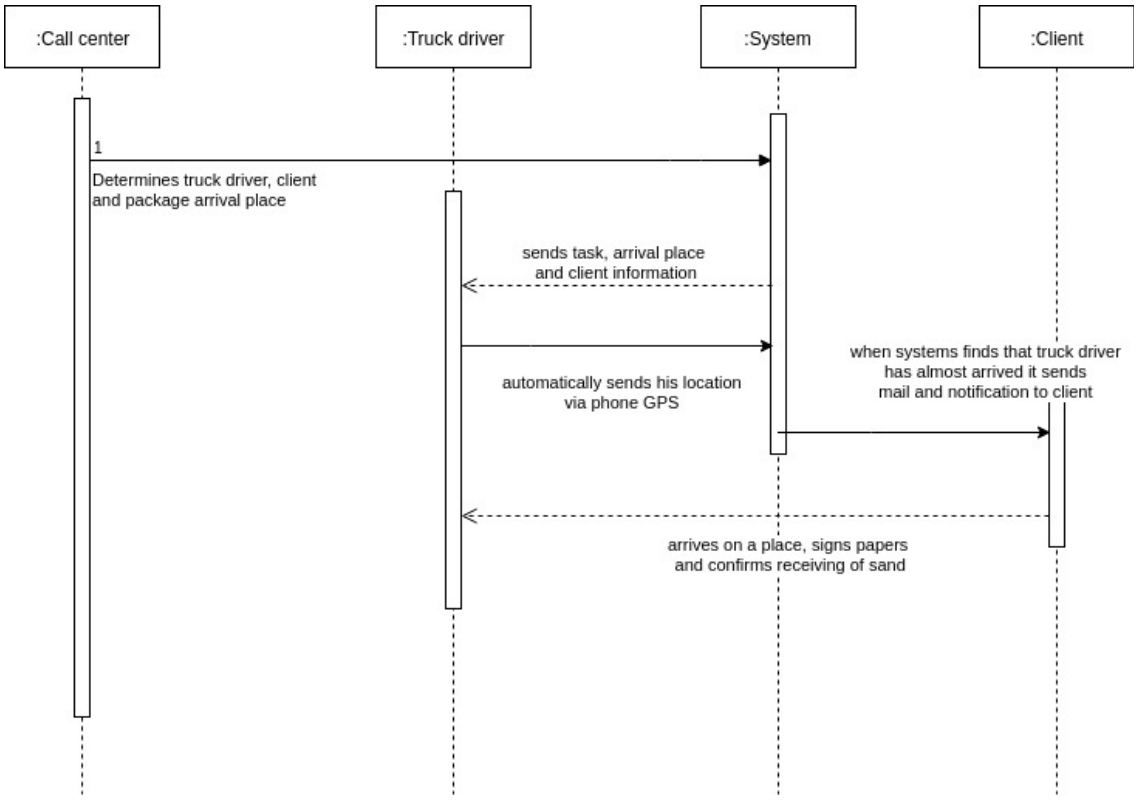

Figure 4 How IT system could solve above mentioned business problem (2)

The proposed solution takes care of work done by the call center. Ideally, it is possible to remove all problematic steps from this business process. If you compare Figure 1 with Figure 3 and Figure 2 with Figure 4 you will notice that “waiting” steps were eliminated by introducing a broker called “System”. Basically system sends the email to the client before truck arrives. The time when the email is sent is based on the location of the truck and average moving speed of the truck; let’s say for the past hour. This results in less work for the call center, no waiting of the truck driver and happier client, who does not need to hurry.

Reduction of communication failures leads to a significant increase in work productivity, releasing the call center from stressful peak hours which usually happen between 11 and 15 o'clock, and reducing idle standing of the vehicles and employees. Client satisfaction with company services is expected to increase because he is notified on time and would not lose time because of his or her company organization failures.

To solve such problems it was agreed to create a system, that should gather the position of the user and forecast his activity – is user driving or standing, or waiting in the traffic jam. The application will be self-hosted on some private VPN server, so and will

comply with European Union GDPR law. The application should be cross-platform – this means that it must work on the most popular mobile operating systems – Android and iOS. The application should be easily closed and should respect user privacy. For the testing reasons, it should support admin and user roles. The end goal is to locate the position of the user by showing a marker on a map.

There were other requirements as well, but to sum up, there were following major functional and non-functional requirements to fulfill:

2.2 Functional requirements

ID	FR-1
Requirement	Application should gather as much available sensors data as possible. Geolocation , devicemotion, deviceorientation sensor data is mandatory.
Users	System end user
Precondition	Authorized end user
Result	User sensors data should be queried and saved to database
Use case	User logins to page where he accepts permissions about using his geolocation and other sensors. After that system should send user geolocation , devicemotion, deviceorientation and possible other sensor data to server and save it to database. Gathering and sending of the data should be done with 1 second interval.

Table 1 Requirement FR-1

ID	FR-2
Requirement	Application should continue sending sensor data in case it is in minimized or in idle state.
Users	System end user
Precondition	Authorized end user who accepted geo-tracking permissions (as in functional requirement ID 1)

Result	User sensors data should be queried and saved to database
Use case	User locks his phone or minimizes or minimizes browser with application tab.

Table 2 Requirement FR-2

ID	FR-3
Requirement	Each sensor write request should include a timestamp
Users	System end user
Precondition	Authorized end user who accepted geo-tracking permissions (as in functional requirement ID 1)
Result	User sensors data should be queried and saved to database with a timestamp field
Use case	When application generates sensor data it also appends creation date, after that the whole request is sent to backend and saved to database.

Table 3 Requirement FR-3

ID	FR-4
Requirement	Application could be reopened if it had crashed or was closed.
Users	System end user
Precondition	Authorized end user who accepted geo-tracking permissions (as in functional requirement ID 1)
Result	Application could be reopened and restored to the same state as before being closed
Use case	User accidentally closes the application browser tab, opens the browser, types application page URL, browser should open application in the same state as before close.

Table 4 Requirement FR-4

ID	FR-5
Requirement	Application could remove all user data if prompted
Users	System administrator
Precondition	Authorized administrator
Result	Application should not contain any user sensor data
Use case	End user prompts admin to remove all user data, administrator removes all data from database.

Table 5 Requirement FR-5

ID	FR-6
Requirement	User could update application without prompts with one click
Users	System end user
Precondition	Authorized end user
Result	Application got updated
Use case	End user refreshes the application page, if application has got any updates then it gets updated automatically by the browser

Table 6 Requirement FR-6

ID	FR-7
Requirement	Admin could see user position on the map
Users	System administrator
Precondition	Authorized admin
Result	Admin knows position of user
Use case	Admin logs in and sees a map with a position of user.

Table 7 Requirement FR-7

ID	FR-8
Requirement	Admin could set starting point and destination for user
Users	System administrator
Precondition	Authorized admin
Result	Both admin and user could see starting point and destination point of the route.
Use case	Admin sets starting point and destination for a truck driver. Truck driver could see destination on web page. Admin could see route of a truck driver on a map.

Table 8 Requirement FR-8

ID	FR-9
Requirement	Admin could see long term routing history of user
Users	System administrator
Precondition	Authorized admin
Result	Admin could see a route of the user in chosen date time interval (1 month ago and more)
Use case	Admin chooses driver and date time interval and sees a route of the user on a map.

Table 9 Requirement FR-9

ID	FR-10
Requirement	Admin could group users into groups
Users	System administrator
Precondition	Authorized admin
Result	Users could grouped
Use case	Admin creates a group and registers each user to group.

Table 10 Requirement FR-10

ID	FR-11
Requirement	Admin could see group members on the map
Users	System administrator
Precondition	Authorized admin, users are added to group.
Result	Admin could see a routes and locations of users in a group on the map
Use case	Admin chooses a group. Markers of members appear on the map..

Table 11 Requirement FR-11

2.3 Non-functional requirements

1. Technical requirements:

    1.1. Application should work on phones with OSes: iOS 9+ and Android 4.4+

    1.2. Application should work on the most popular cross-platform browsers such as Firefox or Chrome.

    1.3. Admin could access the admin dashboard from a browser.

    1.4. Admin dashboard should be simple to use: no actions should take more than 4 clicks

2. Privacy requirements:

    2.1. Application could be turned off for user privacy.

    2.2. User-specific data could be deleted.

    2.3. Data of the users should be stored by the company and not by any 3rd party.

3. Maintenance and business cost requirements:

    3.1. Application is itself is MVP (Minimum viable product), with minimum functionality to fulfill the business goal.

3.2. Application should be developed as quickly as possible with the minimum usage of human resources.

3.3. There should be a large pool of software developers who qualify to enhance the given MVP.

4. Application performance requirements:

4.1. Each interaction with the application user interface should not take longer than 5 seconds

3 **Evaluation and analysis of existing solutions**

Before I begin with the implementation part, I have to compare how do existing solutions that comply with functional and non-functional requirements. I evaluate a 6 most popular/complete/similar solutions found on the internet:

- **Spyzie**
- **SpyFone**
- **Traccar**
- **Google Maps**
- **mSpy**
- **FollowMee**

Each of these solutions provides its own advantages and disadvantages. Those applications could be divided into two main groups:

- **Parental tracking apps** – despite being a parental spying app, some companies claim that they could be used for business purposes as well. Usually, those applications have modern UI, larger client base and more possibilities to track user activity.
- **Employee tracking apps** – those apps are intended for businesses. Some of them appear to suit our business goals quite well. But as you can see later there are some bigger problems, the biggest concern is an inability to control and own the data: most of the solutions store the data on their servers and do not provide any API or customization layer. In addition to that, the user interface of some applications looks quite outdated.

Further I will evaluate each solution in detail:

3.1 Spyzie

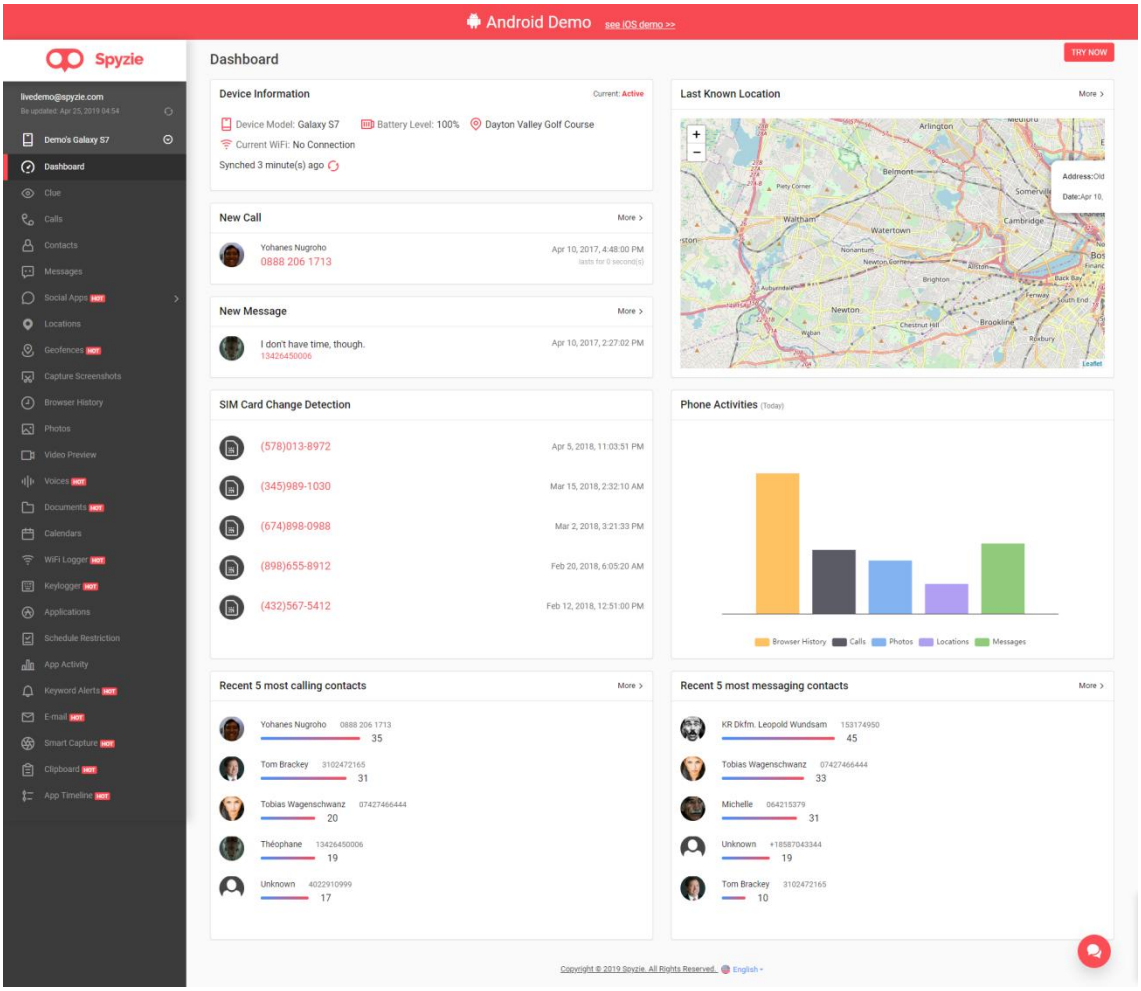

Figure 5 Spyzie screenshot

Spyzie is a monitoring tool to track cell phone activities such as GPS, SMS, calls, photos, browsing history, videos, WhatsApp, etc. [6] Spyzie is one of the most complete phone tracking solution found, functionally covers almost all needs, except it does not have a self-hosted instance, data is stored on the 3rd party server. Runs via separate iOS or Android application.

The main problem is that it does too much “spying” to be suitable for enterprise use: keyboard tracking, messenger, clipboard and screenshot capturing make it look more like a parental app / spying solution / spyware. It is impossible to turn off or turn on unneeded features or customize this behavior. Because of that, I may conclude that Spyzie would not respect the privacy of employees.

It is worth mentioning, that Spyzie is intended to track a limited number of mobile phones – there is no dashboard to track multiple devices, for instance, you cannot see all devices on the same map at once. In my opinion, Spyzie belongs to the **parental tracking apps** group. This kind of software will be hard to integrate into the existing infrastructure: it will track privacy-sensitive information, which should not be tracked, in addition to that, it is impossible to turn the app off during work off-hours. However, Spyzie has the following advantages, which competitors do not have: Easy to use modern-looking user interface, good UX, large userbase, and support.

To sum up, Spyzie has the following advantages and disadvantages compared to other competitors:

1. Advantages:
    a. Modern looking user interface
    b. Probably larger community (ranked first in Google)
    c. Large possibilities to track phone information: call logs, social networks, GPS, SMS etc.
2. Disadvantages:
    a. No possibility to control what things to track and what do not – either you track everything (GPS, Call logs, Browser history) or do not use the app.
    b. No external API
    c. No possibility to self host the instance of Spyzie, everything is stored on Spyzie servers.
    d. Lack of possibility to see devices on the same map at once.
    e. Impossible to turn off during work off-hours
    f. Price is ~100$ year per device.

3.2 SpyFone

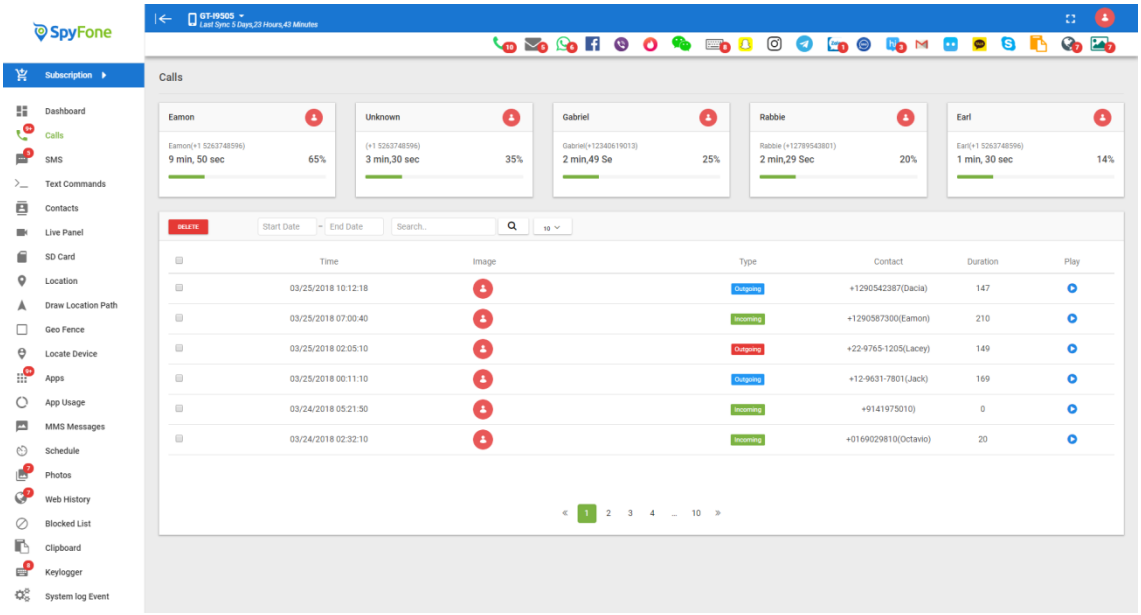

Figure 6 SpyFone screenshot

SpyFone is one of the most popular solutions for monitoring internet activity on mobile phones according to the SpyFone website [7]. Claimed to be used by large newspaper companies: The Times, The Sun and The Guardian. Suffers the same problems as Spyzie: no possibility to store the data on your server, too much control over the phone: reading of email, Facebook messenger, etc. Purely for a tracking app used by businesses, this app does track too much user sensitive information (exactly like Spyzie does).

However, they still advertise their solution to businesses: SpyFone web site states that SpyFone is a parental and employee monitoring tool. In my opinion, the SpyFone app is more parental oriented than business-oriented. It is a question of how ethical it is to monitor every call, text message and email of an employee – in my opinion, you should not track everything on employee phone. It is impossible to customize and tune this behavior – even the base package does track too much user sensitive information.

Currently, I cannot imagine how and why Spyfone is used by such large companies – it does not provide anything unique compared to Spyzie. In my opinion, SpyFone belongs to **parental tracking apps group** more than to business tracking apps group.

SpyFone has following advantages and disadvantages compared to other competitors:

1. Advantages:
    a. Modern looking user interface
    b. Probably large community of users and clients
    c. Used by large businesses (The Times, The Sun and The Guardian)
    d. Large possibilities to track phone information: call logs, social networks, GPS, SMS etc.
    e. Claimed to be solution 1 worldwide.
2. Disadvantages:
    a. Does not offer anything very special compared to Spyzie
    b. Higher price than Spyzie – 200$ to 300$ per year per device
    c. No external API
    d. No possibility to self host the instance of SpyFone, everything is stored on SpyFone servers.
    e. Lack of possibility to see devices on the same map at once.
    f. Impossible to turn off during work off-hours

3.3 Traccar

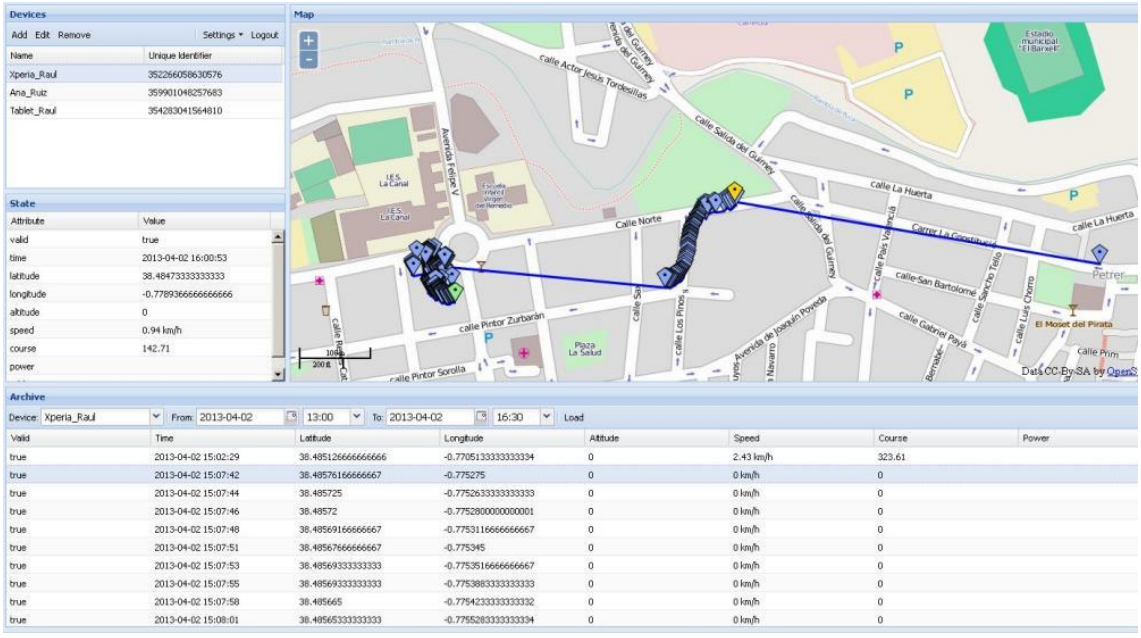

Figure 7 Traccar screenshot

Traccar is a GPS tracking software created for vehicle and personal tracking that could either be cloud-based or self-hosted. Traccar offers real-time view, reports, and notifications. [8]

Currently, Traccar is one of the most serious and complete solutions with the least amount of cons and the most amount of pros: it is open-source, provides both server and client, so the data could be stored on your personal server. It does not position itself as a spy app like the others and does not look like a parental app that has grown into a platform for doing everything related to tracking. It also seems that it is uses GPS only. Basically, because traccar is an open-source sofware, it can be extended in several ways. However, there are a lot of issues reported related to the functioning of traccar, that could be found on their Github page [9]. Traccar is the closest suitable solution for our purposes.

However, there is some functionality missing in the traccar, that is essential for fulfilling functional requirements given by the customer: you cannot store routes for a long time, it is impossible to use traccar directly from web page: you have to download an Android or an iOS app. I have downloaded a traccar Android app and, indeed, it respected user privacy – it never hid tracking notification from phone status bar. However, I have found that it will be quite hard to extend the application, because currently it is quite hard to use and setup, mainly due to fact that there are too many options in admin UI, which cannot be turned off or on, and it is also impossible to turn tracking off for particular hours. Despite that, among the competitors traccar is the **most suitable employee tracking** app for our goals.

Summing up:

1. Advantages:
    a. Open source
    b. Free to use
    c. Most complete employee tracking solution
    d. Business oriented
    e. Privacy concerned
    f. Supports not only mobile phones but special GPS tracking devices
    g. Large community

h. Possibility to self host the instance of an app, web interface and database

2. Disadvantages:
    a. Outdated UI, which needs refreshing
    b. Hard to setup and use admin interface
    c. Hard to extend
    d. No possibility to track user location directly via web interface
    e. High amount of open issues
    f. Impossible to see history of routes
    g. Impossible to set starting point and destination
    h. Impossible to turn off during work off-hours

3.4 Google maps

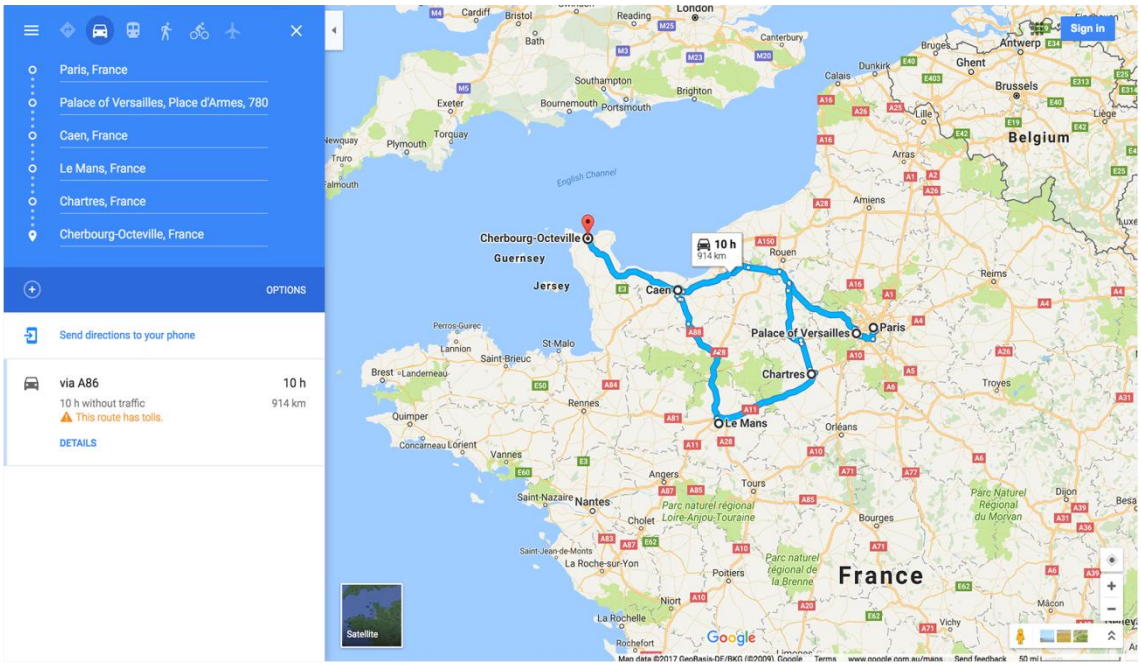

Figure 8 Google Maps screenshot

Google Maps is a mapping service developed by Google. Google maps offers street maps, 360 degree views of streets, as well as real-time traffic conditions, route planning for travelling by foot, car, bicycle or public transport [10].

Google maps could be used as a GPS tracking app if the user agrees to share his location. There are some inconveniences of course: for instance, you cannot group your

employees conveniently; there is no API to track user location. Still, Google maps offer the most sophisticated permissions control for the user, you can either use Google maps app or use Google maps via web page, high accuracy of tracking, probably great security. You still send a lot of third party data to Google, but at least it is well secured. The biggest problem is that Google maps are not intended for such use and you still do not own the data of the employees.

Moreover, Google maps do not have a sufficient amount of needed features – you cannot turn app tracking off during work off-hours, there is no dashboard for tracking multiple users. Despite that, maps and navigation are excellent, and you can use map and navigation at once – none of the competitors offer map navigation and routing at once. Purely for enterprise GPS tracking, Google Maps offer more functionality and control than Spyzie and SpyFone but less than Traccar and FollowMee.

Google maps advantages and disadvantages:

1. Advantages:
    a. Modern looking user interface
    b. High accuracy of GPS tracking
    c. High security standards
    d. Used by millions of people
    e. Backed by big corporation
    f. Best control of permissions among the competitors
2. Disadvantages:
    a. Not intended to be used as a tracking tool: it is impossible to group or watch multiple employee location at once.
    b. Data is stored on Google servers.
    c. No external API for getting a location of user. You can see routes only in the Google maps app or Google maps webpage.
    d. Lack of possibility to see devices on the same map at once.

3.5 mSpy

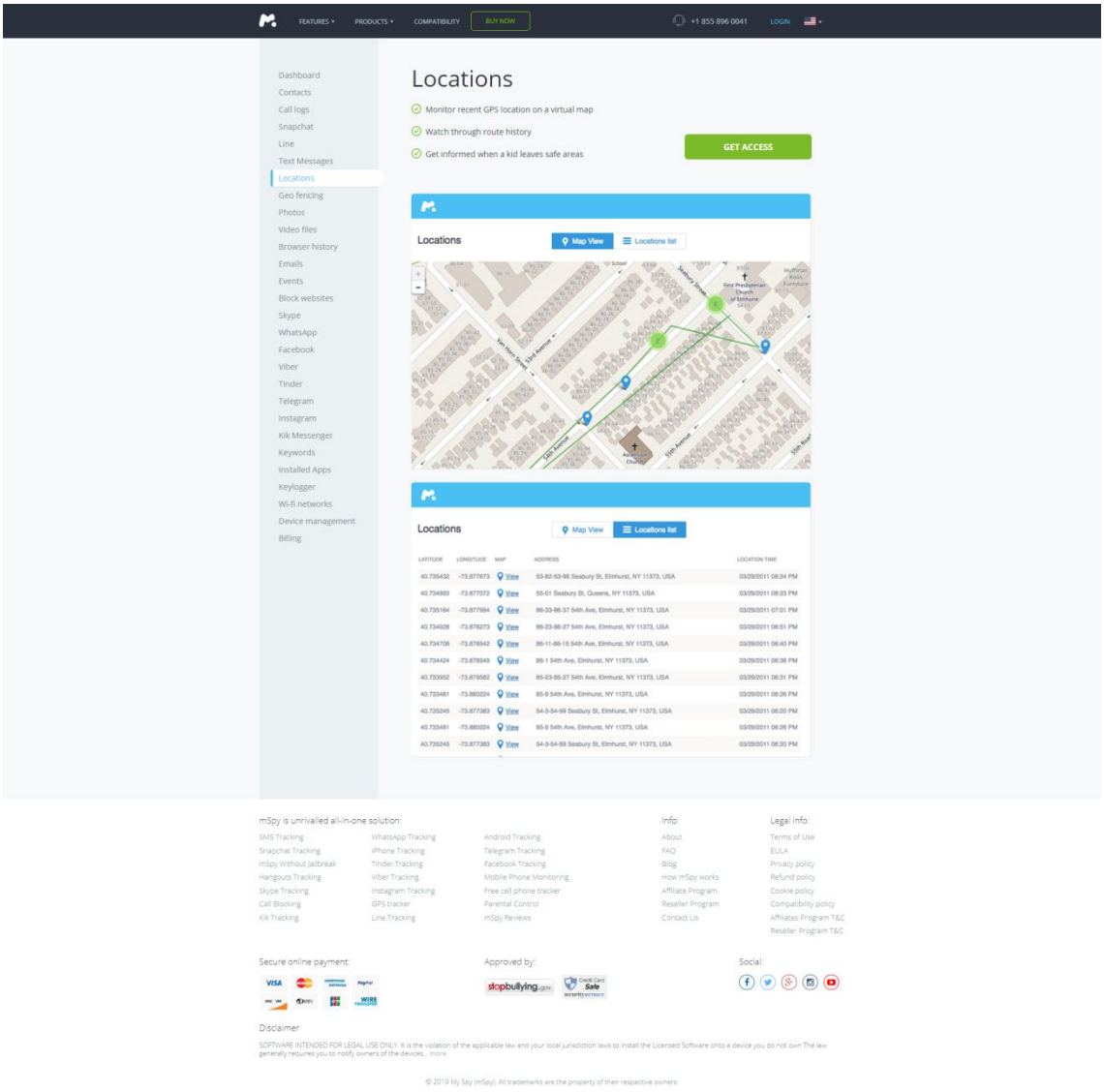

Figure 9 mSpy screenshot

mSpy is a parental control app for smartphones that allows parents to monitor text messages, calls, current GPS location, messengers and social network. [11]

mSpy looks very similar to SpyFone and Spyzie: a parental tracking app with the same options as SpyFone and Spyzie – tracking of location, messengers, mails, calls. Claims to be an "*a leading parental control app for smartphones that allows parents to monitor text messages, calls, current GPS location, Snapchat, WhatsApp and much more"* [11]. However, no statistics or popularity source is provided.

Application suffers from the same drawbacks as Spyzie and SpyFone: it is not intended for enterprise GPS tracking, but for parental control. Among the competitors, it provides most possibilities to track social networking and messaging. For instance, mSpy can keylog following applications: WhatsApp, Viber, Instagram, Facebook, Tinder, Telegram, Skype etc. The main disadvantage, in my opinion, is outdated UI and higher price than Spyzie, mSpy for which mSpy does not provide anything specific compared to competitors.

To sum up, mSpy is not that much different from Spyzie or SpyFone, it is a **parental app** with the following advantages and disadvantages:

1. Advantages:
    a. Large possibilities to track phone information: call logs, social networks, GPS, SMS etc with focus on messaging and social networking.
2. Disadvantages:
    a. Does not offer anything unique compared to Spyzie or SpyFone
    b. No possibility to control what things to track and what do not – either you track everything (GPS, Call logs, Browser history) or do not use the app.
    c. No external API
    d. No possibility to selfhost the instance.
    e. Lack of possibility to see devices on the same map at once.
    f. A little bit outdate UI
    g. Price is ~160$ year per device.

3.6 FollowMee

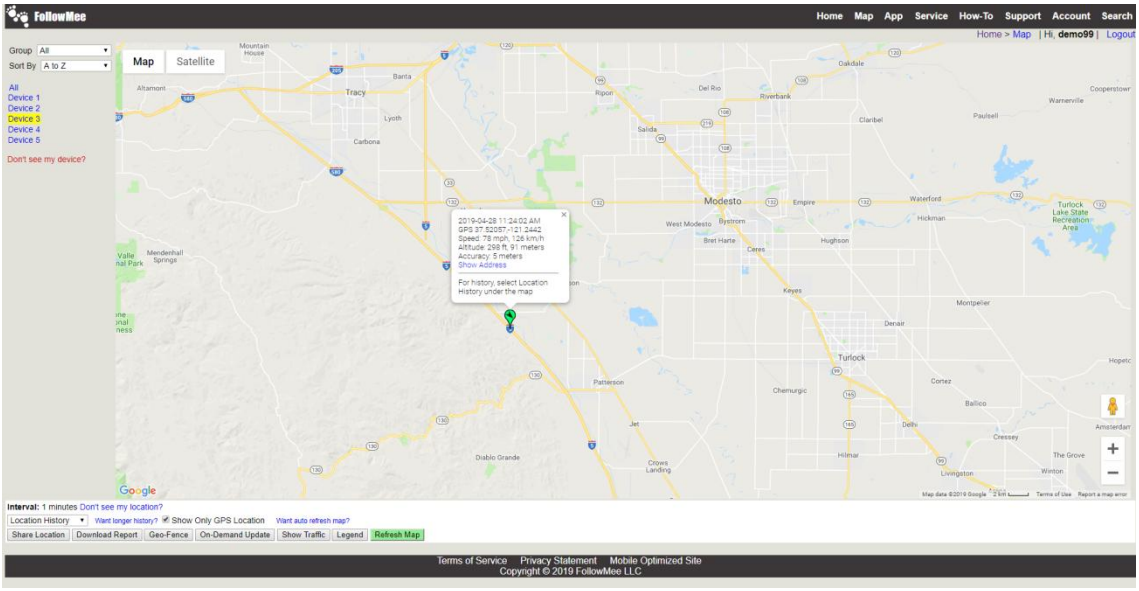

Figure 10 FollowMee screenshot

FollowMee is a GPS tracking solution that does not need an app for working, supports tracking directly from HTML page, has web-based and has a set of convenient features: grouping of tracked users, on-demand updates, legends. In addition to web-page you can directly install an app to following operating systems: Android, iOS, Windows. [12]

Some features are quite unique and are missing in Traccar and other competitors – for instance, archivation of tracking history for weeks, months or years. The main disadvantage of the app is that there is no possibility to self-host the server: all data should be sent directly to FollowMee server, you cannot install or self-host the server. Another minor problem is outdated UI and lack of external API – FollowMee cannot be extended easily. FollowMee clearly belongs to the **employee tracking apps** category.

FollowMee has following cons and pros:

1. Advantages:
    a. Possibility to track GPS without app, only with browser.
    b. You can see every tracked device on map at once.
    c. You can download GPS reports
    d. UI is simple to use
    e. It is possible to store GPS reports for 3 years for an additional price.
2. Disadvantages:

a. No possibility to selfhost the instance.
b. Outdated UI
c. ~50$ per device per year for all features
d. No external API

3.7 Comparison of existing solutions

To compare different solutions and ensure that they are suitable for the problem I will compare solutions on different factors – privacy concerns, possibility to extend, completeness of functional and non-functional features.

To evaluate are those solutions suitable or not for solving the functional and non-functional requirements, I have created a table with a chosen list of most vital functional and non-functional requirements:

Non-functional:

1. **Has self-hosted instance** – it means that only the host company can control, store and see the data of their employees. Basically it means that the data could not be stored in the cloud and only on a private server **(Non-functional requirement 2.3)**
2. **Respects user privacy** – can track GPS and available sensors only – some solutions found on the internet do too much, track phone calls, messages, etc. Client needs purely a location tracker, not spyware or parental solutions **(Non-functional requirement 2.1)** That kind of software will not be accepted by the workers. In other words, it means that tracker should be not a parental app tracker, but an employee tracking app.
3. **It can be used via mobile phone** – does not require any special device – It was agreed that phones are used and not any other devices, which are usually not that cheap. **(Non-functional requirement 2.1).** Tracking information could be sent either via iOS or Android app. Or via web app – which is much more preferable, privacy concerned and secure and also complies with **Non-functional requirement 1.3**.
4. **Application has web-based admin dashboard** (Non-functional requirement 1.3) – it is totally impossible to use an application without this feature, call center needs to track down where the employees are, this is an essential feature, without which the whole program does not make any sense.
5. **Could be extended / has an API –** Application has an API, so it can be extended easily. Open-source applications also belong to this group, because they can be easily extended as well. API is needed for various reasons: one is

that it could be used for implementing the delivery of email when the employee is approaching the client.

Functional:

6. **Admin could see user position on the map** (**FR-7**) – this is the most basic and important feature for call center, complies with **Non-functional requirement 1.3**
7. **Admin could set a starting point and destination for user** (**FR-8**) – This feature is needed for measuring the distance to the client by the logistic center. If the employee is approaching the client then the call center can call the client or issue and email. If the program could be extended I could automate this behavior.
8. **Admin could see long term routing history of user** (**FR-9**) – It is important to store the data for more than one month, this data is required for further analysis by the call center.
9. **Admin could group users into groups** (**FR-10**) – Groups are needed for various reasons, like splitting employees into projects, departments, etc. For instance, if a few trucks belong only to one big project it is reasonable to group them to only one project to see the location of the group on the map at once.
10. **Admin could see group members on the map** (**FR-11**)

Product name / Requirements	Spyzie	Spyfone	traccar	Google Maps	mSpy	FollowMee
Has self hosted instance	No	No	Yes	No	No	No
Respects user privacy – can track GPS and available sensors only	No	No	Yes	Yes	No	Yes
Can be used via mobile phone – does not require any	Yes	Yes	Yes	Yes	Yes	Yes

special device						
Has web based admin panel	Yes	Yes	Yes	Yes	Yes	Yes
Could be extended / has an API	No	No	Yes	No	No	No
Admin could see user position on the map (FR-7)	Yes	Yes	Yes	Yes	Yes	Yes
Admin could set starting point and destination of user (FR-8)	No	No	No	No	No	Yes
Admin could see long term routing history of user – 1 month and more (FR-9)	No	No	No	Yes	No	Yes
Admin could group users into groups (FR-10)	No	No	No	No	No	Yes
Admin could see group members on the map (FR-11)	No	No	No	No	No	Yes
Final decision (All requirements must apply)	No	No	No	No	No	No

Table 12 Features comparison between Spyzie, Spyfone, traccar, Google Maps, mSpy, FollowMee

To make a final conclusion we must take into consideration how well analyzed products comply with the list of requirements I have provided. According to the analysis, it appeared that none of the existing solutions completely complies with non-functional and functional requirements.

The biggest problem is that the most of the solutions do not provide any capabilities to store the data on your own server – only open-source solution traccar complies with this requirement.

The second biggest problem is that most of the solutions are “Parental apps” that do not respect user privacy – from all solutions, only FollowMee and traccar were intended for businesses use.

The third biggest problem is lack of an API and inability to extend and modify source code of provided solutions: from all solutions only traccar could be extended, traccar has REST API and you could probably roll your own endpoints if you wish (note: traccar is open-source solution).

Solutions like SpyFone, Spyzie, and mSpy are parental apps that track everything – GPS, social networking, emails. Clearly the idea of spying personal information of employees is not the goal of the business. In addition to that, data is stored in the cloud – you do not own or control the data. But for business is important not to share the data with 3rd party services.

From all solutions mentioned above traccar appeared to be the most suitable solution, but without member grouping support, quite high amount of issues found on Github, possibility to set starting point and destination, long term tracking and routing history – traccar just cannot be considered to be an appropriate solution for our requirements. From the contenders, FollowMee addresses a big part of lacking functionality of traccar, but cannot be self-hosted, because every piece of data is sent to FollowMee servers.

The analysis revealed that all solutions have their own cons and pros and most of the solutions cannot be changed or modified, from the all above only traccar could be extended, but this requires a great amount of work and the codebase of traccar is rather large and complicated. Codebase of traccar is possible to extend, but it would not be

easy – it is better to create your own MVP that does not address any problems or flaws of the analyzed solutions.

As a result, because none of the above complied with non-functional requirements it was decided to roll out our own solution, which will be tailored exactly for the business needs.

4 Analysis of implementations and architectures

4.1 Analysis of different implementations for mobile app development

As was described in chapter 3.1 it was decided to create a solution that is exactly tailored for business needs. Nowadays, there are numerous possibilities to create a web info system. In this chapter, the analysis of how it could be done and what options are available will be created.

To create a high-quality mobile application that will be used by employees it is important to fulfill given functional and non-functional requirements. It is very important to choose the most appropriate mobile application development strategy. To do so the analysis of each mobile application development approach will be produced.

After a brief analysis, it appeared that the landscape of different mobile platforms and development strategies has seen significant changes in recent times. Each development approach may significantly boost or slower development of application and strongly affects performance, UI and development cost of application. [13]

And currently there are 4 main popular and well known approaches [14] [13] in the mobile application development:

- **Native application development approach** – "developers implement an application for one specific target platform using its software development kit (SDK) and frameworks. The app is tied to that specific environment." [14]

- **Cross-platform application development approaches** – "cross-platform approaches allow developers to implement an app as a single code base that can be executed on more than one platform." [14]. Notable example of framework supporting this approach is React Native.

- **Hybrid application development approaches** – "hybrid approaches emerged as a combination of Web technologies and native functionality. Their runtime environment largely consists of a Web rendering engine, wrapped in a native engine. The source code of hybrid apps uses similar technology like Web apps but additionally

has access to an API for platform-specific features." [14] Currently there are two most popular frameworks supporting this approach: PhoneGap and Apache Cordova.

- **Mobile Web application development approach** - "Mobile Web application implemented with HTML, CSS, and JavaScript use the browser as their runtime environment and thereby capitalize on the good browser support of mobile platforms. When using this approach, developers implement their application as one Web site optimized for mobile devices, which the Web browser then interprets" [14]

Each of those approaches provides its own drawbacks and advantages. Mostly there is a trade-off between the time of development and performance of application – Native applications are usually faster and more performant than Web applications, however, they are much slower in implementation. [14]

For fulfilling our non-functional requirement "Application should be developed as quickly as possible with the minimum usage of human resources." I need to evaluate the advantages and disadvantages of each development approach. I compiled a table from various sources ([13] [14]) which could be used for quickly evaluating various properties of each development approach. This table could be used for making a decision on which approach could suit business goals better.

Decision criterion	Native approach	Cross-platform approach	Hybrid approach	Mobile Web approach
Quality of UX	Excellent	Not as good as native apps	Excellent	Not as good as native apps
App development cost	High	Not as high as native	Not as high as native	Low
Ease of updating	Complex	Medium	Medium	Simple
Time to market	High	Medium	Medium	Short
Developer availability	Medium	?	?	High

Table 13 Comparison between various mobile application development approaches

As you can see from the table above, to fulfill the business requirement "Application should be developed as quickly as possible with the minimum usage of human resources." I will evaluate a mobile web application approach at first. Compared to the other approaches ease of development of mobile web application is higher than for the

native approaches and at the same time, the cost of development is lower. The main reasons for that are well-established JavaScript, CSS, HTML community and high quality of documentation. [14]

In combination with the other requirements from chapter 2.1 and especially with the requirement "Application should work both on Android and iOS" it appears that for fulfilling our business goals with minimum time and resources it is reasonable to begin development by using the "Mobile Web application development" approach.

To compare the properties of Mobile Web development with business goals provided by the client I created a table, which clearly shows how each decision criterion of the Mobile Web development approach fulfills business requirements:

Decision criterion	Mobile Web approach	Requirements coming from business side
App development cost	Low	Application should be developed as quickly as possible with the minimum usage of human resources.
Ease of updating	Simple	Application should be updated without prompts with one click.
Time to market	Short	Application should be developed as quickly as possible with the minimum usage of human resources.
Developer availability	High	There should be large pool of software developers who could enhance the given MVP.

Table 14 Comparison between mobile web approach and business requirements

Summing up, Mobile Web approach has one unmentioned advantage: Because codebase is HTML, JavaScript and CSS it is possible to switch from Mobile Web approach to Cross-platform approach (which also uses HTML, JS, and CSS) without rewriting all

existing code. Mobile Web approach has more agility in terms of app development costs and does not require any specific knowledge about iOS and Android.

As a consequence, if a development of employee tracker proceeds with the use of web mobile development approach it is important to analyze what kind of sensors does browser APIs offer, what technologies could be used, analyze possible implementation problems, analyze sensor compatibility across various browsers as well as possible performance implications of this approach.

4.2 Analysis of needed sensors and their API-s

To determine sensors needed for analyzing the activity of a user or vehicle I referred the article "Using Mobile Phone Sensors to Detect Driving Behavior" [15] That article explains that sensors such as accelerometers, GPS, and microphone could be used for determining driving behaviour on the roads of India. The article confirms that it is possible to analyze the turning of a car and braking only by using an accelerometer. Activities such as using a car honking are possible to collect via microphone. Combining all sensors and their data it is possible to give a conclusion and measure where the traffic jams occur. Similar article "TrafficSense: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones" [16], explains how we may combine sensor data altogether and mix it up, to reason about some condition, such as braking, or even save battery power by turning some sensors off, when some sensors are redundant in some particular reasoning situation.

Above mentioned articles use accelerometer, microphone, GSM radio, and/or GPS sensors for monitoring. However, since those articles were written some years ago (most of them in the period of 2011-2014), but nowadays smartphones received even more sensors. In our case, we need a similar solution, but we could use more sensors and more appropriate sensor APIs for each situation. To determine which sensors could be used we need to analyze the list of all available sensors.

Mainly the most respectable source of information regarding all available sensors for mobile web application is World Wide Web Consortium (W3C) and MDN. Here is the list of sensor and storing APIs I found which could be used for our solution:

4.2.1 Environmental Sensor API-s:

- Geolocation API [17] - This API enables the user to ask the location of a mobile device. What is important that it already has watch method, can use GPS sensor (if GPS is turned on). Most important that it can be enabled only on HTTPS sites due to security reasons. It also requires user to accept the tracking permissions.

- DeviceOrientationEvent [18] - The deviceorientation event is fired when fresh data is available from an orientation sensor about the current orientation of the device as compared to the Earth coordinate frame. This data is gathered from a magnetometer inside the device.

- DeviceMotionEvent [18] - provides web developers with information about the speed of changes for the device's position and orientation.

- Devicelight API [19] - It appears that mobile devices can detect change in ambient luminosity. This api may be suitable to detect environmental changes - like going from the street into the room and vice versa.

- Proximity API [20] - With proximity sensor you can detect how far your phone is from some object. For instance, it could detect do you hold your phone in the hands or near your ear.

- Humidity API [21] - this API is experimental. You can get a humidity percent of a device. It is unclear how many phones support this feature.

- Navigator interface [22] - represents the state and the identity of the user agent. It allows scripts to query it and to register themselves to carry on some activities.

4.2.2 Useful but not mandatory API-s that could be used:

- Battery Status API [23] - You can gather the battery/power consumption statistics, current battery level etc. It might be useful for measuring of a efficiency of a polling. This API does not require permissions from users.

- Webworkers API [24] - This API enables the javascript to execute code in multiple threads. It also could help with making the polling work even if the user locks the phone. Also it may be required for running various polling scripts in the background

- and also take advantage of multicore phones.

- Page Visibility API [25] - By using this interface we could tell If browser tab is idle or not. When the app becomes idle, we could wake it up by sending appropriate signals or notifications. This API does not require any special permissions from the user permissions.
- Websockets API [26] - Connection between the client and server should be energy efficient. Maybe it would be much better to constantly keep connection between them instead of sending and receiving separate requests.

4.2.3 Data storing API-s

- IndexedDB API [27] - is an object store that can save and retrieve data through JSON objects [28]. Probably this is one of the best options for my application because some sensor data might highly unstructured (due to failure/disabled state of sensors). Because app should work offline, the data should be stored somewhere. I suppose if there will be a decent amount of sensors enabled, then we will could out of 5MB of localstorage quite quickly. However MDN states that: “IndexedDB API is powerful, but may seem too complicated for simple cases. If you'd prefer a simple API, try libraries such as localForage, dexie.js, ZangoDB and JsStore that make IndexedDB more progrmmer-friendly”

- Application cache - actually, this maybe a very useful option if application stays offline, it provides possibility to cache resources and to control which fallbacks could be used if there is no internet connection.

- Web storage (localstorage) - it is a key value store for storing JS primitives (supports only strings and numbers). Limited to 5MB of data.

- Web SQL database - abandoned and not maintained [28] .

- File system - abandoned and not maintained [28].

4.3 Analysis of sensors compatibility for different browsers

From the previous chapter, it became clear that browser offers more than enough sensors and APIs to locate and determine the activity of the user.

However, there are many browser implementations and not all of them support those APIs in the same way. Before starting any kind of development we need to analyze how currently most popular browsers support those APIs.

To evaluate this risk we need to find and compare how different browsers support their sensor APIs. Currently it was required to target 2 cross-platform browsers. Client Company agreed that we could choose Chrome and Firefox:

- **Chrome**: because it is most popular cross-platform mobile browser worldwide

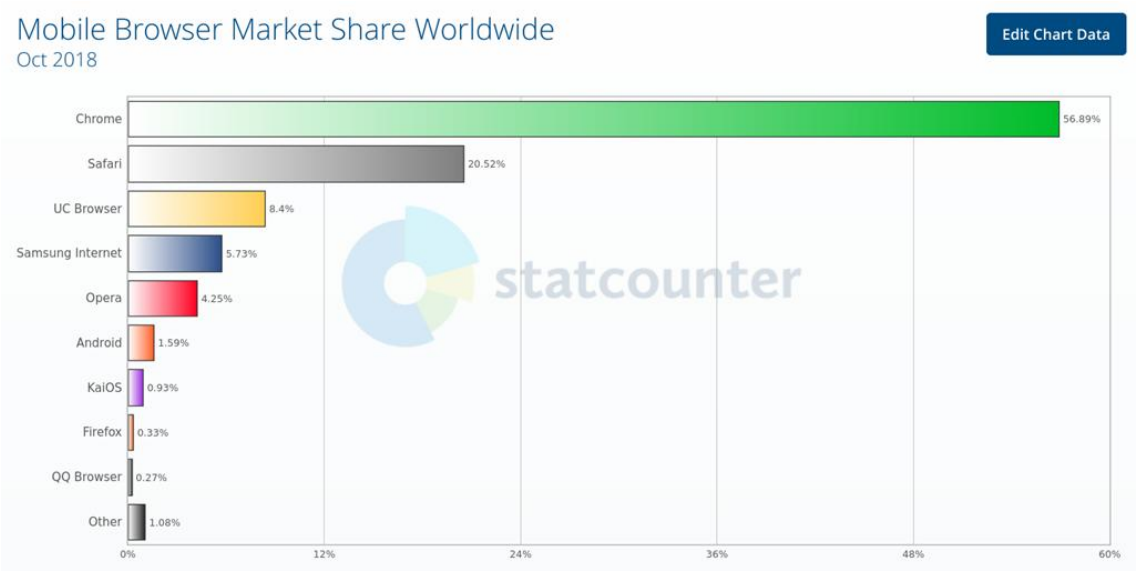

Figure 11 Browser usage chart (Source: StatCounter Global Stats for October 2018)

- **Firefox**: not because of its popularity, but because of well established organization, supporting of standards, and open source license – in our situation Firefox is used as a fall-back browser, mostly if something is wrong with end-user mobile Chrome, then the user could install Firefox and continue working.

Both of those browsers could be installed on all needed (for our purposes) and most popular mobile operating systems: Android and iOS.

However, this does not mean that the other browsers (like Safari or Opera) are not supported; their sensors will be accepted and used as well if they implement W3C sensor standards and expose sensors APIs – supporting every possible browser is not a main focus for the beginning, but a good goal for the further development.

The bigger problem is the fact that there are differences in sensor availability for each browser.

Table below provides information on each browser sensor availability for Mobile Chrome 70.0.3538 and Firefox 63 compiled from MDN and W3C:

Web API / Browser	**Mobile Chrome 70**	**Mobile Firefox 63**
Geolocation [17]	Yes	Yes

Battery [23]	Yes	No (Old specification?)
Deviceproximity [20]	No	Yes
Deviceorientation [18]	Yes	Yes
Humidity [21]	Not supported yet	Not supported yet
Devicemotion [18]	Yes	Yes
Devicelight [19]	No	Yes
DeviceLightEvent [19]	No	Yes
Web workers [26]	Yes	Yes
Page Visibility API [25]	Yes	Yes
Navigator [22]	Yes	Yes

Table 15 Availability of sensors for different browsers

From the table above we can see what output we could expect from each browser – Firefox and Chrome will act differently in the same execution environment: Firefox supports more sensors such as Deviceproximity, Devicelight, and DeviceLightEvent, but does not support new Battery Sensor API standards.

This information is valuable for testing our solution – we will not expect output from a browser which does not have support for a particular sensor.

Nonetheless, most important sensors from the business point of view are Geolocation, Battery, deviceproximity, deviceorientation, devicemotion, and navigator – are supported by both Chrome and Firefox and we will target them in a first place. This means that there is no considerable risk in choosing a mobile development approach over native.

However, because availability of sensors is different across browsers the response of the APIs will be different for each browser – in the long term, it makes things more complicated if we will support more and more browsers.

4.4 Analysis of possible implementation problems and limitations

Sensor availability analysis revealed that there are hidden risks for a web mobile development approach – difference across browsers in terms of sensor API support. Let's analyze other risks as well:

The biggest problem I can see is the lack of the mechanism to query all sensors at once as stated in the Generic Sensor W3C document [29]. The second biggest problem is the conflicting support of all the sensors and APIs across different browsers [30].

The third problem comes when the phone becomes idle, locked or browser is not used actively. Even then phone should send at least some amount of sensor data. This means that I should find some way to keep the browser awake even in those scenarios. Possibly wakening of the browser could be achieved with push notifications. Problems with the phone becoming idle probably could be solved with Webworkers API, by keeping the process in a separate thread, which will be never suspended.

The fourth problem is the synchronization of data between server and client – the client must have support for offline mode – synchronizing the aggregated data with server could be a complicated task. However, there are solutions tailored for those kinds of problems. One of them in to use PouchDB [31] and CouchDB. PouchDB is javascript browser in-memory database that can sync with CouchDB and vice versa. This technology solves the main problem with the fact that the app could lose internet connection but has to re-establish it when it gets online and resend/sync the data with the server.

There are many more issues addressed in article "Software Engineering Issues for Mobile Application Development" [32] such as:

1. "Do mobile web applications behave differently when connected using the telephone network (3G, 4G) than when using an 802.11 (WiFi) or 802.16 (WiMax) connection? Are there differences in security? Is there a significant difference in responsiveness? Are traditional fallback and exception-handling techniques adequate, or does the higher likelihood of a dropped connection (or intermittent connectivity) require additional mechanisms?" [32]

2. “Are there new techniques needed for assuring data integrity, or will the synchronization techniques from traditional client-server computing suffice? Does potential loss of connectivity or battery power represent a risk to program and/or data integrity if such an event occurs during a transaction or system update?” [32]

3. “Should applications be designed differently depending on the speed of the network on which they are being used? In Asia, some countries offer rates of 50Mb or higher, while typical speeds in the US, even with 3G networks, are below 1 Mb.” [32]

4. “How does a developer create applications that will maximize battery life and resource usage?” [32]

To sum up, all those questions will be taken into account during development phase to provide stable and reliable piece of software.

4.5 Analysis of web mobile development performance

Clearly the performance of the browser is lesser than of native code [33]. However, it is not clear how much performance differs between Chrome and Firefox on different platforms such as Android and iOS.

According to the article “Mobile application development: web vs. native” [4], I have to take into account is the fact that generally there could be a huge performance difference between IOS and Android devices. For instance, in the article [4] iOS performed better in Sunspider JS benchmark and Android device in V8 benchmark [4]. This means that profiling the code may be a quite complex and obvious task.

In my personal opinion, the performance of solution relies mostly on optimization done by a programmer who does the app. For our purposes, it is important to create a solution that does not drain the battery too quickly and provides adequate performance.

4.6 Conclusion: web mobile development

After analysis of all risks, it appeared that web mobile development approach is a perfect candidate for our MVP solution: from both business and software development side, it addressed all the possible issues quite well.

It is not only my personal point of view, big corporations and academic papers share my opinion as well: for instance, in the article “Mobile Application Development: Web vs. Native ” [4] you could get the idea why web platform is the major platform for future applications: At first, even big corporations, such as Google cannot support all possible platforms [4] due to economic reasons. This is because of the fact that web applications are more cross-platform by default [4] - you do not have to write a separate program for each SDK, however, compared to native approach, UI may look alien and not that pretty compared to native app development [4].

Regarding of the future of web first development approach it could be assumed that it will continue to improve further: number of sensors is growing and it appears that various companies and organizations in the past, such as Nokia, Google, Mozilla, and IEEE are experimenting with implementing even more sensor functions, for instance, detection of walking or using of a vehicle, detection of current noise [34]. Or even determining what is going on in the front of the camera by using the computer vision and OpenCV library [34].

Because for further development web mobile development approach was chosen, solution could be adjusted and extended with those following new technologies and approaches just by implementing and incorporating new sensor APIs into the client-side.

In conclusion: web mobile development approach makes our solution easy to develop, support and enhance with minimal price and effort, this makes mobile web development a perfect candidate for further development.

4.7 Analysis of different architectures

In general bigger part of modern web application architectures could be classified as:

- More monolithic / less service-oriented – application could be deployed and scaled only as one big service. The downside of this architecture is inability to scale particular service parts of an application. Only whole application could be scaled to the maximum extent of any possible request – you will have to scale the whole application if there is a bottleneck in any particular case. However,

there are many advantages of such architecture: ease of deployment, overall stability due to independence from other services and networking. [35]

- More (Micro) service oriented / less monolithic – application parts could be deployed and scaled independently; also this approach follows carefully one of the most important SOLID principles: SRP (Single Responsibility Principle). There are numerous downsides of this approach as well: more complex deployment model, less stability in case of network outages, more complex monitoring and logging. [35]

In my opinion, for the MVP more monolithic architecture is a better option (at least in the beginning). What is more important to apply Modern Web Architecture patterns:

- Single Page Application – takes advantage that nearly every browser runs JavaScript, we could decouple View Layer from backend and scale backend independently from the view layer. It also allows much more responsive and intuitive application designs than the dynamic page approach [35]
- Near Cache - use a Near Cache located within the client implementation. Cache the results of calling the Backend for Frontend services so as to reduce unnecessary round trips to the server [35]. It is also possible to cache the whole frontend for offline use.

From point of perspective of architecture patterns there are a couple of choices: [36]:

- Layered architecture pattern – Provides good separation of concerns for each layer and in tackles complexity.
- Event-driven architecture pattern – Event-queue based architecture, mostly suitable for micro services. Provides good performance and scalability of services.
- Microkernel architecture pattern – Mostly good for desktop systems, might be overly complex and unusual for web applications.
- Space-based architecture pattern – Provides good performance for relational databases but is costly to implement [36]:

In general, it is considered good to start from layered architecture pattern because “it good general-purpose architecture and a good starting point for most systems” [36]

In our particular case, architectures and patterns should be chosen for ease of development rather than for scalability or performance. Those architectures and patterns will let develop and deploy MVP with higher velocity.

4.8 Conclusion

During the following chapter in-depth analysis of most important aspects of application was performed: analysis of existing solutions, analysis of various ways, problems, APIs and architectures for our application. With all those topics covered we will proceed to the most practical part of the thesis – implementation.

5 Implementation

In this chapter, the comparison and choice of databases, platforms, and frameworks will be performed. The choice will be conducted according to the analysis.

5.1 Databases

It is important to use the correct database for a particular problem. In our case those problems and requirements are:

- Support for offline mode in case of client internet connection outage
- Synchronization of a client with the database in case of client internet connection recovery
- Support of storing highly unstructured sensor data
- Strong and stable write performance of unstructured data (sensor information)
- Consistency in case of important business data – user management, metadata etc

To solve those problems, we will use 3 separate databases, each of will serve its own specific purpose:

- CouchDB and PouchDB – document-oriented NoSQL databases with eventual consistency and near-real-time client to server replication. By using them it is relatively easy to achieve replication to the server with almost “real-time” websocket speeds. Those databases are an excellent choice for storing and retrieving unstructured sensor data. Both of those databases could power an admin dashboard, where you will see user position movement almost in real-time.
- Postgres – relational SQL database that will be used for everything where consistency is important and guaranteed: user management, authorization, storing of meta information, etc. Great for ensuring the integrity of relational data.

5.1.1 CouchDB and PouchDB

In order to make solution offline ready it is important to store GPS data locally in case of a network outage and send missing data back to the server when the network appears. One of the easiest and most complete solutions found is to use local (embedded into browser) database called PouchDB and its server-side twin brother called CouchDB. The most important feature of those databases is a seamless synchronization/replication support [37], which assists in a development of offline real-time reactivity – there will be no need to write code or implement complex synchronization logic between client and server and vice versa when network appears or disappears. Data is populated on the client and then is replicated to the server database automatically. However, let's analyze each database in isolation:

5.1.2 CouchDB

Apache CouchDB is an open-source database that focuses on ease of use, availability and partition tolerance. CouchDB is a NoSQL document-oriented database that uses JSON to store data, JavaScript as its query language, and HTTP calls for an API. The Apache CouchDB was chosen for storing sensor data because of easy client-server sync-replication. Apache CouchDB is a scalable and mature technology both in terms of performance and support. For instance, core technology powers NPM (Node packet manager) – one of the largest software package registries. The most important features are: Multi-version concurrency control, eventual consistency, map reduce and multi master-replication.

For solution implementation, CouchDB offers a large number of convenient features. One of those features is easy user and database management. For instance, each user in our system has its own CouchDB database instance (which stores his locations). Let's create a database, user and assign a user to the database, making database private and readable only by the user. To create a database I simply call:

```
await nano.db.create("somedatabase");
```

Figure 12 New CouchDB database creation code snippet

To create a new CouchDB user I call the code:

```
await nano.use("_users").insert({
      "_id": "org.couchdb.user:" + ”someuser”,
      name: “someuser”,
      type: "user",
      roles: [],
      password: “somepass”
    });
```

Figure 13 New CouchDB user creation code snippet

Assigning CouchDB user is also not complex (each database has security meta database):

```
await nano.use("somedatabase/_security").insert({
      "admins": {
          "names": [“someuser”],
          "roles": []
      },
      "members": {
          "names": [“someuser”],
          "roles": []
      }
});
```

Figure 14 Granting access of particular database to user - code snippet

Another convenient feature of CouchDB is build-in admin dashboard:

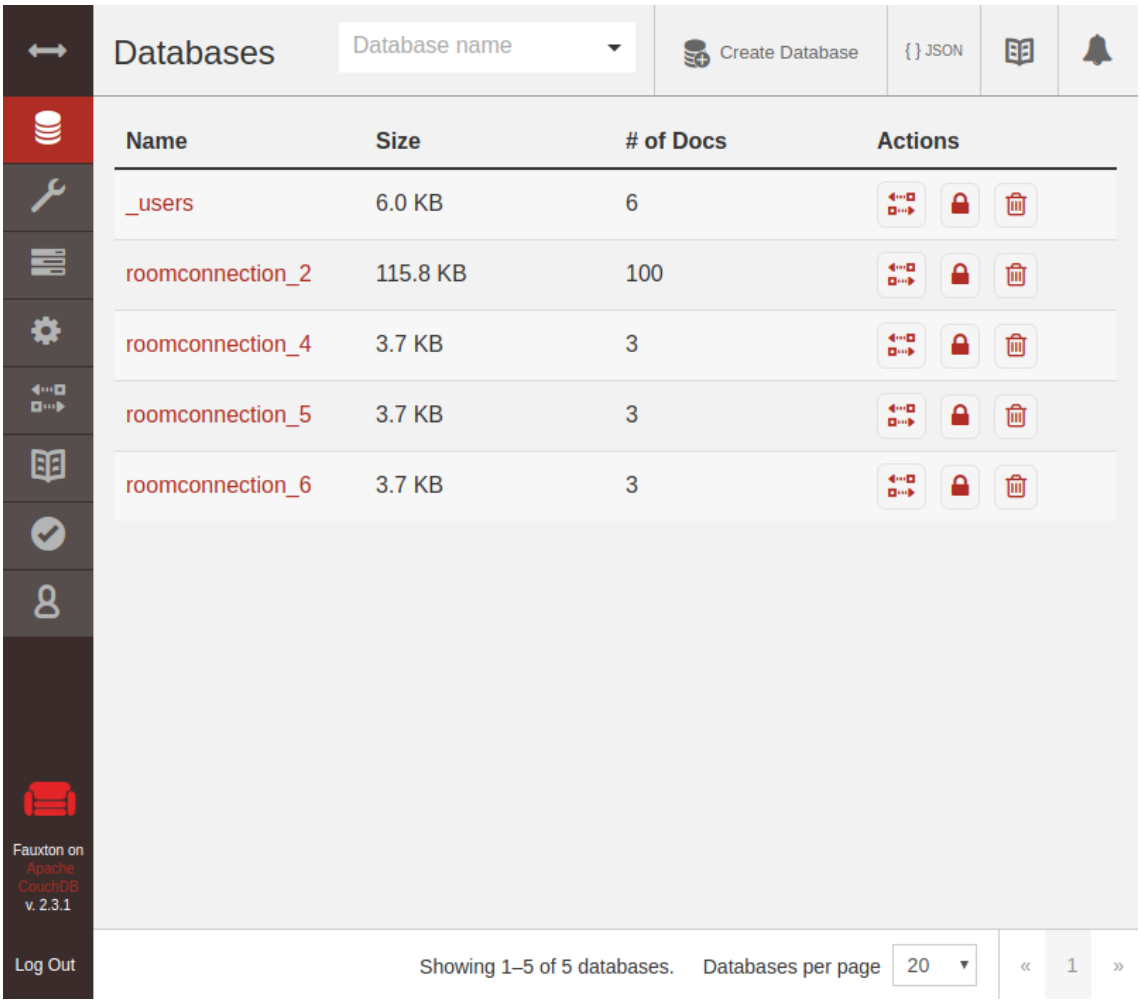

Figure 15 CouchDB admin UI screenshot

5.1.3 PouchDB

PouchDB is an API-compliant implementation of CouchDB that can work in a browser environment. It uses Indexed DB API, localstorage and other fallback technologies (like cookies) to store data in a browser. The main reason it exists is easy replication with CouchDB: you do not have to worry about restoring the connection or sending updates in a batch: everything is done for you automatically. This is great for the following scenarios: Client has sent the sensor data to the server, but lost internet connection, when internet reconnects, PouchDB automatically knows what missing pieces of data it should resend to the backend.

PouchDB shares similar and compatible API with CouchDB and is very easy to replicate with CouchDB:

```
const db = new
PouchDB(`https://someuser:somepass@localhost:5984/somedatabase`);
```

Figure 16 Creating new PouchDB instance code snippet

To insert a record PouchDB shares same API with PouchDB:

```
db.put({
          _id: app.get("user")+ new Date().toString(),
          ...sensor.serialize(state),
          user: app.get("user"),
          dateTime: new Date()
        });
```

Figure 17 Inserting into record into PouchDB code snippet

Synchronization between PouchDB and CouchDB is done automatically.

5.1.4 Postgres

Postgres is a well known open-source relational database. Postgres will be used for storing everything except sensor information, where real-time synchronization support is crucial. For instance: user management, sessions, authorization, metadata, destinations, etc.

Postgres was chosen because of zero cost, good support and great amount of tutorials, tools, admin dashboards, good support and integration with almost any backend, platform, language, and technology.

5.2 Backend platform and framework

When choosing the backend framework we must take into account following important properties:

- The platform must have a large pool of developers
- The platform must have support for real-time interactions
- The platform must have strong support of NoSQL (CouchDB) and SQL databases (Postgres)
- The backend framework must simplify and enhance development velocity

- The more functionality framework (Authorization and authentication, validating of JWT tokens, etc) provides the better it is.

- The platform must have TypeScript support

5.2.1 NodeJS

Node.js is a JavaScript run-time environment that executes JavaScript code outside of a browser. The NodeJS will be used mainly because of its great support of real-time applications and websockets, good integration with CouchDB and the fact that you do not need to learn an additional programming language: both backend and frontend could be created with JavaScript. This particular feature is convenient for some developers and brings benefits to IT support – the same workforce/developers could be used for creating frontend and backend simultaneously.

5.2.2 FeathersJS

FeathersJS is a web framework for NodeJS that emphasizes real-time support, RESTful resources, sockets and flexible plug-ins. FeathersJS leverages all benefits given by NodeJS: easy creation of RESTful resources transmitted via HTTP or Websockets. One of the key benefits is easy integration with almost any database: CouchDB, Postgres, Mongo, Oracle, etc. The framework also provides easy authentication and authorization, validating of JWT tokens with the numerous amount of plugins, good support of RESTful resources, command line code generator and front-end client.

Most important is that FeathersJS is well suited for rapid application development. Each service created with FeathersJS creates a RESTful resource. The logic of the services could be customized as well. For instance, the following code:

```
class Users extends Service {
  constructor(options: Partial<KnexServiceOptions>, app: Application) {
    super({
      ...options,
      name: "users"
    });
  }

  async create(data: any, params: Params) {
    const presentUsers = await this.find({ query: { email: data.email } });
    if (presentUsers.total > 0) {
      throw new Error("This e-mail is already taken");
    }
    const user = await super.create(data, params);
    const account = await app
      .service("accounts")
      .create({}, { transaction: params!.transaction });
    await app
      .service("memberships")
      .create(
        { userId: user.id, accountId: account.id },
        { transaction: params!.transaction }
      );
    return user;
  }

  async remove(id: number){
    throw new Error("Not supported");
  }
}

export default function (app: Application) {
  const Model = app.get('knexClient');
  const paginate = app.get('paginate');

  const options = {
    Model,
    paginate
  };

  // Initialize our service with any options it requires
  app.use('/users', new Users(options, app));
}
```

Figure 18 User service implementation code snippet

will produce a customized RESTful resource with 7 endpoints:

Service method	HTTP method	Path
service.find({ query: {} })	GET	/users
service.find({ query: { activated: true } })	GET	/messages?activated=true
service.get(1)	GET	/users/1
service.create(body)	POST	/users
service.update(1, body)	PUT	/users/1
service.patch(1, body)	PATCH	/users/1
service.remove(1)	DELETE	/users/1

The service could be consumed with ordinary HTTP calls or with a higher level FeathersJS frontend client:

```
app
        .service("users")
        .create({ email, password, fullname })
        .then(
          () => {
            this.props.history.push("/actions");
          },
          (error: { message: string }) => {
            this.setState({ ...this.state, error: error.message });
          }
        );
```

Figure 19 Interaction with backend through FeathersJS client library – code snippet

Summing up, chosen framework makes creating of web services easy fast and simple by reducing the amount of boilerplate code.

5.3 Frontend framework

Frontend Framework must comply with following terms:

- Great amount of already premade plugins, components – for rapid application development
- Integration with PouchDB
- Good real time support
- Typescript and JavaScript support

5.3.1 React

ReactJS is a JavaScript library developed by Facebook intended for creating single page applications. React was chosen because of good community support and significant amount of prebuilt components and UI libraries.

React integrates well with PouchDB and basically with anything JavaScript related. Whole Admin UI will be built with React because there will be a lot of real-time interaction with the map and users. Client UI will also be built with React because, how it was mentioned before, React integrates quite well with PouchDB and FeathersJS.

In addition to above-mentioned, React offers a splendid ecosystem of different libraries, a decent amount of tutorials and community support. In my particular case, I will use Ant Design framework to bootstrap a user interface.

Below you could observe the implementation of a user GPS location tracking page. The page receives credentials of CouchDB, replicates them to PouchDB and sends all mobile phone sensor data to backend every 5 seconds:

```
// @ts-ignore
import allsensors from "allsensorsjs";
import React, { useState, useEffect } from "react";
import { DefaultLayout } from "../components/defaultLayout";
import { pouch } from "../shared/pouch-client";
import { authenticatedApp } from "../shared/auto-auth-client";
import { CardForm } from "../components/cardForm";
import axios from "axios";

const Log = ({ match }: { match: { params: any } }) => {
  const [sensorState, setSensorState] = useState("");
  const [address, setAddress] = useState("");
  useEffect(() => {
    let sensor: any;
    authenticatedApp().then(async app => {
      const roomMetaData = await app
        .service("log-room")
        .get(match.params.roomId);
      sensor = new allsensors.GlobalSensor({ queryPeriod: 5000 });
      sensor.listen((state: any) => {
        pouch(roomMetaData.account.couchUser, roomMetaData.account.couchPass,
roomMetaData.roomConnection.couchDbName).put({
          _id: new Date().getTime().toString(),
          ...sensor.serialize(state),
          user: app.get("user"),
          dateTime: new Date()
        });
        axios
          .get(

`https://nominatim.openstreetmap.org/reverse?format=json&lat=${state.geo.coor
ds.latitude}&lon=${state.geo.coords.longitude}`
          )
          .then(response => {
            setAddress(response.data.display_name);
          });
        setSensorState(JSON.stringify(sensor.serialize(state), null, 4));
      });
    });
    return function cleanup() {
      if (sensor) {
        sensor.listeners = [];
      }
    };
  }, []);

  return (
    <DefaultLayout showSidebar={false}>
      <CardForm title=”Monitoring page”>
        <span>Your location: { address }</span>
      </CardForm>
```

```
    </DefaultLayout>
  );
};

export default Log;
```

Figure 20 Tracking page source code

5.4 Component integration and communication

In the analysis above, architecture, frameworks, and components for the proposed solution were chosen. The deployment diagram below shows the relations and communication protocols of those subjects.

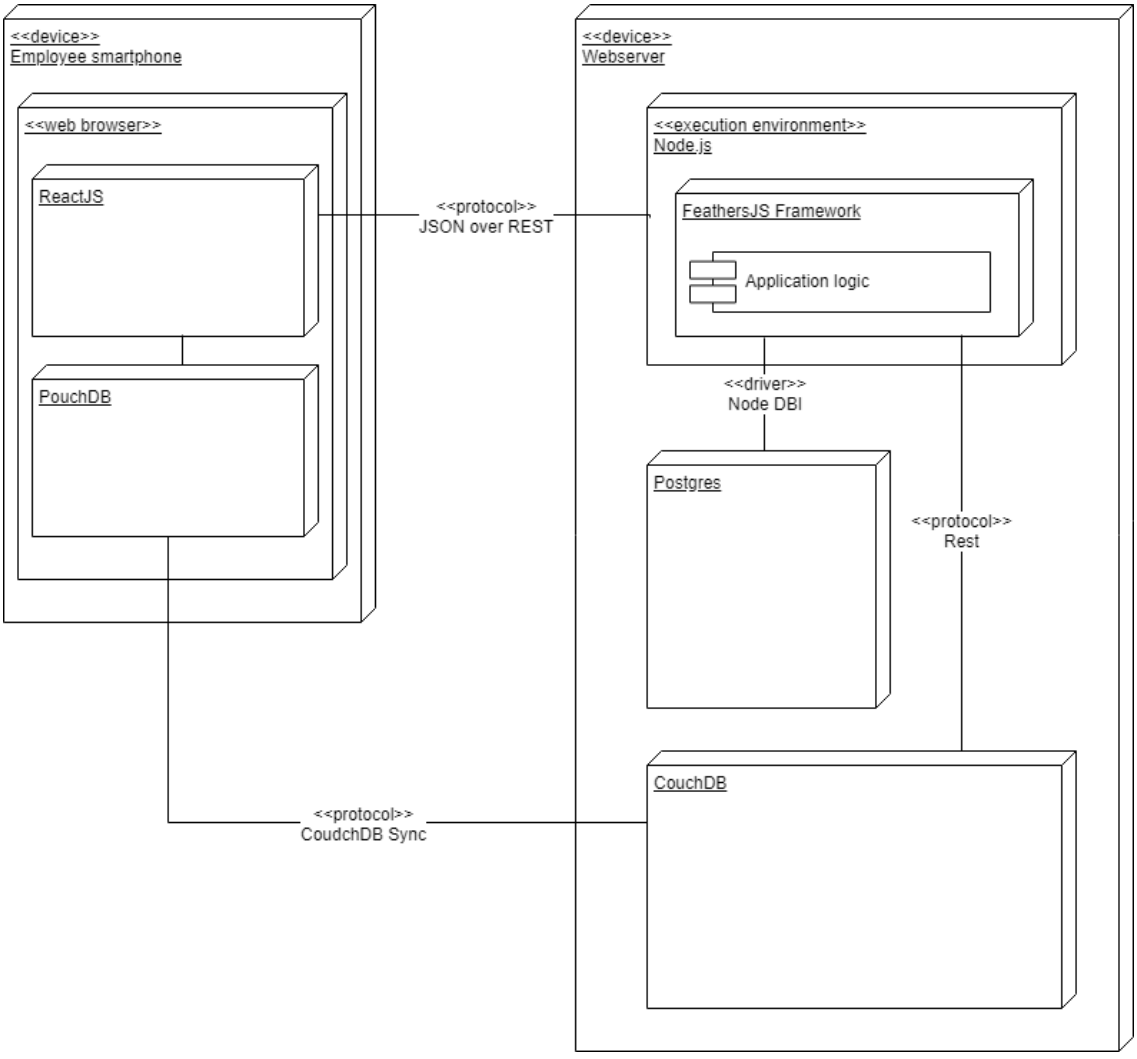

Figure 21 Deployment diagram

The browser will send whole information via 2 channels:

- Via JSON over REST to backend, where the backend will process the data and save it to Postgres.
- With PouchDB via CouchDB Sync protocol for real time sensor information

Both Databases will be connected to backend service. This model will ensure that our development velocity will be optimal and performance will be excellent, since we use appropriate databases for appropriate data models.

5.5 Use cases

5.5.1 User signup

To signup, user must enter his email, full name, and password. The user must confirm his email and admin must validate user identity and email manually. After sign up, user is automatically redirected to the login page. On the login, user sees information about how to confirm his email and identity. This process is similar across all competing solutions.

User signup process is illustrated below.

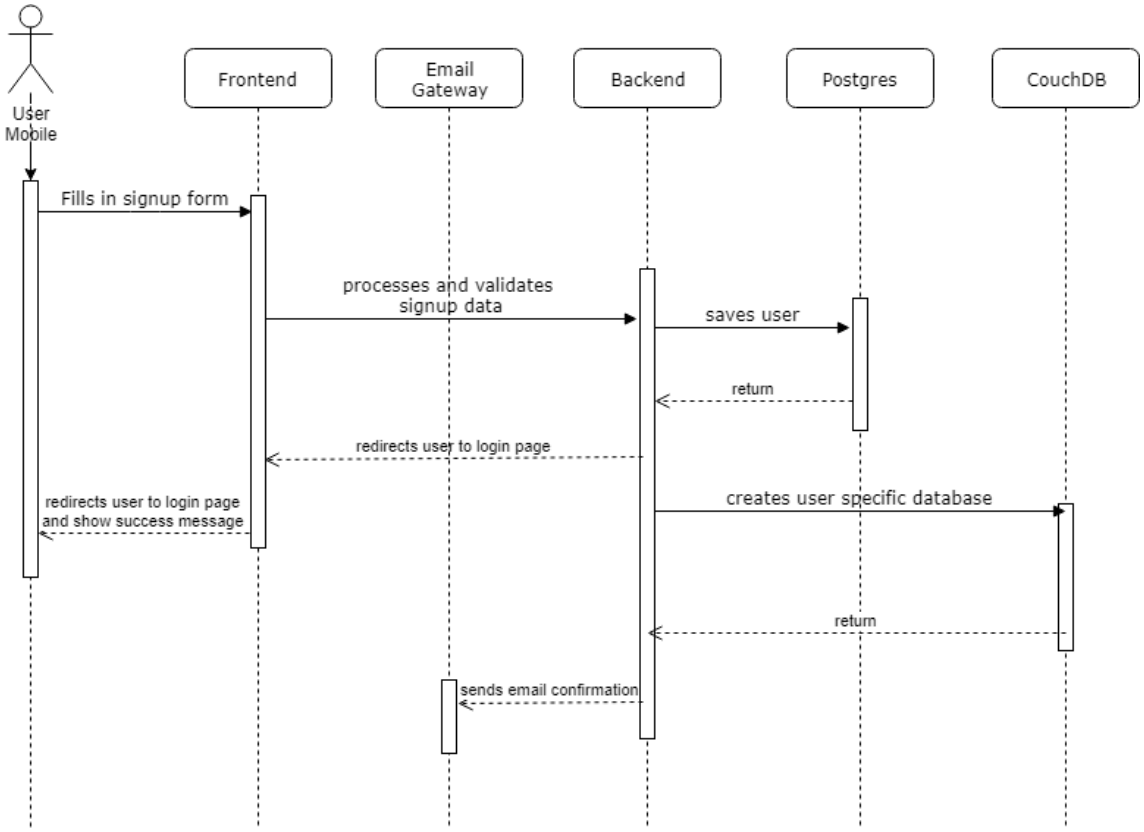

Figure 22 User signup sequence diagram

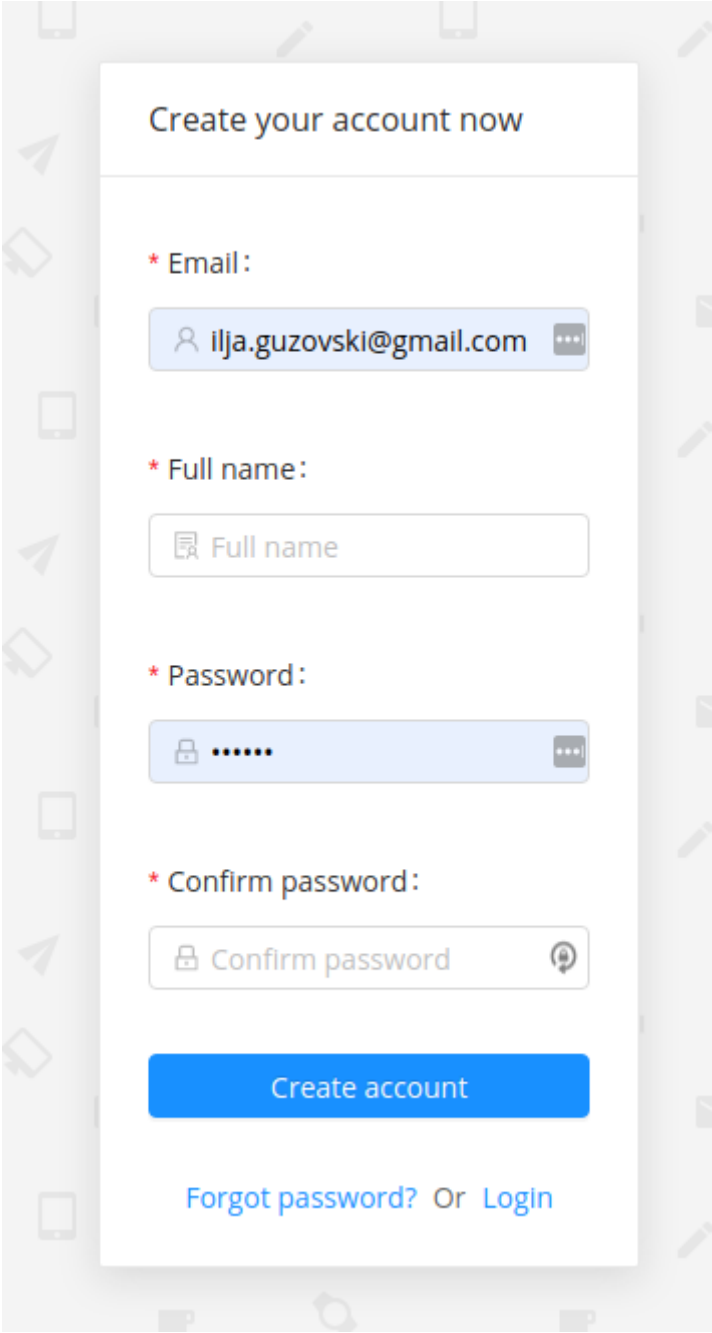

Figure 23 Signup page UI

5.5.2 User login

To login, user must enter his email and password. If the login was successful then the user is redirected to the monitoring page. The user login process is illustrated below. This process is similar across all observed solutions and between all web applications in general.

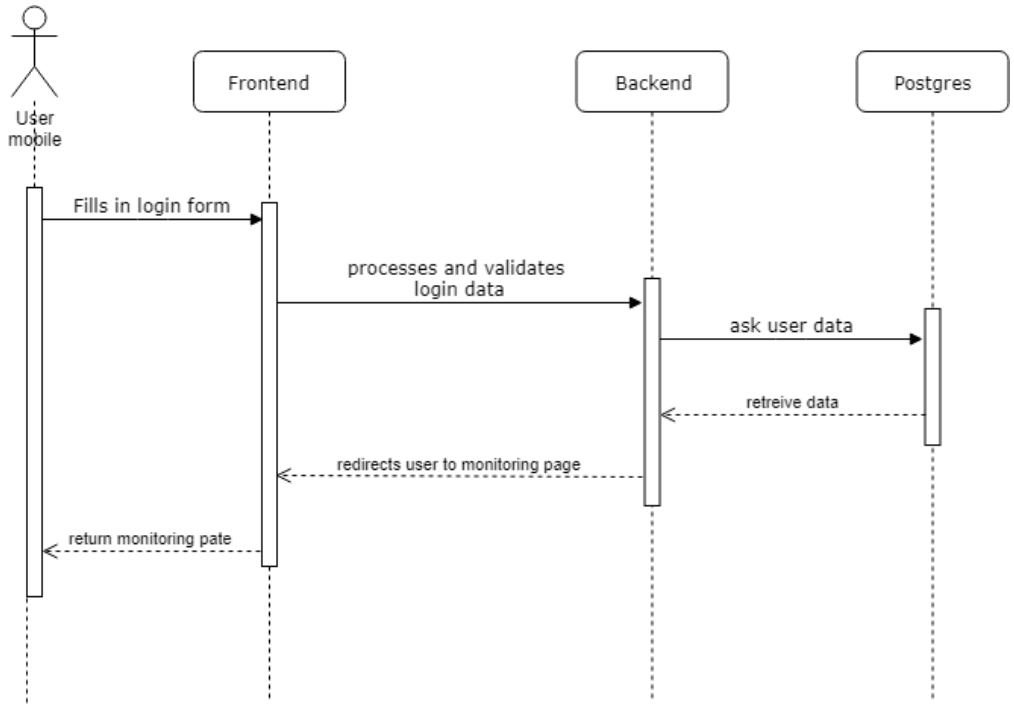

Figure 24 User login sequence diagram

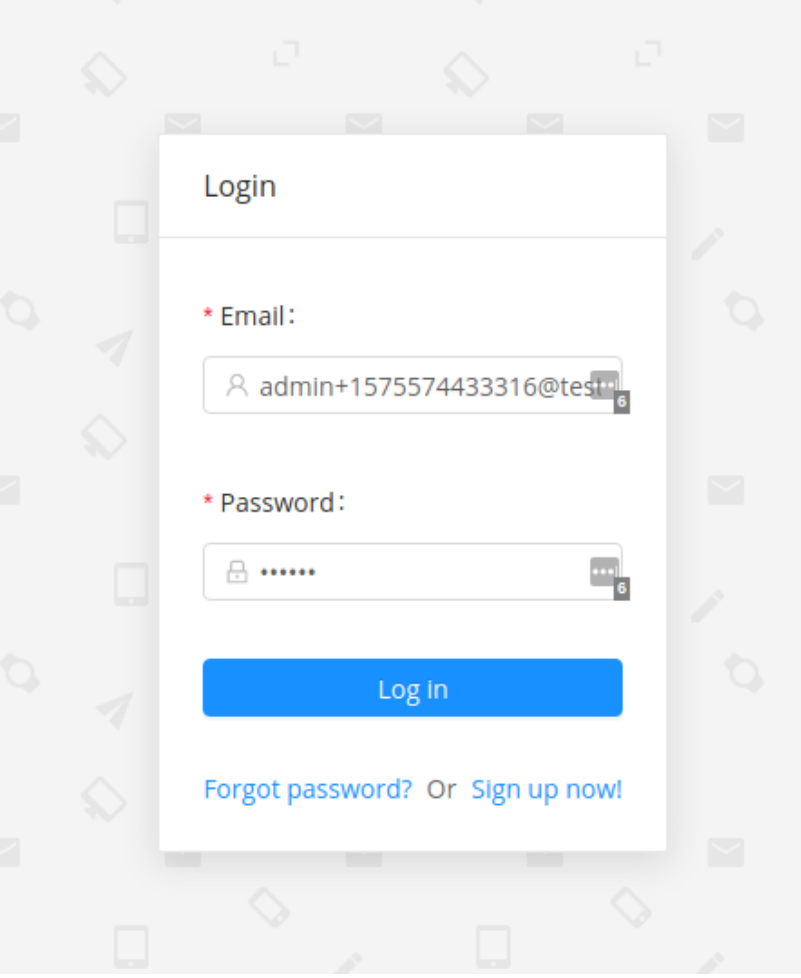

Figure 25 Login page UI

5.5.3 User location/sensor tracking (FR-1, FR-2, FR-3, FR-4, FR-6)

To share location and all other related sensor data (such as proximity, device orientation, device motion, etc) user must be logged in and proceed to a special monitoring page. On the monitoring page user must manually turn on monitoring by pressing the button “Start monitoring”, browser may prompt if he agrees to share his location, the user must agree to proceed. Sensor tracking process is illustrated below. This process is different from other competing solutions (like traccar or Spyzie etc) because it happens in the browser and does not require a native app. PouchDB ensures that location data could be written even if user is offline or if user internet connection is poor. PouchDB directly syncs to CouchDB and could restore connection if it was interrupted. Each user who is monitored has its own CouchDB database – in this case deleting user location data will be as simple as possible. Postgres, in this case, is used as metainformation storage – it stores location URL of CouchDB database that is linked to the user.

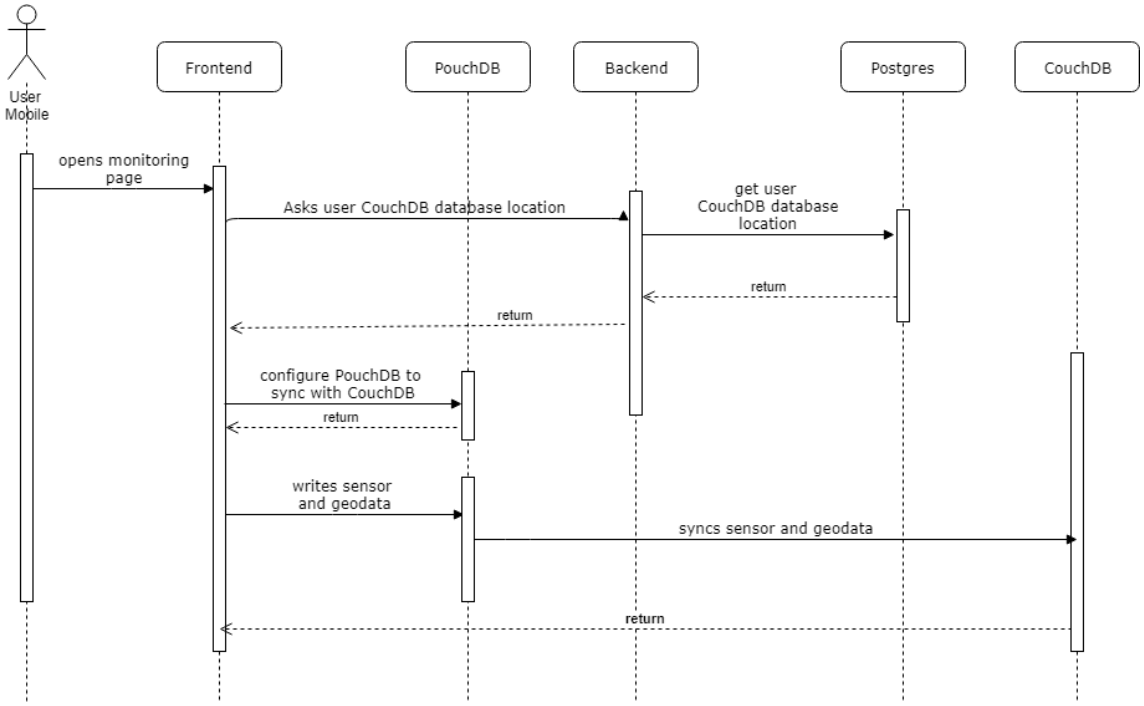

Figure 26 User location tracking sequence diagram

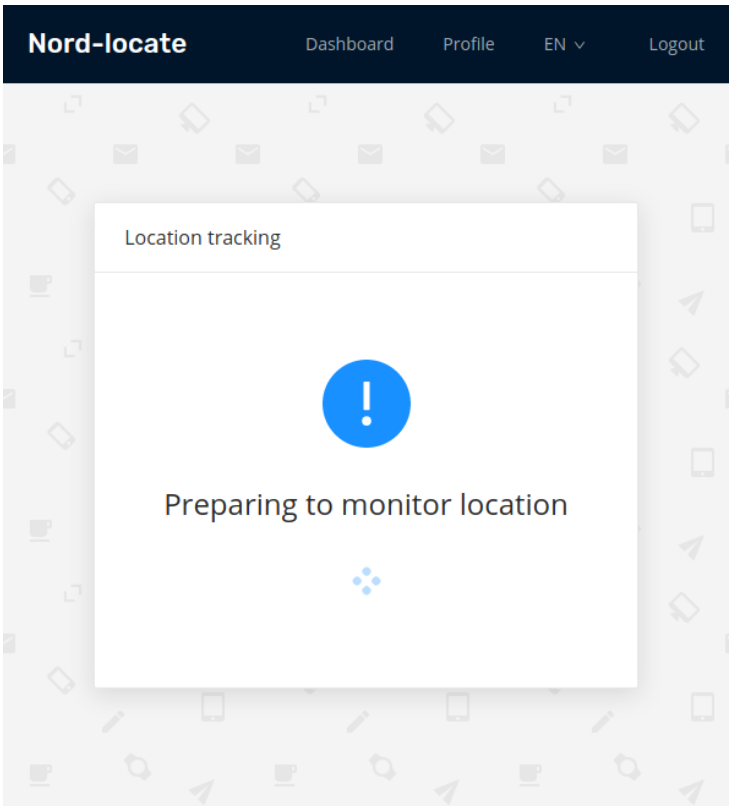

Figure 27 Monitoring page UI

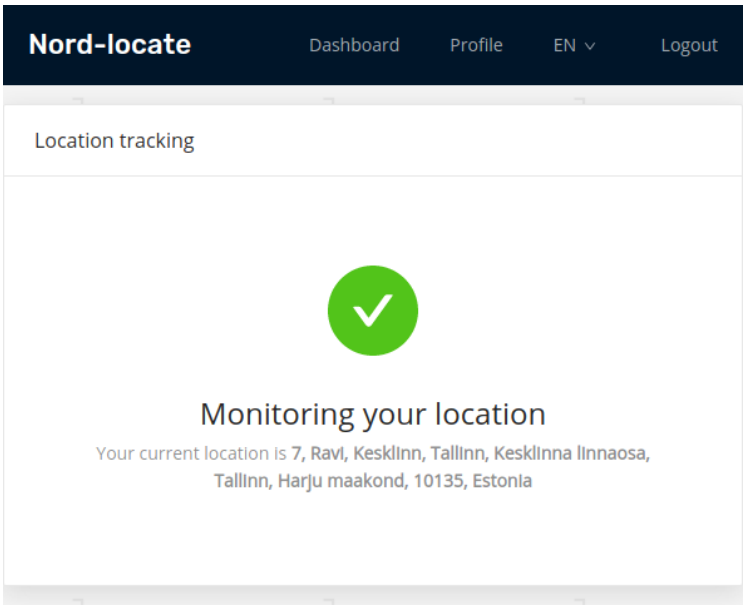

Figure 28 Monitoring page UI – monitoring/sync successful (2)

5.5.4 User location monitoring by admin (FR-7)

To monitor user location admin must be logged in and navigated to a dashboard page. On the dashboard page, admin will see a map with various coloured markers. Each marker represents a user located at a specific point. The a dmin monitoring process is illustrated below. The point here is to ask each CouchDB database for the latest user location and combine the results. Traccar and FollowMee implement this functionality in a similar way: you could open a map and see markers – each marker is related to some user.

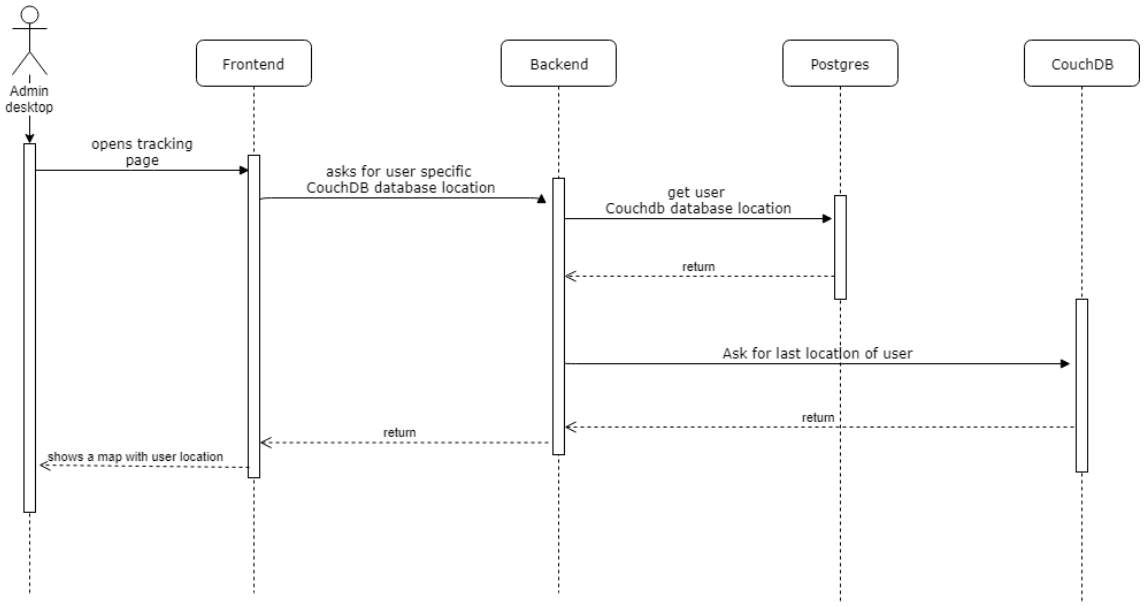

Figure 29 User location monitoring sequence diagram

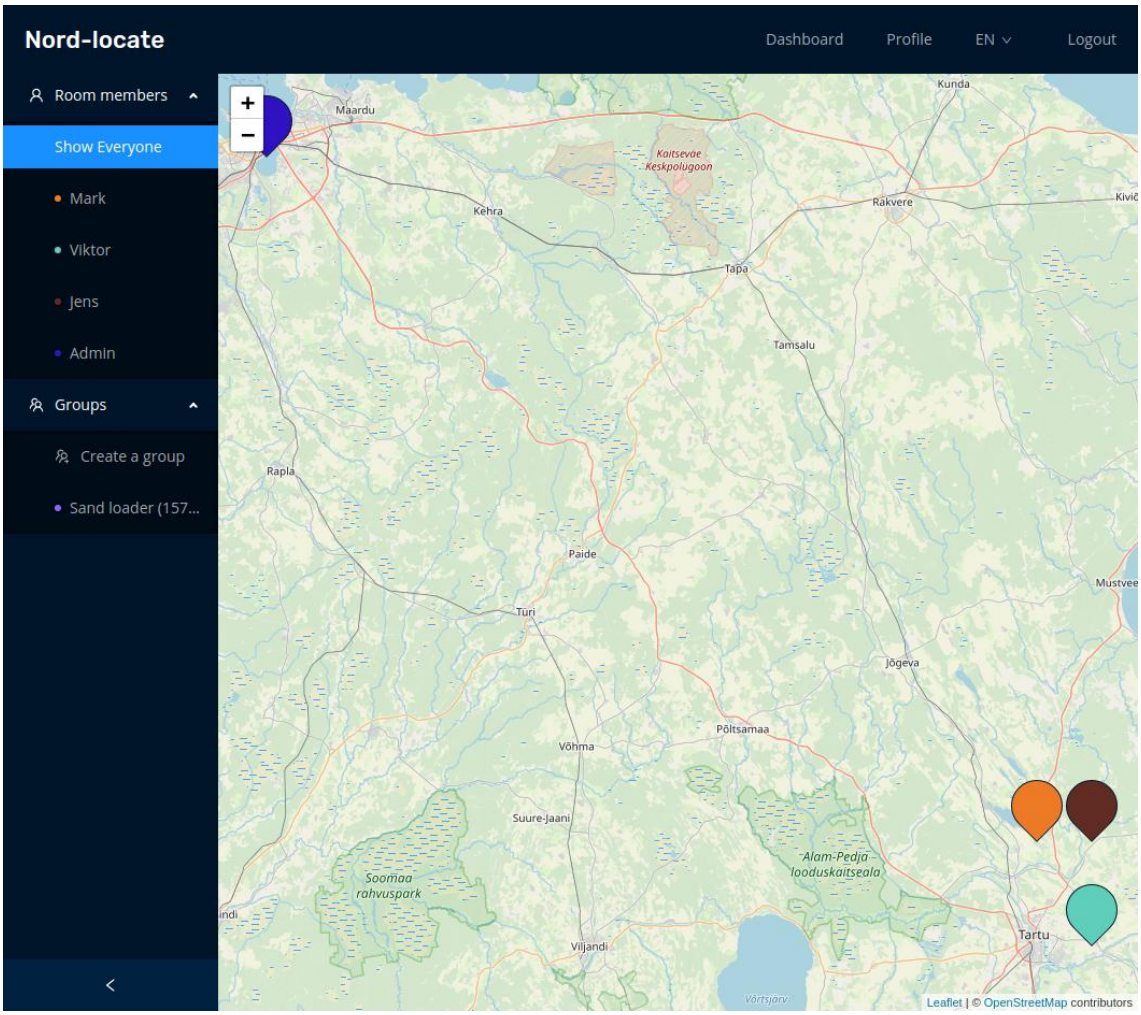

Figure 30 Monitoring page UI (Users marked as markers)

5.5.5 Set starting point and destination of user (FR-8)

Admin could set destination for user: this could be done directly from the map or from the user management page. The end result is that the user will receive notification to his mobile phone, which will tell the user where to follow. Most of the analyzed solutions (Spyzie, mSpyand other parental solutions) do not implement this functionality. In our case, the notification will be sent with the use of Web Notification and Web Push Notification API. The whole process is illustrated below. Note that for this particular case we do not need to interact neither with CouchDB or PouchDB. All trip metainformation that cannot be modified by the user is stored in Postgres.

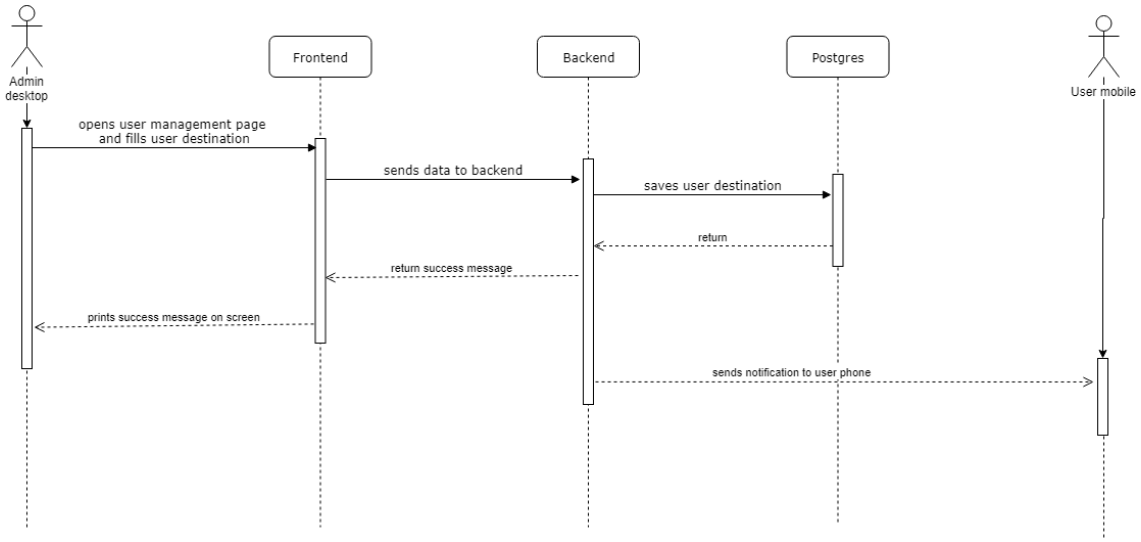

Figure 31 Set starting point and destination of user sequence diagram

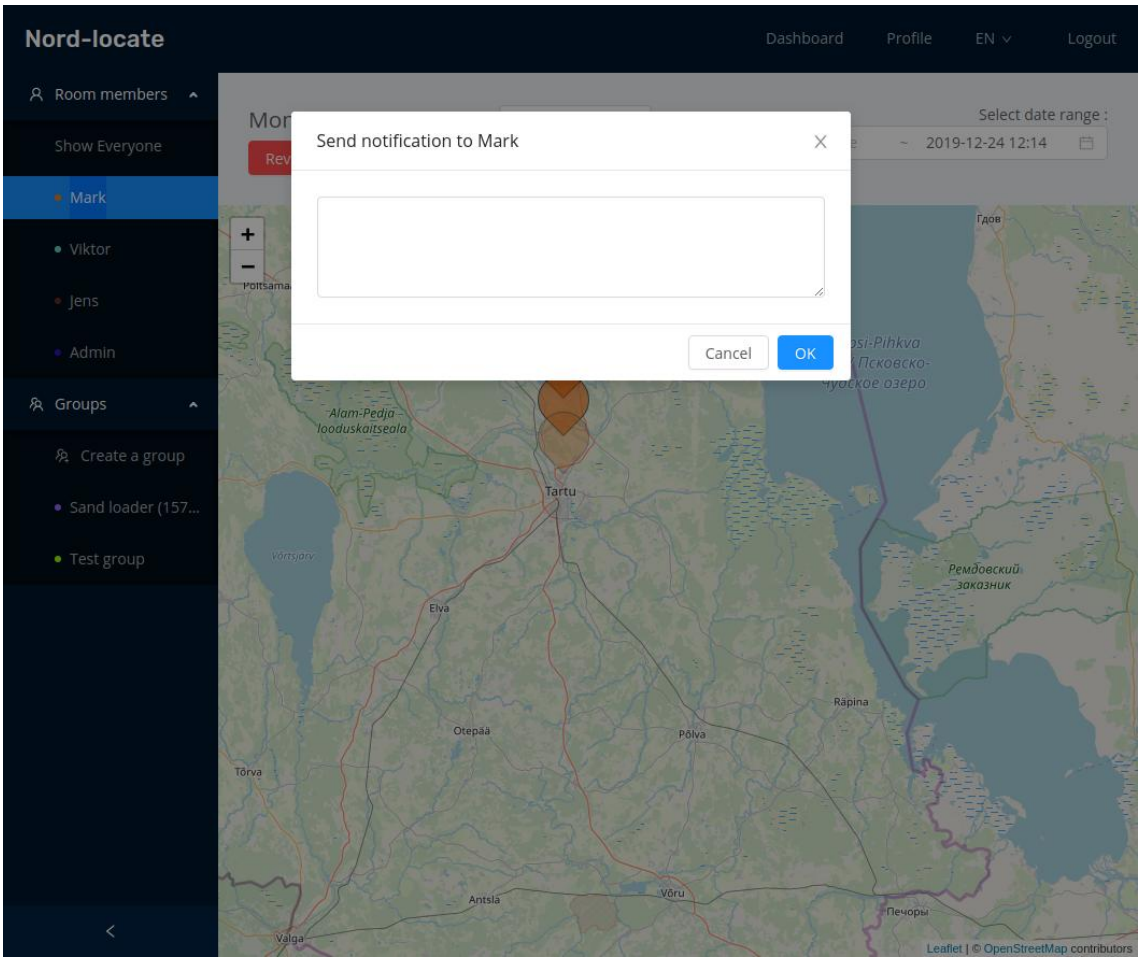

Figure 32 Send notification modal UI

5.5.6 Admin sees long term navigating history (FR-9)

Admin chooses a date and time period on the map and sees a routing of the user within given time span. Process is illustrated below. This is done in similar manner as in traccar or FollowMee. Note that each request in CouchDB is marked with timestamp, so it is relatively easy to filter this data.

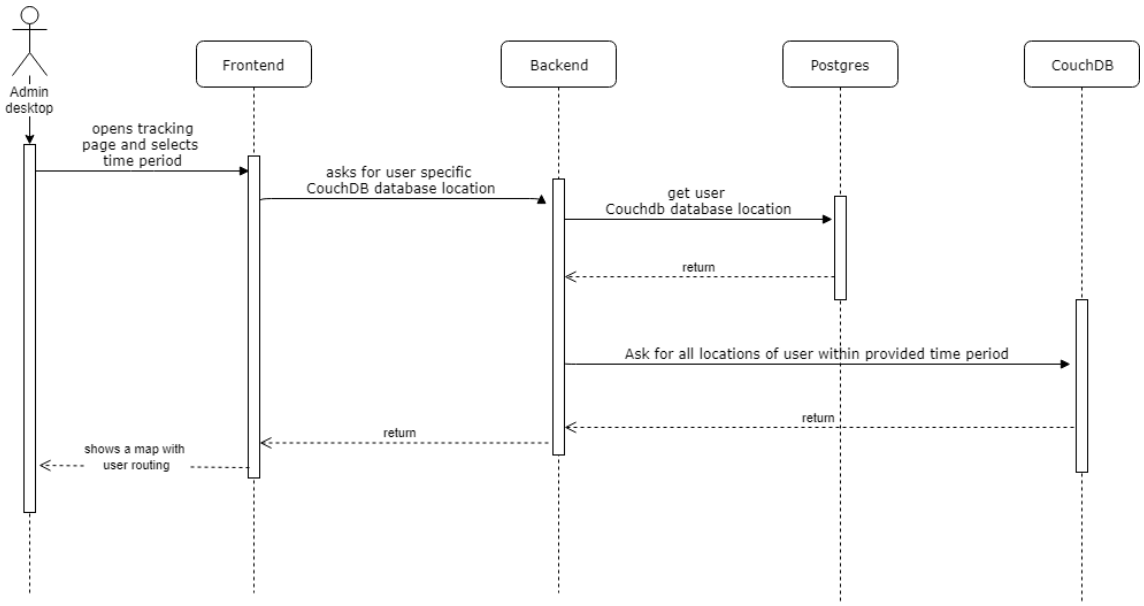

Figure 33 Admin sees long term navigating history sequence diagram

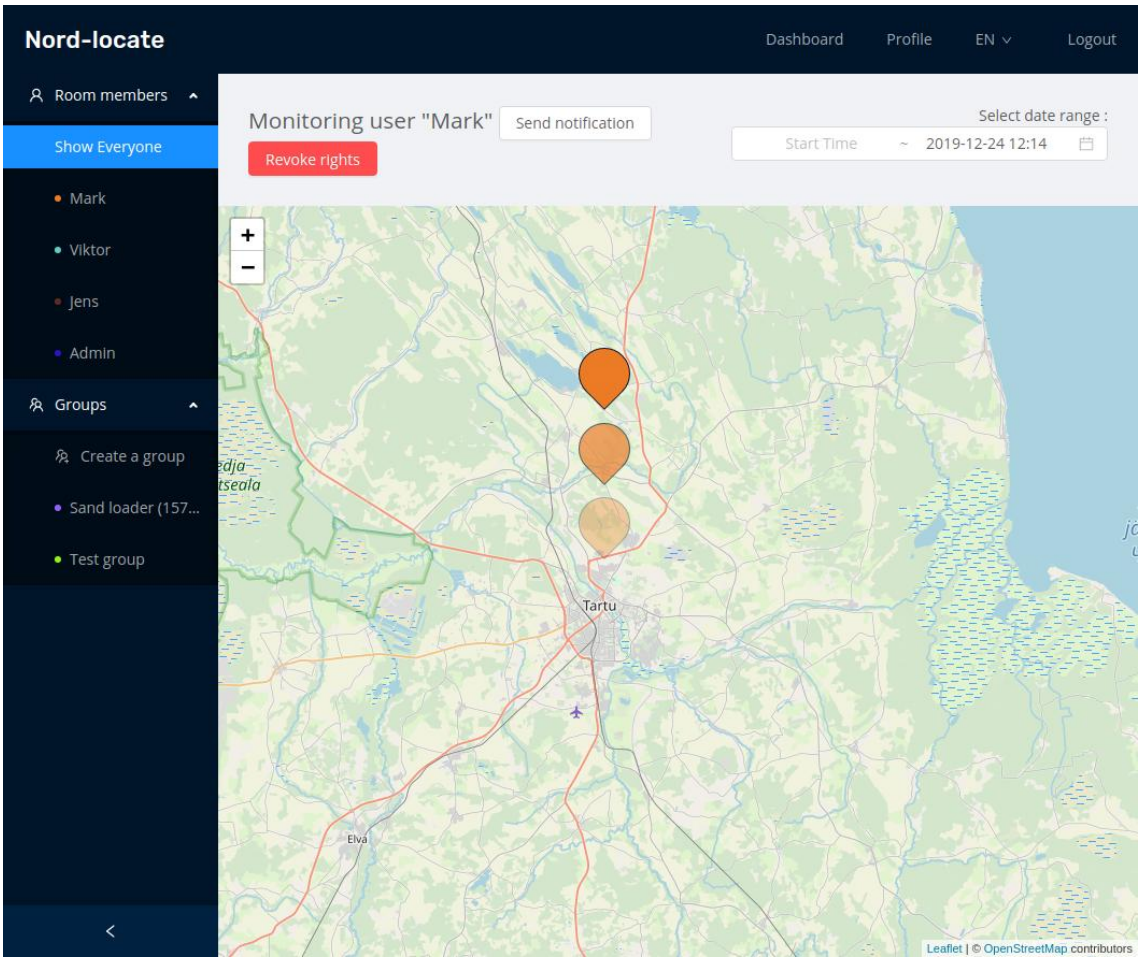

Figure 34 User history monitoring UI

5.5.7 Admin creates group of users (FR-10)

Admin opens a map page, presses “create group button”, fills name of a group and adds members with autocomplete input box. When the user presses the “create” button, the whole information is transmitted to a backend where it is written to Postgres. Other solutions (except FollowMee) do not support appropriate grouping of users. The process of a group creation is illustrated below. When the group is created admin is navigated to a page where he can see all group members on a map.

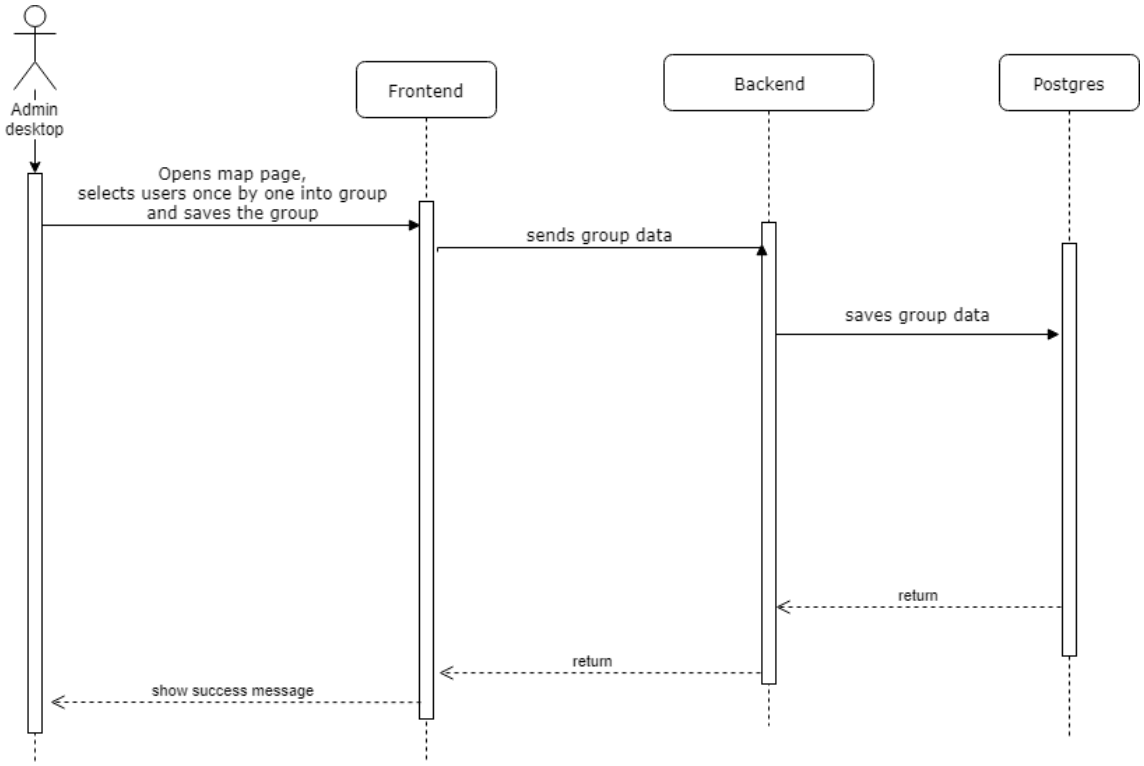

Figure 35 Admin creates group of users

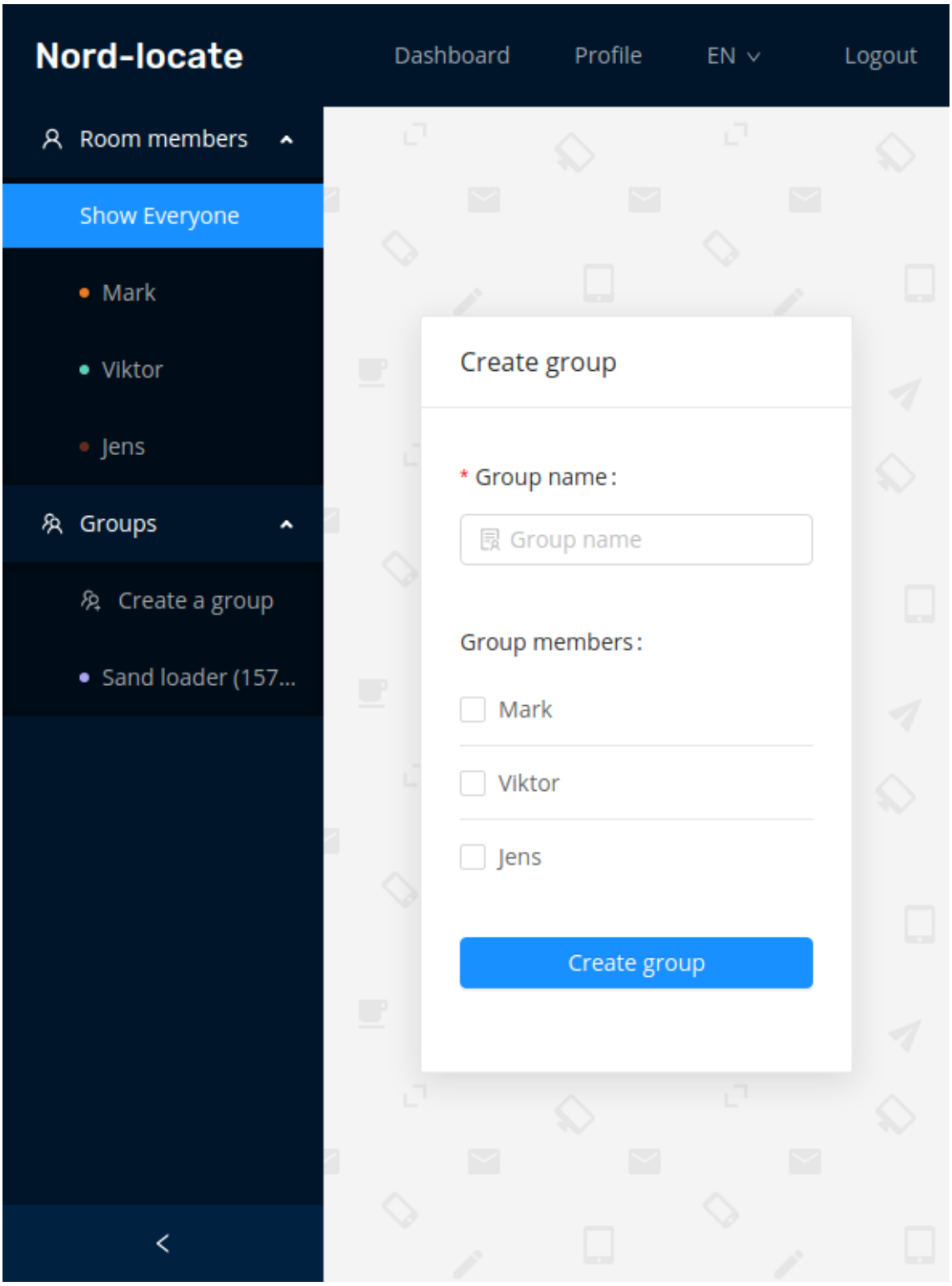

Figure 36 Create group page UI

5.5.8 Admin sees group members on the map (FR-11)

Admin could select a group directly on a map page. When the group is selected then only group members and their markers appear on the map. Technically it is most expensive operation because for each member we will ask the latest location. However, it should not be the case if the amount of group members does not exceed hundreds of members. It is also important to note that users could belong to multiple groups at once. Process of is illustrated below:

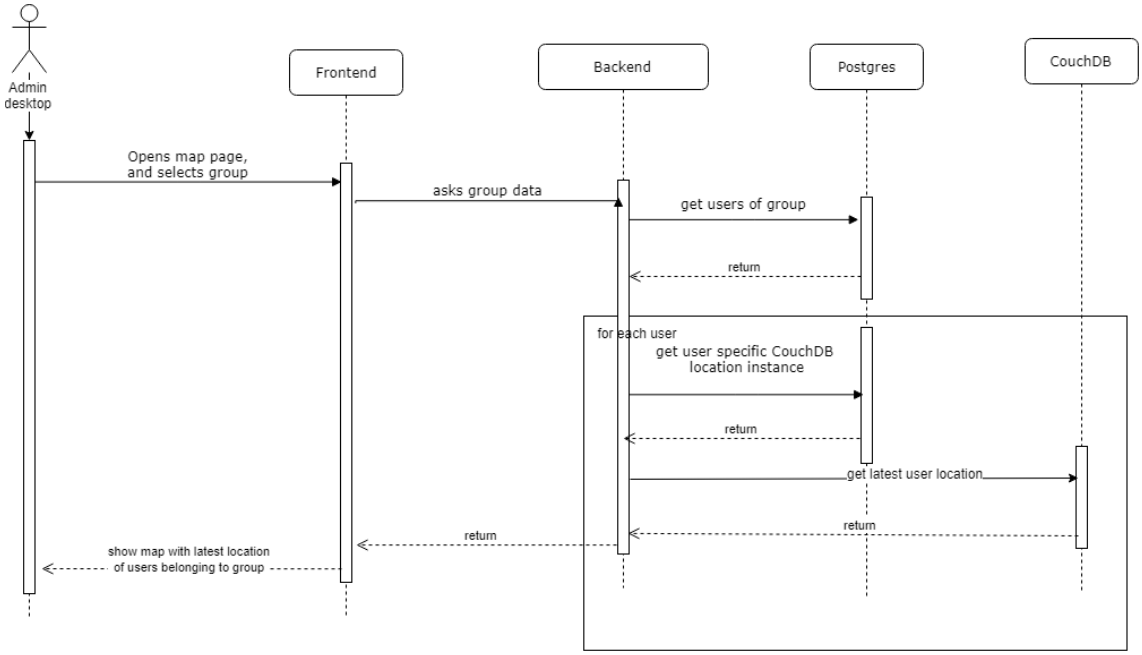

Figure 37 Admin sees group members on the map

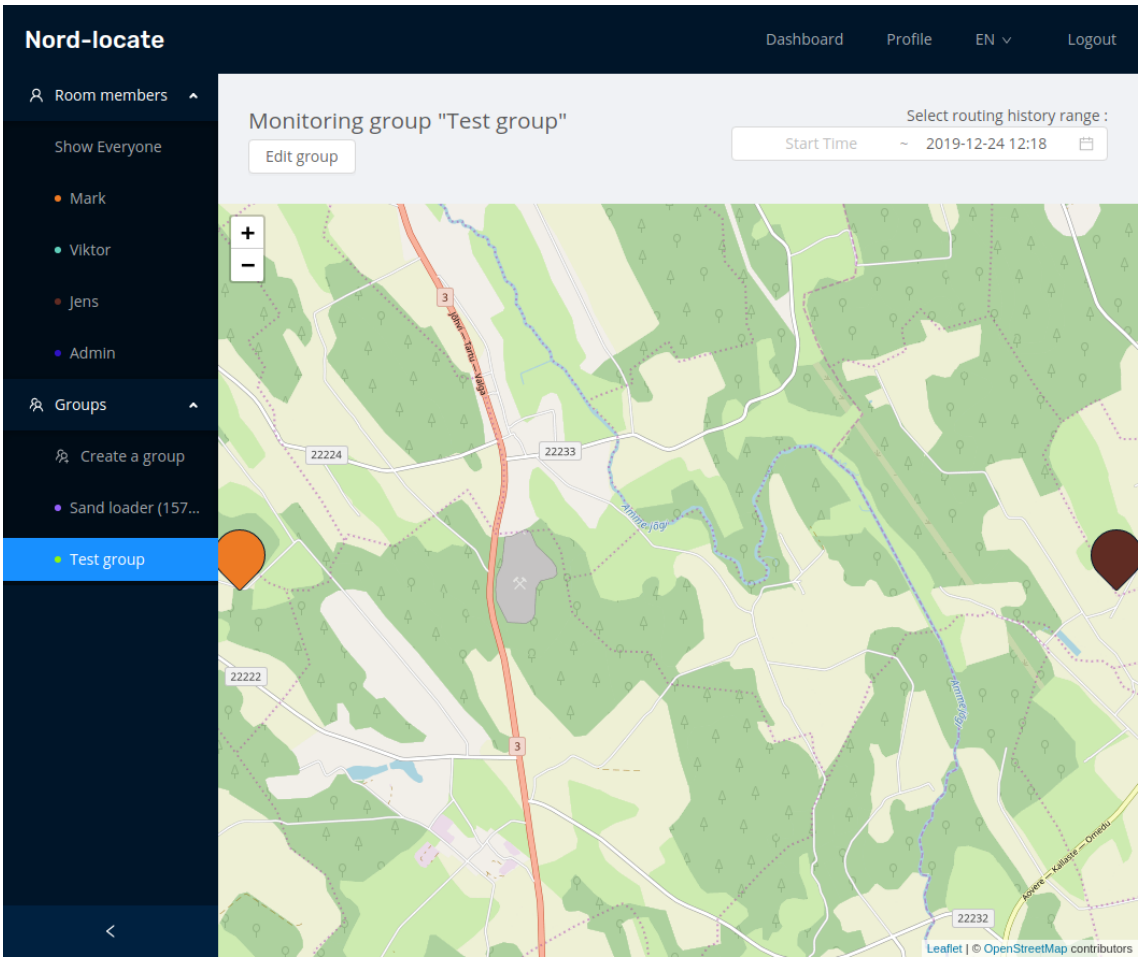

Figure 38 Show user group members page UI (group users are marked with markers)

5.5.9 Removing user data (FR-5)

Admin can remove the user and wipe out all his location data. This feature is unique and does not present in any observed solutions. Since we store user location only in users own dedicated database it is relatively easy to remove that data – by erasing it. Removing personal user data is illustrated below.

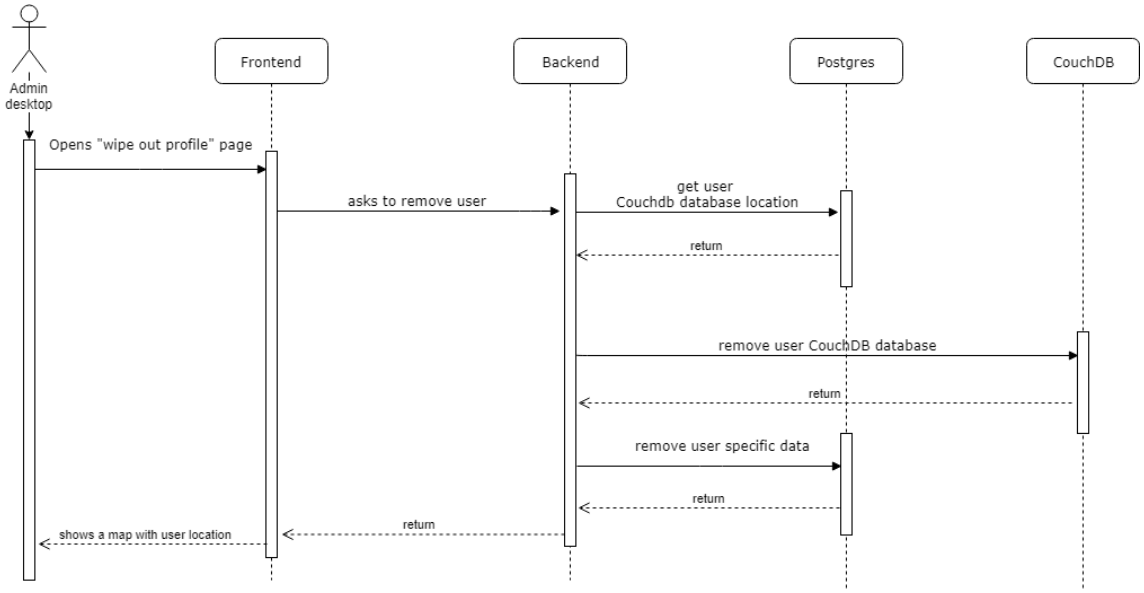

Figure 39 Removing user data sequence diagram

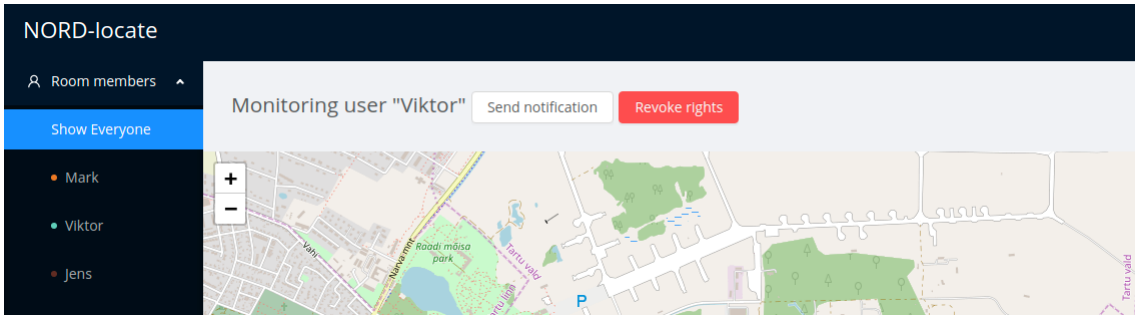

Figure 40 Delete user data (Revoke rights button)

6 Application testing

Application testing is divided into two sections: testing of functional requirements and testing of non-functional requirements. If both types of requirements comply and pass, then we may conclude that the application successfully conforms to given business requirements and goals.

6.1 Functional requirements testing

Functional requirements were tested with the use of Test Driven Development methodology – I have written most of the tests in advance to source code. Test suites consist mostly from unit tests, integration, and end to end tests. In addition to fully automated test process, some studies show that utilizing Test Driven Development may lead to better software quality in overall [38].

Both backend as well frontend were written in TypeScript which prevented some amount of defects: amount of defects and saved time was hard to measure, but subjectively it was substantial during the refactoring phase.

Backend and frontend unit tests are executed with the use of testing framework Jest.

End to End tests are executed with the use of testing framework Jest and Puppeteer. Puppeteer is a high-level API to control Chrome browser, it could be used in the same manner as Selenium web driver. Here is a small example of code that tests creating of user and admin, the recording location of the user via browser, as monitoring of location by admin:

```
const timestamp = new Date().getTime();
describe("Create and log user test", () => {
  beforeAll(async () => {
    await page.goto("http://localhost:3000/");
    jest.setTimeout(120 * 1000);
  });

  fit("should register user and admin, log user location, and show user
location on admin map", async () => {
    const roomName = "TestRoom+" + timestamp;
    const admin = "ilja.guzovski+admin" + timestamp + "@gmail.com";
    const user = "ilja.guzovski+user" + timestamp + "@gmail.com";
    const password = "123456";
    console.log("//////////////////////");
    console.log(timestamp, user, admin, password);
    console.log("//////////////////////");
    await registerAndLogin(admin, password);
    await click("#create-room");
    await createRoom(roomName);
    await registerAndLogin(user, password);
    await type("#search-room-input", roomName);
    await click("#search-room-input+span button");
    await context.overridePermissions("http://localhost:3000/", [
      "geolocation"
    ]);
    await page.evaluate(() => {
      mockGeolocation();
    });
    await page.waitFor(10000);
    const text = await page.evaluate(() => document.body.textContent);
    await expect(text).toContain("Guangzhou");
    await login(admin, password);
    await click(".room-link");
    await assertThatMapIsPresent();
  });
});

async function assertThatMapIsPresent() {
  await page.waitFor(".leaflet-container");
}

function mockGeolocation() {
  navigator.geolocation.watchPosition = function (cb) {
    setTimeout(() => {
      cb({
        coords: {
          accuracy: 21,
          altitude: null,
          altitudeAccuracy: null,
          heading: null,
          latitude: 23.129163,
```

```
          longitude: 113.264435,
          speed: null
        }
      });
    }, 1000);
  };
}

async function registerAndLogin(email, password) {
  await register(email, email, password);
  await login(email, password);
}

async function register(email, fullname, password) {
  await page.goto("http://localhost:3000/register");
  await type("#normal_login_email", email);
  await type("#normal_login_fullname", fullname);
  await type("#normal_login_password", password);
  await type("#normal_login_confirmPassword", password);
  await click("button[type=submit]");
}

async function login(username, password) {
  await page.goto("http://localhost:3000/login");
  await type("#normal_login_email", username);
  await type("#normal_login_password", password);
  await click("button[type=submit]");
}

async function createRoom(roomName) {
  await type("#room_name", roomName);
  await click("#create-room-button");
}

async function pause() {
  await jestPuppeteer.debug();
}

async function type(selector, text) {
  await page.waitFor(selector);
  await page.type(selector, text);
}

async function click(selector) {
  await page.waitFor(selector);
  await page.click(selector);
}
```

As a result some small bugs were revealed and fixed during testing phase. Below I have provided complete list of functional requirements that I have tested:

Test	Result	Comment
FR-1: Application should gather as much available sensors data as possible. Geolocation, devicemotion, deviceorientation sensor data is mandatory.	Pass	This requirement was tested with use of manual testing as well as end-to-end and unit tests. In addition to mandatory sensors, app also recorded battery information and navigator information.
FR-2: Application should continue sending sensor data in case it is in minimized or in idle state.	Pass	This requirement was tested manually. I have found that result substantially depends on mobile phone power saving mode: if it is enabled then no data is sent from browser. Probably because background data sending of browser is blocked
FR-3: Each sensor write request should include a timestamp	Pass	This requirement was tested with unit test as well as with end-to-end tests.
FR-4: Application could be reopened if it had crashed or was closed.	Pass	This requirement was tested manually. If webpage crashes or was closed, it could be reopened without login. Sending of sensor data continued after reopening the page seamlessly.
FR-5: Application could remove all user data if prompted	Pass	This requirement is covered by special script that was tested manually. All location data and user specific setting (user PouchDB database and Postgres database rows) could be wiped out from the system entirely.

FR-6: User could update application without prompts with one click	Pass	This requirement was tested just by refreshing the page. However may vary depending on a browser and how aggressively it will cache an application.
FR-7: Admin could see user position on the map	Pass	This case was covered by end-to-end test. No problems revealed during multiple runs.
FR-8: Admin could set starting point and destination for user	Pass	This case was covered by end-to-end test. No problems revealed during multiple test runs.
FR-9: Admin could see long term routing history of user	Pass?	This case was tested manually. In this particular case long term history (like day or week) is impossible to create within given small duration of end-to-end test. Some days of monitoring were correctly displayed on a map. However some locations were incorrect, during the test I never been there: this probably happened due to lack of GPS accuracy.
FR-10: Admin could group users into groups	Pass	This case was covered by end-to-end test. No problems revealed during multiple test runs.
FR-11: Admin could see group members on the map	Pass	This case was covered by end-to-end test. No problems revealed during multiple test runs.

Table 16 Functional requirements testing results

6.2 Non-functional requirements testing

Non-functional requirements were tested mostly manually. App fulfilled all non-functional requirements. Some requirements were fuzzy and impossible to test and measure objectively. Here is a complete list of non-functional requirement, requirement testing methods, results, and explanation:

Test id	Testing method	Result	Comment
1.1: Application should work on phones with OSes: iOS 9+ and Android 4.4+	Manual	Pass	This non-functional requirement was tested manually: no defects were found. Newer browser versions are still supported on older mobile OSes. Running Chrome 78 on Android or iOS did not reveal any particular differences between browser implementations for different OSes.
1.2: Application should work on most popular cross-platform browsers.	Manual	Pass	I have tested an app on Firefox 44 on Android phone, Chrome 78 on Android phone. Firefox sent lesser sensor information than Chrome. Most significant and mandatory sensor data gathering (Geolocation, devicemotion, deviceorientation) worked well on all tested browsers.
1.3 Admin could access admin dashboard from browser such as Firefox or Chrome.	E2E test	Pass	Chrome was tested via end-to-end tests. Firefox was tested manually. No significant difference between two browsers was found.
1.4: Admin dashboard should be simple to use: no actions should take	Manual	Pass?	This requirement was badly formed. It is hard to tell what “action” is. Most of complex tasks: like creating a group of

more than 4 clicks.			users does not take more than 4 clicks.
2.1: Application could be turned off for user privacy	Manual	Pass	Browser does not send data if it was closed or tab was closed.
2.2: User specific data could be deleted.	Manual	Pass	I have found that this non-functional requirement overlaps with functional requirement FR-5. All user data could be deleted with script if prompted.
2.3: Data of the users should be stored by the company and not by any 3rd party.	Manual	Pass	This requirement is fulfilled by the nature of the app itself: we user only internal databases and do not expose user location data to anyone else.
3.1: Application is itself is MVP (Minimum viable product), with minimum functionality to fulfill the business goal.	Manual	Pass	Nothing useless was implemented for the app and all functional and non-functional requirements were fulfilled. User experience and user interface could be improved though.
3.2: Application should be developed as quickly as possible with the minimum usage of human resources.	Manual	Pass	This requirement is hard to test: I have performed substantial analysis on frameworks and platforms – and picked most popular, productive (based on other people feedback) and suitable for business goals. Possibly there could be other better options for rapid application development (of which I am not aware for instance).
3.3: There should be large pool of software developers who qualify to enhance the given MVP.	Manual	Pass	JavaScript, NodeJS and TypeScript are one of the most popular and fast evolving technologies nowadays. Amount of developers for those technologies is substantial and

			increasing.
4.1: Each interaction with application user interface should not take longer than 5 seconds	E2E Test	Pass	This particular test was tested with unit and end-to-end tests. Yet no performance problems were revealed.

Table 17 Non-functional requirements testing results

6.3 Testing results: conclusion

During the testing phase, no significant defects were found: the application successfully complies with given functional and non-functional requirements. However, not all parts of the application were fully automated with unit, integration or end-to-end tests.

Namely, the currently reported code coverage covers about 89% lines of code.

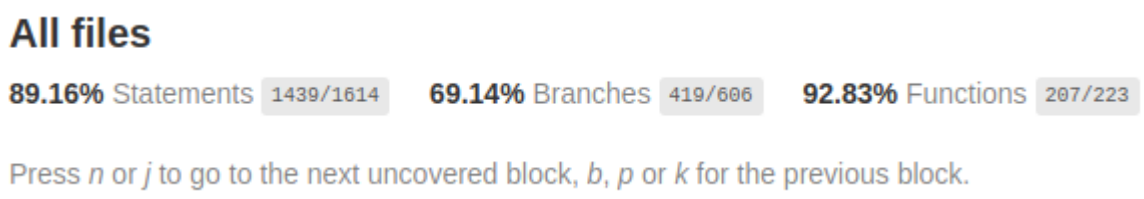

Figure 41 Part of code coverage report

Increasing the code coverage and enhancing the quality of tests could become a goal for further improvement.

The application testing also revealed that end-to-end tests provided most value in our particular case – due to the fact that application interacted a lot with a browser, browser sensor APIs, in-memory database PouchDB, and backend. The integration and unit tests were not used that much – due to the fact that it was much easier to cover the whole application with end-to-end tests than to mock every service. Most of the services performed only more or less complex interaction with databases (Postgres, CouchDB) and did not compute anything particular.

However, testing brought to light the fact that it is hard and time-consuming to test different browser versions on different operating systems (iOS and Android) – the amount of testing work is tremendous. Compatibility between platforms and browsers is

also lacking – Firefox and Chrome behavior is different on different operating systems and devices. For instance, iOS handles idle web-applications differently than Android, and because of that, it is far more important to keep the browser page open on iOS than on Android.

Summing up, the MVP testing phase proved that the proposed solution could work as intended on given browsers and operating systems and provide same business value as separate Android and iOS native apps.

7 Conclusion

The created solution could be used as a basis for creating parental, tracking, and ride-hailing applications. The worked out solution could save a lot of development time and money because there is no need to create and support separate Android and iOS applications – in case of a web application approach it is unnecessary. One advantage of such approach is independence from Google or Apple app stores: the published application could not be blocked or taken down by Google or Apple due to violations of Google or Apple store rules.

The current solution was taken and used in the host company: each employee was given a phone and instructions on how to use the app. The testing by the customer proved that the system could be used in a required way. Unfortunately, further fate of a solution remains unclear – the MVP is used, but no further active development is planned, probably to the fact that developing of app proved to be more expensive than expected. The current solution, however, proved that call center actions could be automated – it was reported that tracking of employees in real-time and sending automatic emails on arrival helps and saves time.

The code provided in the appendix is not exactly the same code done for the client. Obligations cannot give me the right to publish source code that I do not own. However, the code provided in the appendix fullfils all the functional requirements of the described tracking system. The code of system provided in the appendix uses the same technologies and databases, but a different UI design and a multi-tenant domain model and does not contain any hardcoded logic.

The code of this platform could be adapted and used as a platform for creating map-based applications:

- Car, bus, bicycle, electric scooter renting and ride-hailing applications
- Parental and tracking applications
- Delivery applications: Wolt, Bolt food

- Job searching apps of any kind: construction and repair works, home arriving hair dresser, home repair works and interior improvement, cleaning services, moving services etc.

The location of source code of application could be found in Appendix 1. To execute the code you need to have Node 8+ and NPM installed. The code is divided into two separate git repositories:

- Frontend repository – contains UI client part, based on React and Ant design UI library. To start the project locally it is sufficient to execute command “npm start” in the root dir of repository.

- Backend repository – server side part, based on NodeJS and FeathersJS. If you do not want to pull dependencies (like Postgres and CouchDB) manually, you can install docker and simply execute command “docker-compose up” to pull and execute the dependencies. To start the project locally in the root dir execute command: “npm start”

Note that both frontend and backend should be running at once to see a final working result.

As a result, the developed solution and ideas could be used as a basis for web map-based offline application architecture and implementation.

8 **Summary**

During this work, an analysis and implementation of a web-based location tracking platform were performed. As a result, a tracking application MVP that could gather sensor data from a mobile device via a browser was developed. The produced application could be used in different browsers such as Firefox or Chrome, working in different devices, such as tablets, smartphones and even on ordinary laptops and desktop workstations. Tracking application supports different operating systems such as iOS and Android.

The produced platform was analyzed and compared with other similar platforms and applications**.** During the analysis phase, it became clear that there is a lack of good enterprise monitoring solutions on the market. Most of the competitors like Spyfone, Spyzie, mSpy do not respect the privacy of the user and behave more like spyware than a monitoring solution suitable for use in the enterprise. Other competitors (such as Google Maps, Traccar, FollowMee) were not fulfilling the business or privacy requirements.

Created application is privacy concerned, does not depend nor on Google or Apple store, is easy to install, update and remove. User can easily stop monitoring process just by closing the page or remove all related monitoring data by sending delete request to admin.

The current thesis, application, and architecture could be used as one of the sources of information or inspiration for creating similar applications.

In overall, all given functional and non-functional requirements were fulfilled. As a result, the tracking solution surpasses it's free and paid analogs in given functional and non-functional requirements. The MVP still needs to be developed to become a proper end-system that could support more functionality and features. This could become the goal for further developments.

9 **References**

[1] S. S. N. C. D. D. R. C. Lisa Rayle, “App-Based, On-Demand Ride Services: Comparing Taxi and Ridesourcing Trips and User Characteristics in San Francisco,” University of California, Berkeley, August 2014. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.662.707&rep=rep1&type=pdf.

[2] “Uber in play store,” [Online]. Available: https://play.google.com/store/apps/details?id=com.ubercab&hl=en.

[3] “Uber on app store,” [Online]. Available: https://itunes.apple.com/ee/app/uber/id368677368?mt=8.

[4] B. L. Andre Charland, “Mobile application development: web vs. native,” [Online]. Available: http://dl.acm.org/citation.cfm?id=1941504.

[5] “PYPL PopularitY of Programming Language,” 18 11 2019. [Online]. Available: http://pypl.github.io/PYPL.html.

[6] “Spyzie hopmepage,” [Online]. Available: https://www.spyzie.com/. [Accessed 4 12 2019].

[7] “SpyFone homepage,” [Online]. Available: https://spyfone.com/. [Accessed 4 12 2019].

[8] “Traccar homepage,” [Online]. Available: https://www.traccar.org/. [Accessed 4 12 2019].

[9] “Traccar github issues,” [Online]. Available: https://github.com/traccar/traccar/issues. [Accessed 4 12 2019].

[10] “Google Maps,” Wikipedia, 4 12 2019. [Online]. Available: https://en.wikipedia.org/wiki/Google_Maps.

[11] “mSpy homepage,” [Online]. Available: https://www.mspy.com/. [Accessed 4 12 2019].

[12] “FollowMee Homepage,” [Online]. Available: https://www.followmee.com/. [Accessed 4 12 2019].

[13] “Survey, Comparison and Evaluation of Cross Platform Mobile Application Development Tools,” [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.849.9061&rep=rep1&type=pdf.

[14] “Evaluating Cross-Platform Development Approaches for Mobile Applications,” Henning Heitkötter, Sebastian Hanschke, Tim A. Majchrzak, [Online]. Available: https://pdfs.semanticscholar.org/b79d/91011b1986745374cc271991d24bd87e1175.pdf.

[15] N. J. S. K. Pushpendra Singh, “Using Mobile Phone Sensors to Detect Driving Behavior,” [Online]. Available: http://dev3.acmdev.org/posters/dev13posters-final4.pdf.

[16] V. N. P. a. R. R. Prashanth Mohan, “TrafficSense: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones,” April 2008. [Online]. Available: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/tr-2008-59.pdf.

[17] G. I. Andrei Popescu, “Geolocation API Specification 2nd Edition,” 8 November 2016. [Online]. Available: https://www.w3.org/TR/geolocation-API/.

[18] D. a. S. W. Group, “DeviceOrientation Event Specification,” 16 April 2019. [Online]. Available: https://www.w3.org/TR/orientation-event/.

[19] A. K. (. Corporation), “Ambient Light Sensor,” 7 March 2019. [Online]. Available: https://www.w3.org/TR/ambient-light/.

[20] R. B. (. C. Anssi Kostiainen (Intel Corporation), “Proximity Sensor,” 5 March 2019. [Online]. Available: https://www.w3.org/TR/proximity/.

[21] https://dvcs.w3.org/hg/dap/raw-file/tip/humidity/Overview.html, “Ambient Humidity Events,” 11 January 2017 January 2017. [Online]. Available: https://dvcs.w3.org/hg/dap/raw-file/tip/humidity/Overview.html.

[22] “Navigator interface,” 28 June 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/Navigator.

[23] M. L. Anssi Kostiainen, “Battery Status API,” 7 July 2016. [Online]. Available: https://www.w3.org/TR/battery-status/.

[24] G. I. Ian Hickson, “Web Workers,” 25 September 2015. [Online]. Available: https://www.w3.org/TR/workers/.

[25] A. J. J. M. Ilya Grigorik, “https://www.w3.org/TR/page-visibility-2/,” 17 October 2017. [Online]. Available: https://www.w3.org/TR/page-visibility-2/.

[26] “Web sockets,” 18 July 2019. [Online]. Available: https://html.spec.whatwg.org/multipage/web-sockets.html#network.

[27] J. B. (. I. Ali Alabbas (Microsoft Corp.), “Indexed Database API 2.0,” 30 January 2018. [Online]. Available: https://www.w3.org/TR/IndexedDB-2/.

[28] A. Wharry, “HTML5 Offline Technologies,” [Online]. Available: http://andreawharry.com/assets/img/portfolio/WharryAndreaFinalPaper.pdf.

[29] “Generic Sensor API,” 7 March 2019. [Online]. Available: https://www.w3.org/TR/generic-sensor/.

[30] [Online]. Available: \http://mobilehtml5.org/.

[31] [Online]. Available: https://pouchdb.com/.

[32] A. I. Wasserman, “Software Engineering Issues for Mobile Application Development,” [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.475.3806&rep=rep1&type=pdf.

[33] A. S. G. S. Luis Corral, “Mobile multiplatform development: An experiment for performance analysis,” [Online]. Available: https://pdf.sciencedirectassets.com/280203/1-s2.0-S1877050912X00048/1-s2.0-S1877050912004516/main.pdf?X-Amz-Security-Token=AgoJb3JpZ2luX2VjEJ3%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJGMEQCIHpmFxZ2SvADFTLu8RCwIT8L88cx%2FYJuNSIceFtwiBp1AiAx38kftjYHvl.

[34] M. S.-S. Dmitry Namiot, “On Physical Web Browser,” [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1603/1603.04031.pdf.

[35] B. W. Kyle Brown, “Implementation Patterns for Microservices Architectures,” [Online]. Available: https://hillside.net/plop/2016/papers/proceedings/papers/brown.pdf.

[36] M. Richards, “Software Architecture Patterns,” [Online]. Available:

http://sddconf.com/brands/sdd/library/Architecture_Patterns.pdf.

[37] “PouchDB Replication,” 2 12 2019. [Online]. Available: https://pouchdb.com/guides/replication.html.

[38] N. N. Thirumalesh Bhat, “Evaluating the efficacy of test-driven development: industrial case studies,” Center for Software Excellence, Redmond, WA, 2006. [Online]. Available: https://dl.acm.org/citation.cfm?id=1159787.

[39] “mSpy Homepage,” [Online]. Available: https://www.mspy.com/. [Accessed 4 12 2019].

Appendix 1 – Platform source code

The source code of a developed platform is available at the address **https://gitlab.com/twiretech/monitoring/monitoring-frontend** (front-end git repository) and **https://gitlab.com/twiretech/monitoring/monitoring-backend** (back-end git repository). Frontend e2e tests are available under monitoring-frontend repository integration folder.