

Insert here your thesis' task.

CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF SOFTWARE ENGINEERING



Master's thesis

GPS pinned messaging application

Mikhailova Daria & Bc.

Supervisor: Ing. Jan Vaclavik

7th May 2016

Acknowledgements

I would like to thank my supervisor Ing. Jan Vaclavik for his assistance and advice while leading this project and for giving me a chance to learn new technologies and bring one of my ideas to life.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 7th May 2016
.....

Czech Technical University in Prague

Faculty of Information Technology

© 2016 Daria Mikhailova. All rights reserved.

This thesis is school work as defined by Copyright Act of the Czech Republic. It has been submitted at Czech Technical University in Prague, Faculty of Information Technology. The thesis is protected by the Copyright Act and its usage without author's permission is prohibited (with exceptions defined by the Copyright Act).

Citation of this thesis

Mikhailova, Daria. *GPS pinned messaging application.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2016.

Abstrakt

V několika větách shrňte obsah a přínos této práce v českém jazyce.

Klíčová slova Replace with comma-separated list of keywords in Czech.

Abstract

The goal of this thesis is to explore a new way of creating mobile applications using Javascript technologies and implement a complete system using Javascript. The system will comprise of mobile application which brings a new way of communicating by leaving messages at GPS locations, web application and a REST API server.

Keywords Replace with comma-separated list of keywords in English.

Contents

Introduction	1
1 Requirements	5
1.1 System functional requirements	5
1.2 System use cases	7
2 Analysis and design	11
2.1 Technology used	11
2.2 System design	19
2.3 Database	20
2.4 Augmented Reality Feasibility study	23
3 Realisation	27
3.1 Server	27
3.2 Mobile Application	39
3.3 Web Application	48
3.4 Testing	49
Conclusion	51
Bibliography	53
A Acronyms	57
B Installation guide	59
B.1 Server	59
B.2 Mobile application	59
B.3 Web application	59
C Web application screens	61

D	Mobile application screens	63
E	Supervisor configuration	65
F	Express server for React routing	67
G	Integration of Wikitude and React Native	69
H	Contents of enclosed CD	77

List of Figures

0.1	Messaging apps have surpassed social networks Source: [1]	1
2.1	Node.js system	12
2.2	Node.js server processing compared with traditional web server processing of requests	13
2.3	Photo marker example Source: [2]	18
2.4	Black and white marker example Source: [3]	19
2.5	Points of interest Source: [4]	20
2.6	Interaction between parts of the system	21
2.7	MongoDB model of GPS pinned messaging application	22
3.1	OAuth2 authentication protocol flow. Source: [5]	32
3.2	Server directory structure	39
3.3	Redux structure. Unidirectional dataflow.	41

Introduction

The rapid usage growth of messaging apps over the past several years can only be described as astronomic. Currently mobile messaging applications are offering so much more than just simple messaging and media exchange. Some of them connect users with brands, bring news and even let you shop through them. Messaging apps's user base is outweighing the most popular social networks.^[1] As you can see on figure 0.1, which compares WhatsApp, Facebook Messenger, WeChat and Viber to Facebook, LinkedIn, Twitter and Instagram. The scale is in billions of users.^[6]

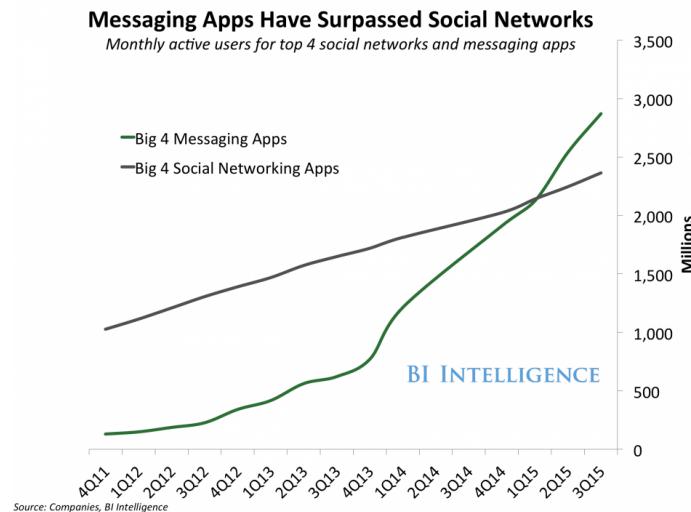


Figure 0.1: Messaging apps have surpassed social networks

Source: [1]

All of the mentioned above makes messaging applications and social networks a vital part of our life. Nevertheless the rapid growth in functionality let's not forget their main aim. The main aim of both types is to connect

INTRODUCTION

people and let them exchange messages in a cheap and easy way. Both messaging application and social network fulfil this goal very well. However there is a certain communication gap which has not been addressed yet.

We use messaging applications and social media to communicate with people we already know. We become highly suspicious when it comes to meeting new people in the case we do not have any friends in common. Let's use Facebook as an example. To be honest Facebook was not created for connecting with strangers and its main purpose is to stay connected with your friends and family. The old superstition that many people online, who are not your acquaintances, are not who they pretend to be is still on. So how do we meet new people online? Facebook offers many interest clubs and event pages where we come in contact with people we do not know, but hardly any meetings or friendships arise from that. Internet also offers old school forums and communities where people talk about their common interests such as movies or music. But in case you just want to meet someone not based on your interest, just a random person the only way to do it is via a dating site. You have to sign up for a dating website or download an app such as Tinder. There are many people on dating sites who are just looking for friends and not long life partners. In the wide circles looking for friends on dating websites is considered slightly strange. If you are extravert enough you can just meet people on the street or in the bus. But what do you do when you are in a foreign country? What do you do when you are just wandering around looking for a company to do some sightseeing? What if you just really liked this museum and want to find so-thinking people without the whole going extremely public about it?

Meet the GPS pinned messaging application. The idea is very simple. Usual messaging applications require the communicating parties to identify themselves with each other, either using their phone numbers or emails. This creates a certain barrier in communication. People are not that interested or afraid of just chatting with strangers online since in order to do so they have to reveal a lot of their personal information to the other party. The GPS pinned messaging application does not require any details from the other party in order to communicate. Imagine you are travelling and you just moved in a hotel. You do not know anyone in the city and would like to have a company for the sightseeing. In this case you just leave a message right on the GPS location of your room and everyone in the hotel would see it, since they would be within a short reach of your GPS location. Anyone could answer your message and probably very quickly you will manage to find yourself a company for the walk in the city.

GPS messaging application allows a user to leave messages pinned to GPS location anywhere. It could be a review message saying how much someone liked a certain gallery or a message warning fellow travellers of scams or robberies happening in the area. University students could exchange messages with the whole lecture hall during the lecture. The only way to read such a

message is to be within short distance of it. You cannot read the message if you are far away. This makes messaging a quest, when you go through the city with new messages appearing on the map. This brings freedom in messaging. It is almost the same as you would go and directly talk to someone. This messaging style leaves out all the unnecessary information which you have on the social networks, it keeps the users identity private and therefore you do not know complete biography of a person before talking to them. And most importantly it allows you to interact with anyone who is within your GPS reach. It is private and public at the same time, limited by location and messages validity only.

CHAPTER 1

Requirements

The main goal is to create a system consisting of a messaging application which will allow users to pin messages to GPS location, web application for uploading 3d objects and viewing statistics and a REST API server to communicate with both of them.

Messaging application will offer two types of messages. Firstly users will post simple text messages with an option of attaching a picture. Secondly users will be able to post a 3D object. Both types of messages are commentable. Each posted message can have a validity from 1 day up to 365 days. Users are only able to read a message once they arrive at its GPS location. The user will be able to browse through his messages and in case someone commented on his message he will be notified of it.

Web application is a complimentary portal which allows to browse through a common 3d gallery and choose objects for messages, upload your own 3d objects and view messaging statistics.

The server will be providing data for both applications and will also serve as an authorization authority.

1.1 System functional requirements

1. Registration

User can register using their email and password using mobile application.

2. Authentication

Users will be authenticated through tokens using OAuth 2 protocol.

3. Leave a message

The user is able to leave a message pinned to his/her GPS location using mobile application. There is an option of adding an image to the message.

1. REQUIREMENTS

4. Leaving a 3D object

The user is able to leave a 3D object pinned to his/her GPS location. The user can choose a 3D object from the gallery of objects or upload their own object following the instructions on the web.

5. Read a message

The user is able to read a message only once he/she is close to its GPS location.

6. Comment message

The user is able to comment any message he stumbled upon.

7. User browses through his messages

The user will be able to browse through a list of the messages he posted and see how many people viewed the message and whether there are any new comments since the last time he viewed the message.

8. View statistics

Users can view statistics of their messages, for instance how many times the message has been viewed and how many comments were left. There will also be general statistics for everyone showing popularity of different countries by the number of messages.

9. View and change user details

The user will be able to update his profile using mobile application. He will be able to upload a profile picture, choose a country, change email or password.

1.1.1 System non functional requirements

1. Performance

There must be short response time for server requests.

2. Resources

The application should not consume too much battery power in both active and idle modes.

3. Recoverability

Should be able to recover fast in case of server crash or any other failures.

4. Data Integrity

The messaging and other data has to be consistent in any situation.

5. Documentation

Extensive and detailed documentation must be provided.

6. Extensibility

The application should be created using technologies that it would be easy to extend and create new features in the future.

7. Scalability

The application needs to be able to cope with a growing amount of requests.

1.2 System use cases

1.2.1 User registers

1. The user opens the application and taps button *Become a member* on the main screen.
2. The user fills in his username, email and password.
3. In the case no validation errors occur the user is redirected to the map.

1.2.2 User leaves a message on his GPS location

1. The user opens the application and logs in.
2. The user views the map, where his current location is marked.
3. The user taps on Add message to create a message.
4. The user chooses *text message* and types in the text. He attaches a picture by pushing add image button and chooses the picture from his photo library. Then the user saves it and the app performs a redirect to the map view again.
5. The user can see that now next to the marker with his current location appears an icon symbolising his message.

1.2.3 User reads messages in the area

1. The user logs in.
2. The user opens the application and views the map.
3. There are icons on the map showing where different messages are. The user can only see messages in the preset radius from his current location.

1. REQUIREMENTS

4. The user walks closer to the chosen message on the map.
5. Once the user is close enough he can view the message and comment on it.

1.2.4 User comments on a message

1. The user logs in.
2. The user opens the application and views the map.
3. The user walks close to the chosen message on the map.
4. The user opens message detail.
5. There is a list of comments under the message and a comment box
6. The user posts a comment on a message and it immediately appears under the post box

1.2.5 User uploads a 3D object and posts it as a message

1. The user opens the web application.
2. The user logs in and chooses upload a 3D image from the menu.
3. There are instructions on uploading the image, which size and which formats are accepted.
4. The user uploads the 3D object, which is added to his object collection.
5. The user clicks on object collection in his profile and sees that his new object has been added.
6. The user open mobile application and performs Use case 1, but chooses to create a *3d message* instead of *text message*. Under the message text box there is an option to view user's 3d gallery.
7. 3d gallery offers an option of viewing the object in AR before adding it to the message.
8. The user chooses a 3d object by viewing its thumbnail and the object is attached to the message.
9. Once the message is saved the user is redirect to the map and can see that a new message has been added to his current location

1.2.6 User checks on updates for his messages

1. The user opens the application and views the map, where his current location is marked.
2. The user opens the menu and chooses My messages.
3. The user sees a list of his messages with numbers of total views. In the case there has been a new comment on the message the user will see a new comment icon next to his message with a number of new comments.
4. The user can see that now next to the marker with his current location appears an icon symbolising his message.

1.2.7 Profile change

1. The user opens the application and views the map, where his current location is marked.
2. The user opens the menu and chooses Profile.
3. The user can see his information on the profile page.
4. The user taps on edit and now can edit his profile information. Each profile edit is confirmed with password. The user can add a profile picture, change email, password and country.
5. Once the profile changes are saved the user is redirected to the profile preview where he can see the changes.

CHAPTER 2

Analysis and design

2.1 Technology used

2.1.1 Node.js

Node.js is open source and cross-platform runtime environment with most of it being written in Javascript. Node.js uses non blocking event-driven I/O to remain flexible and lightweight when faced with intensive real-time application with high data transfer demands.[7] The non blocking part means that some commands, such as reading from a file, are delegated and executed in parallel and they use callbacks to signal completion. For instance in a blocking language such as PHP, the command executes only after previous command has finished running.[8] Node should not be used for heavy computations, since it will eliminate all of its benefits. Node manipulates with one thread. Therefore resource consuming computations can clog the thread and prevent other requests from being processed. It is best used for creating fast, scalable network applications, since its biggest advantage is its ability to handle a massive number of simultaneous connections. Another massive advantage of Node.js is built in support for package management using NPM tool. NPM provides the developer with a set of reusable components which are easily installed and used in the project. It is one of the largest set of open source libraries.[9]

The way Node.js works is demonstrated on figure 2.1. The whole action of Node.js happens in the Event loop. Event loop is a set of events defined in your application which node is waiting for to be fired. Once an application sends a request, an event is fired and is placed in the event queue. Event loop picks up the event and processes it. In the case that the event is blocking such (e.g. image reading) it passes its processing to one of its worker threads from its own worker thread pool. When the worker is finished processing the event, it returns the response using a callback which was passed to it with the task. The result is passed to the application through Node.js bindings [8]

The main advantages of running a node.js server:

2. ANALYSIS AND DESIGN

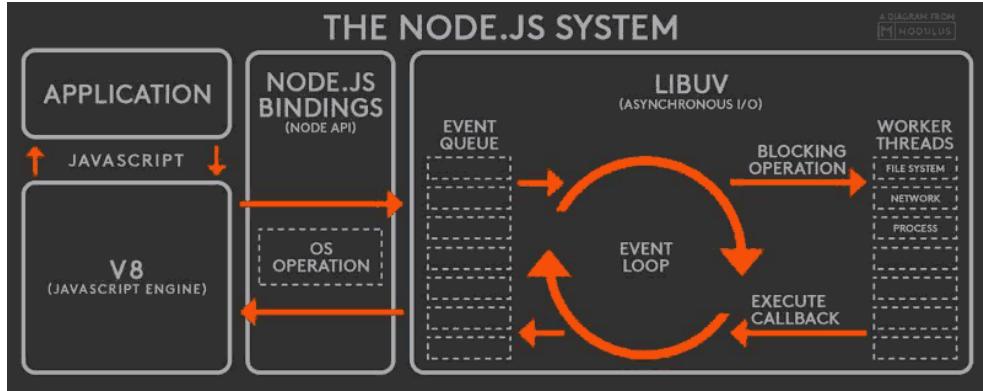


Figure 2.1: Node.js system

- Node.js server can asynchronously process incoming requests. Therefore the response rate is faster.
- Node.js is event-driven, and uses non blocking I/O by utilising callbacks, event loop and event queue. This means that we use browser style concurrency model on the web server.
- Since it is javascript we are able to take advantage of the V8 Javascript Engine. V8 Javascript Engine compiles Js code into native machine code which brings better performance comparing to usual techniques such as interpreting.
- Node js is well suited for real time application that run across various devices. It is perfect for fast delivering of data to many requestors at the same time.

Figure 2.2 shows a comparison of request processing on a node server and a traditional web server (e.g. web server Apache). On a traditional web server for each request a new connection thread is created from a limited thread pool or the server waits for any of the working threads to finish so it can delegate the request to them. Many threads use a lot of RAM and the system eventually slows down or even crashes in the case of a lot of requests. What Node.js does differently is it operates on a single-thread using non blocking I/O calls, which allows it to support a lot of simultaneous connections. Assume that each request-thread will demand 2MB of memory. Let's also assume that the server is running on an 8GB of RAM. This leaves us with a conclusion that when using a traditional web server it is capable of processing about 4000 simultaneous requests. Node.js in contrary is theoretically able to process over 1M simultaneous requests due to its scalability. [7]

2.1. Technology used

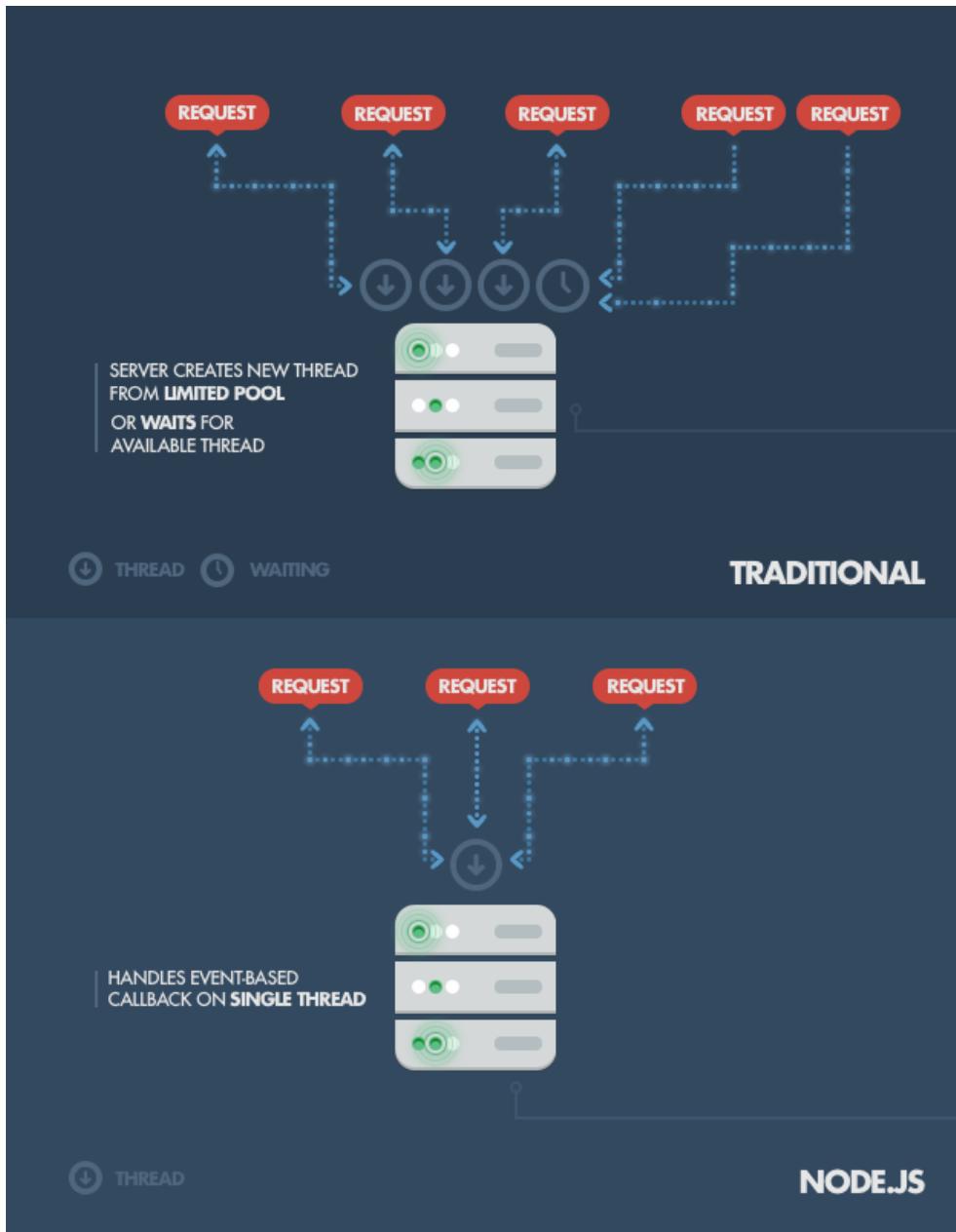


Figure 2.2: Node.js server processing compared with traditional web server processing of requests

2.1.2 React

React is a very flexible view layer written in javascript. It has many advantages over the existing ways of treating front-end, with the main one being the way it is organised. It breaks down the application into components and makes

2. ANALYSIS AND DESIGN

it very easy to make changes without affecting other code. The components, each of them representing a single view (e.g. a button, form), can be reused for other applications since each component represents an independent part of the programme. This also makes it easy to code and read code since looking at one file tells you how the whole part of the system functions. [10]

It uses a declarative programming style. Declarative programming is when we tell the machine what we want to see as the result and let it figure out how it should be done. [11]

React's way of creating application might not seem correct to many developers since it embeds css and html in javascript, which has been considered a bad practice throughout the past decade. Mixing markup with logic is what we have been always avoiding. However there are quite a few issues with CSS which React JS, due to its approach, solves without any hacks. For instance no need to worry about dependencies, dead code elimination, minification, isolations, etc. Moreover this approach allows us to create independent and easy to read components which can be changed at any time without worrying about affecting the rest of the application.

One of the biggest advantages is the virtual DOM. React does not re-render the whole DOM every time you make a change. React creates a diff between the existing DOM and your changes and applies the patch to the existing DOM, changing only a part of it and saving loads of time.

Server rendering is another very important point. Server rendering allows to render the whole page on the server side (e.g. Node.js), which is faster and SEO-friendly. Both virtual DOM and server side rendering make React's performance very high.

Furthermore React has good descriptive warnings which make the debugging process easy. Its declarative approach makes the code predictable and easier to understand. Therefore the learning curve is flat and developers start making changes with confidence faster.

React relies on unidirectional data flow. The data, called props in React, are passed from parent to child. When the props change the component is re-rendered in order to keep up to date with the application data changes. Also every component has its own private data called state. [10][12][13]

2.1.3 ReactNative

2.1.3.1 Why not use Native?

The native mobile development is difficult and cumbersome. Firstly it is difficult to place different components of your app on the screen since you often need to compute exact positions of all of your views. Moreover every time the developer makes a change in the app they need to recompile the whole application, even if it is the smallest change, such as enlarging fontSize. This takes a lot of time from the development process and slows down the whole

development cycle. In the case the developer wants to create a multi platform application, let's say for iOS, Android and Windows, they need to write the same application three times in three completely different languages. Apart from the need to learn specific platform implementation and principles, which are usually not reusable for other platforms, the developer also has to think about memory management, thread concurrency and deployment process. All these issues are important and bring a necessary overhead when developing native applications

Even though development of native applications is a lengthy and difficult process, there are reasons why it is worth it. One of the main reasons why developers go and make native apps is the access to mobile native functions which allow to create a better-feeling experience, such as UI components - date pickers, sliders, etc. This main feature of native development is reproduced in ReactNative as you can read further.[12]

2.1.3.2 Why ReactNative?

It is the JS framework which works with mobile's native functions directly, which is its huge advantage.

React Native runs the code using an interpreter and provides a native bridge to construct and fully interact with platform's native elements. Just like React, which renders divs and spans in the browser React Native renders native higher-level platform-specific components inside of an embedded instance of JavaScriptCore in the application. It also allows to use CSS Flexbox which creates a better User Experience and an easier placement of the elements with no need for computing their position.

Furthermore working with React is faster than with other platform native languages, since you do not have to recompile the whole app any time you make the tiniest change, you just refresh the application and view your changes straight away. Its very handy and it saves a lot of development time. React Native also provides us with the ability of parallelising the work, which increases the speed of the application.

This framework introduces a totally new approach for developing mobile apps and enforces a new principle: 'Learn once, write anywhere'. Facebook provides implementations and supports both IOS and Android versions. Since React Native is a very recent framework it does not fully support native features for both systems. At the moment Facebook has more components and APIs ready for use with IOS than for Android. Android, on the other hand has a lot of open source components which are easily accessible.

ReactNative can be used alongside with the native code. In order to use native code functions the developer has to create a bridge between ReactNative and native and export the functions they want to use in javascript.

The last but not the least, ReactNative uses React, view framework described earlier. [12]

2.1.4 Objective C

React Native is a very young framework and it has not been fully developed, not all the native components have been wrapped with javascript in order to use in React Native applications. Therefore using Objective C while developing react native application is one of the conditions when the application requires a little extra functionality.

2.1.5 MongoDB

MongoDb is a non relational database which stores its data in documents. If well designed it provides a very fast way of retrieving data. Among the main advantages over usual RDBMS belong:

- Easy structure. The developer does not need to create difficult relations, foreign keys and relation tables in the schema, since there is no schema. It is a document based database, which holds collections of documents.[14]
- No complex joins. When retrieving data you simply ask for a certain document and get all the needed information straight away. It takes just one request to the database to get your data. The documents are usually comprised of key-value pairs by which are easy to filter.[14][15]
- Flexible. It usually happens that some entities populate certain fields and some do not and then the developer ends up with a lot of NULL value fields. Also adding/removing a new field to/from the RDBMS gets very complex, depending on the relations and the data which the database is already filled in with. Mongodb provides incredible flexibility here, since you can just add fields in some documents and do not add them in others. Doing so you eliminate null value fields and also get rid of the problem of adding/removing the field without breaking the whole structure. [15]
- Easy conversion. The data in Mongodb is stored in json format, therefore once retrieved it is ready to use, with no need of conversion and mapping between formats and structures.[14]
- Query ability. MongoDb uses its own query language which is similar to SQL and easy to learn to those who are familiar with SQL. [14]

In terms of security MongoDb does not allow sql injection since it does not use Structured Query Language to retrieve data. However it is still vulnerable to NoSQL injections. When querying the document set you can use a string containing a Javascript expression or you can pass a javascript function. Providing a great flexibility it also provides a loophole for attackers. In

the case of passing a function if the user input is not properly validated and escaped the attacker can pass various malicious parameters. Therefore the developers have to make sure they take proper care of the user input. Moreover before version 2.4 mongo had a db variable available in the javascript context, which if misused could provide the attacker with a lot of information about document collections. For more information about this issue please refer to [16] and [17] or have a look at this article [18], which contains working examples of malicious code.

2.1.6 Webpack

Webpack is a module bundler, a tool which packs the whole application together, translates everything needed to pure javascript, html and css. The bigger your app becomes the more modularised it is. It is a good development approach to break up your javascript application into multiple modules. The code becomes cleaner and easier to read, the loading time of your application also decreases since you do not load everything at once. Webpack takes these different modules with dependencies and merges them serving the content understandable by the browsers. [19]

2.1.7 Augmented Reality

One of the options this mobile application provides is leaving 3D objects pinned to GPS. In order to implement this I used technology called Augmented Reality.

Augmented reality is the real world enhanced by the virtual reality. Virtual reality objects are placed in the real world settings and can be viewed with the help of various devices: mobiles, tablets, smart glasses etc.

There are several different types of Augmented reality and it is not within the scope of this thesis to mention and describe all of them. However I would like to mention a few interesting types which I was researching about when choosing which type of Augmented Reality I would use.

- Marker based AR

The marker was created to help the device overcome the difficulty determining orientation of the camera and understanding what environment it was observing. The marker is an easily detectable external sign placed on any surface. Once the camera detects and recognises the marker it can define the correct scale and pose of the camera. This technique is commonly known as marker based tracking. Marker can be anything from a clear black and white image to a colour photo. The main condition is that the marker has to be recognisable and not visually merge with the environment surrounding it. Preferably the marker

2. ANALYSIS AND DESIGN

should be a square since four points are enough for the camera to recognise its orientation precisely. Figure 2.3 and Figure 2.4 are examples of different types of markers. Each marker can have information encoded in it, which would be identified by the device and further processed, for instance commercials might encode video links, which are played when the marker is recognised. Marker based systems are popular due to their fairly easy implementation and a number of accessible frameworks implementing that functionality out of the box. [3]

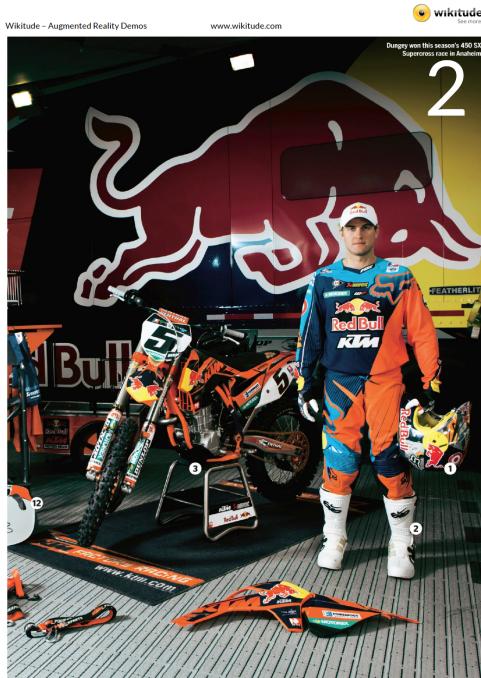


Figure 2.3: Photo marker example

Source: [2]

Marker based AR is implemented in the famous IKEA application which came out in 2014. IKEA made a smart move and made their catalog a marker. The user would place catalog on the floor, point their phone at it and an IKEA piece of furniture would appear.[20]

- Markerless AR

Markerless Augmented Reality as states the name uses no marker which makes its task more difficult. There are different methods of how to implement markerless tracking but I will only mention the following two as they seem of the biggest interest to me. They are simultaneous tracking and mapping (SLAM) and extensive methods or parallel tracking and mapping (PTAM). Both of these method use no prior knowledge of the

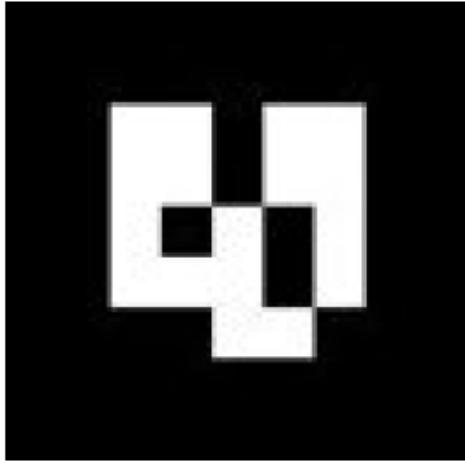


Figure 2.4: Black and white marker example

Source: [3]

environment. Both methods map the environment creating feature maps using various feature detection methods and algorithms. It is not within the scope of this thesis to describe these methods in detail however if interested please refer to [3].

- Location based AR This type of Augmented Reality places objects in real world on the pre-defined GPS coordinates. There are several subtypes for location based Augmented Reality:
 - POI or Points Of Interest is when an application registers a list of points of interest and shows them to the user based on his location. For instance it could be different restaurants or shops in the area. Figure 2.5 shows a sample application with POI AR.
 - Placing of 3D objects. This type is similar to POI but instead of 2D markers on the screen it uses 3D objects which are placed on the given GPS coordinates. Depending on the implementation some object are interactive while others are only for observation.

Augmented Reality can be interactive with user clicking on links and objects to get more information about the scene, product or a building.

2.2 System design

The goal of the thesis is to create a messaging system which will comprise of a mobile application, web application, server and a database. Figure 2.6 shows the proposed system's design. As the figure states the core of the

2. ANALYSIS AND DESIGN



Figure 2.5: Points of interest

Source: [4]

system is a Node.js REST API server. It stores data in a NoSQL document database mongodb. Web application will be implemented in React and will be communicating with server through its API. Mobile application will also communicate with the server through API and will be implemented in React Native. The request load is expected to be high due to the system's purpose - messaging. Therefore Nginx http server will be configured as a load balancer using strategy least connectivity for helping to deal with requests. Least connectivity strategy forwards the incoming request to the server with the smallest number of connections.

2.3 Database

One of the most important parts of any application is the database design. It is essential to understand how your data will be handled, saved and organised before starting to build an application. As was already mentioned in the previous chapter I chose MongoDB for this implementation. MongoDB does not define a schema design principles or design notation which I could use. Therefore I used a simple notation where each square element represents a single document in the collection as the figure 2.7 shows. The arrows are the references, the way where the arrow is pointing the reference is provided.

MongoDb has a different view on data than relational databases. Rela-

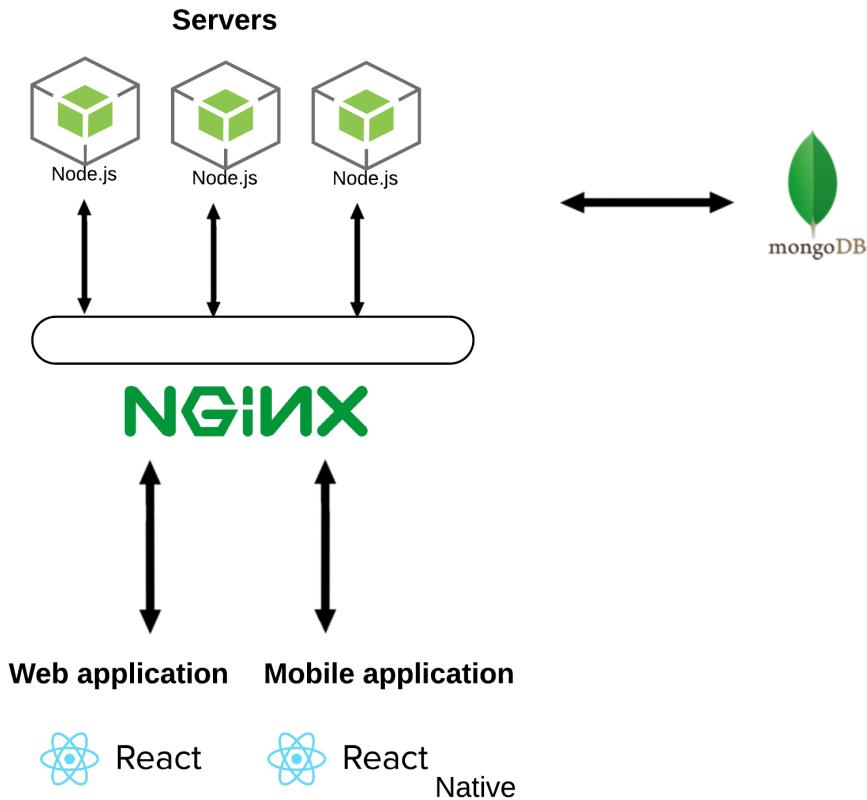


Figure 2.6: Interaction between parts of the system

tional databases are focused on what answers we have when document based databases are more focused on what questions we have. Therefore minor duplications are allowed, because they will save time preventing the database performing joins and looking up information in other tables.

In GPS pinned application the main concern is the messages. We need to know on one request what is the message and all details about it.

- Message

Message contains all the necessary data about the message. The message document contains message text, description, which is a 150 characted substring of text, its location with latitude and longitude and also city and country, which will be populated based on the Google Api response using coordinates. The message also contains object user with user's id and username. The case is that when previewing the message the only information we need to show the user is the username of the person

2. ANALYSIS AND DESIGN

who left the message. This minor duplication saves us time we would have spent looking through the user table in order to find correct user and get his username. MongoDB being a document database allows us saving the image as a base64 string right into the document. Maximum size of the MongoDB documents is 16 MB. We are dealing with mobile phone pictures which are on average between 2 and 5MB so we don't need to worry about space. Just for the information in case of bigger images it is also possible to save them in the collection. MongoDB has a tool which splits big documents in several smaller allowing any size to be safely saved in the collection.

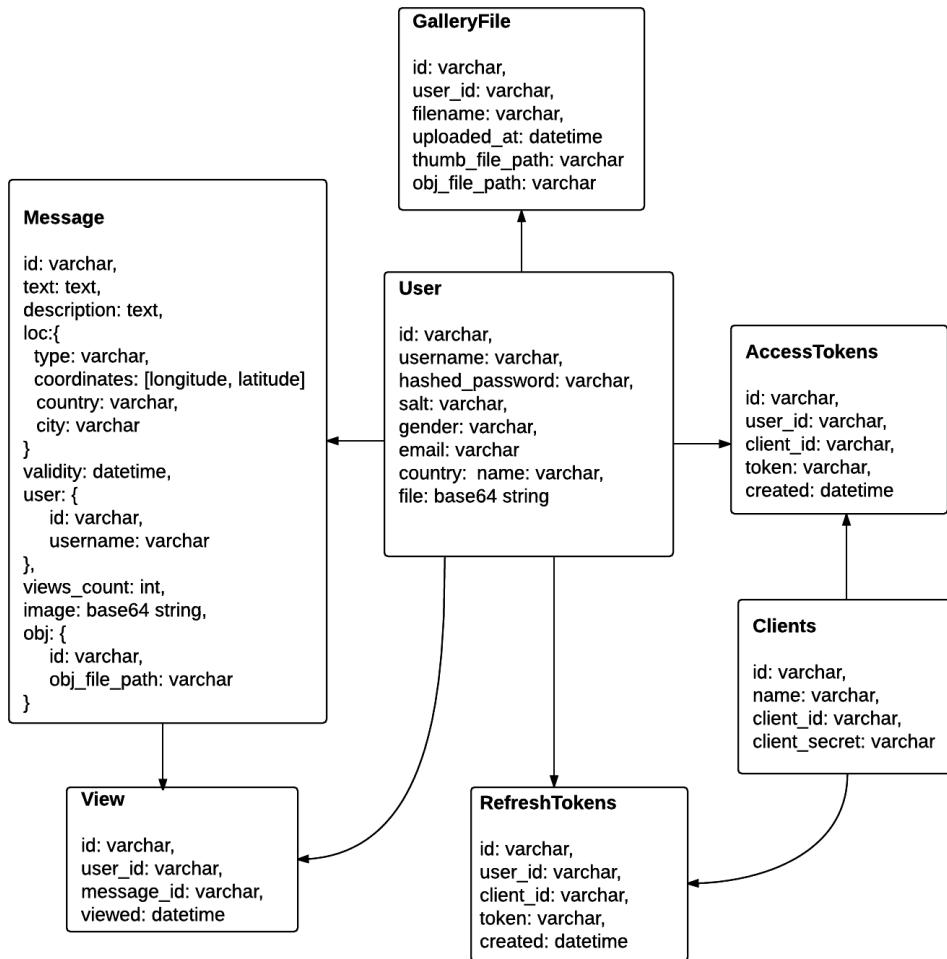


Figure 2.7: MongoDB model of GPS pinned messaging application

Some messages will be 3d obj and in order to save time and server side operations object id and its path will be saved directly in the message

document.

- User

User document contains standardised user data, authentication fields and a profile picture. User document also has a country string saved in it. The country list is saved in a json file and is provided to the user as a dropdown select in mobile application.

- GalleryFile

Every user will have their own object gallery. They can upload their 3d object there according to the instructions given on the web. There will also be a common gallery from which users can choose objects they like and add them to their gallery. User gallery objects will have field user id populated while common gallery objects will have that field set to null. Each object will have a thumbnail and an actual file paths saved.

- View

This collection will have info about when and which user viewed which message. This collection will be used for statistics and when checking for new comments and views of users messages.

- AccessTokens

Access tokens are generated on the server side and are connected with a user and a client.

- RefreshTokens

Refresh tokens are used for regaining an access token after it expired.

- Clients

Clients documents is a list of client application communicating with the API.

2.4 Augmented Reality Feasibility study

After conducting a thorough research concerning various Augmented reality implementations, techniques and frameworks I limited my options to three choices. My goal is to be able to place a 3D object at a given GPS location. Therefore I was looking for a location aware solution. Below are listed different AR solutions I was considering.

- Open source Augmented Reality SDK

There are a few open source AR SDKs with IOS support, implemented in Objective C. The one which seemed to be the most relevant can be

2. ANALYSIS AND DESIGN

found at <https://github.com/promet/PRAugmentedReality>. This is neat small framework which supports even the older iPhones. It supports a type of AR called POI or Points Of Interest. It is well documented and has a video tutorial on how to get started with the framework. There are also well commented code and a sample application.

Advantages:

- Support for Points of Interest therefore Geo location support

Disadvantages:

- No support for 3D object. The developer would have to implement 3D object rendering himself
- Implemented in Objective C, which is not the most developer friendly language
- The last update of the framework was performed 2 years ago. The framework is not supported anymore. There is no community and in case of problems or bugs the developer has to deal with everything himself. There is an option of communicating with the author, but he didn't respond to my email inquiring about the possible support of 3D images, so I would not rely on that.

Conclusion: This option does not seem to be suitable as it requires a deep understanding of Objective C in order to implement rendering and fix possible problems which might arise during development. The focus of this thesis is to explore new way of creating apps such as using javascript wrappers for objective C to create pure native experience.

- Unity 3D - Kudan AR plugin

Kudan is an Augmented Reality SDK. Kudan supports iOS, Android and Unity cross platform game engine. It offers a much better Augmented Reality experience using its Advanced tracking for both marker and markerless augmented realities. For more information about Kudan please see official website at [21]

Advantages:

- SLAM - Simultaneous localisation and mapping is a great technique displaying 3d objects. It has already been shortly described in the previous chapter. The beauty of it is that it maps the environment and puts the object in it so it looks natural. For instance if your camera would be facing a table the object will be put on the surface of the table.

Disadvantages:

- SLAM - even though it is an advantage, it is also a disadvantage since for my purposes I need to have an object placed at a certain location. Unfortunately it is not possible with SLAM since it is not a location based technique. Therefore when using SLAM and turning the camera around, the object will always stay in front of the camera, which is an unwanted result for me.
 - New framework - it is not a severe disadvantage, but usually new frameworks tend to be buggy. It is better to wait some time until the next version is released to start using any new system.
 - Resource changing - the mobile application is supposed to serve the url of the AR object to render. Kudan Unity plugin did not offer an easy and pretty way of doing so. With Kudan serving new AR object would involve physical moving of files and recompiling which would not be possible in my application.
- Wikitude AR Advantages:
 - Javascript API - Wikitude provides a clean API with good documentation. It is a huge advantage since the main technologies I am using in my thesis are all javascript based.
 - Big community and support - The Wikitude AR has a big community and a forum with a lot of different issue to browse through. It has been already developed for several year, which means loads of useful features were implemented and bugs eliminated.
 - iOS integration - My application starting point is objective C project and I was looking for a solution which would be easily integrated in Native - React Native flow. The documentation provides with a good example of how to work with the Wikitude.

Disadvantages:

- Trial version - the only available version to use is a free version, however with full support of all SDK's features, is trial. There is a watermark on the camera screen when using AR.

After careful consideration I decided to choose the Wikitude AR SDK with trial version. It seems to be the best solution, not minding the trial flags on the camera. It provides an easy way of integrating the framework in my application. Javascript API is also a very big advantage.

CHAPTER 3

Realisation

3.1 Server

The server is implemented as a REST API Node.js server using Express framework. Express is a commonly used base for a server. It includes the main components and libraries the developer needs to create a server. It is a minimalist framework that serves as a solid starting point, providing a robust set of features, middleware and methods. It makes creating API quick and easy. Many frameworks created for working with Node.js are based on express. Express is even shipped with a generator which provides you with a basic directory structure. Detailed info about the framework, including documentation and examples can be found at [22].

3.1.1 Endpoints

REST API design is an important part of the realisation process. It helps defining application's functions and capabilities and develops an overall understanding of the workflow.

- Index

GET /

renders API documentation

- Messages

GET /messages?lat=' '&lng=' '

- lat - latitude coordinate
- lng - longitude coordinate

3. REALISATION

retrieves all messages within the preset radius of 50 meters from the given location specified by latitude and longitude

POST /messages

saves a message to the database

GET /messages/:id

– **:id** - message id, string

returns a message with the given id

GET /messages/user/:id?page=' '&limit=' '

– **:id** - user id, string

– **page**

– **limit**

returns user's messages, supports pagination

GET /messages/:id/comments?page=' '&limit=' '

– **:id** - message id, string

– **page**

– **limit**

returns message's comments, supports pagination

POST /messages/:id/comments

– **:id** - message id, string

saves a comment for a message specified by id

- Files

POST /files/form

special endpoint for saving the web form for uploading gallery images

GET /files/gallery?page=' '&limit='

- **page**
- **limit**

returns common gallery files, supports pagination

GET /files/gallery/user/:id?page=' '&limit='

- **:id** - user id, string
- **page**
- **limit**

returns gallery for the user, specified by id, supports pagination

- Users

GET /users/me

returns information about the currently authenticated user

GET /users/:id

- **:id** - user id, string

returns information about the user specified by id

POST /users

creates a new user

PUT /users/:id

- **:id** - user id, string

updates information about the user specified by id

3. REALISATION

- OAuth

POST /oauth/token

Request body content-type is `application/json` and it must contain

- `client_id`
- `client_secret`
- `username`
- `password`

If `username` and `password` are correct the endpoint returns a token for authenticated communication with the server.

All of the above listed endpoints are only accessed with a valid token with exception of the index which provides documentation.

3.1.2 Authentication

One of the functional requirements for the server is to provide authentication for users. The authentication will be implemented using OAuth 2 protocol. It fits well our needs because we will be communicating with our server through API from two different applications. Figure 3.1 demonstrates OAuth 2 authentication flow. There are 3 main actors in the authentication process.

- User

The user owns a resource which an application wants to access. In the case of authentication the resource is user information, their credentials and personal information. The 3rd party needs to access it in order to identify and authenticate the user.

- Service API

Service API is an Authorisation and a Resource Server. It hosts user protected information and verifies the identity of the user so it can provide the client application with an access and refresh tokens to communicate with the server.

- Client

The client is an application which wants to access user information and other resources which user allows it to use. Before the application is able to access any of that information the user has to authorise it's activity.

The authentication flow is the following:

1. The client application sends the user an authorisation request asking them to allow permission to access their information. It usually appears as a new window, where the application shows the user a list of information it wants to access.
2. In the case that user agrees he provides the client application with an authorisation grant, which is used for further verification of user's identity
3. The Client application uses obtained authorisation grant to form another request to the Authorisation/Resources server, which contains it's client id, client secret, authorisation grant type and the details of authorisation grant sent by user.
4. If the server identifies the client application and the user it issues a response which usually contains an access token, a refresh token and a token expiration date. Authorisation process is complete at this stage. Both client and user were verified by the server.
5. Once application has the token it can issue requests to server to access protected resources which belong to the user.
6. If the client application sends the correct token it is granted access to protected resource. In the case the token has expired the client can always use refresh token in order to receive a new access token.

There are several different authorisation grants that are supported by OAuth 2. It is not within the scope of this thesis to discuss all of them however if interested please refer to [5]. The Password authorisation grant was chosen for authentication with the server. It requires the user to provide the client with their username and password. This grant should only be used in the case when the application is trusted by the server such as our case, when both applications and the server are created as one system.

In order for this whole process to work the client application has to be previously registered with the Authorisation/Resource server. After the registration it will receive a client id and a client secret or so called client credentials which are used for authenticating with the server.

Example of the request with authorisation grant which is described in the step 4 is showed below.

POST <https://oauth.example.com/api/oauth/token>

Request body:

```
1 {  
2     "grant_type": "password",  
3     "client_id": "public_client_id",  
4     "client_secret": "very_secret_client_secret",
```

3. REALISATION

```
5     "username": "mark",
6     "password": "secretpassword"
7 }
```

Abstract Protocol Flow

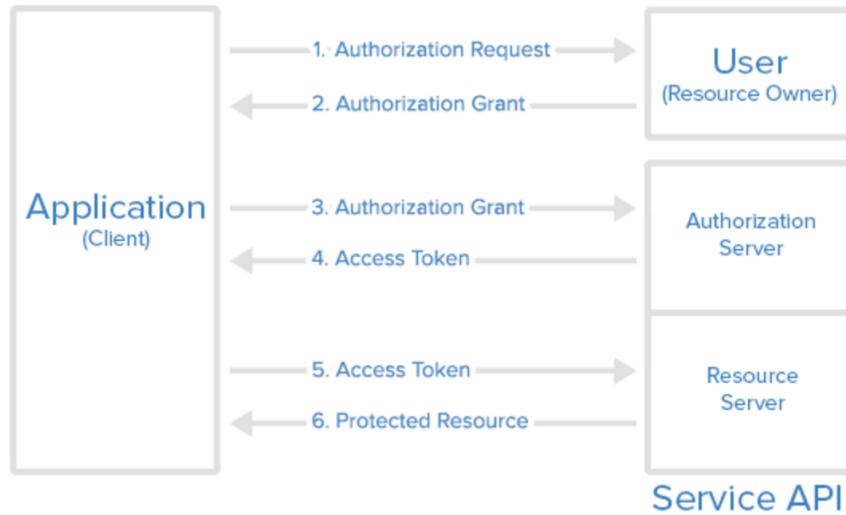


Figure 3.1: OAuth2 authentication protocol flow.

Source: [5]

3.1.3 Files processing

The server is working with files attached to the messages, 3d objects and their thumbnails.

Web application will have a form which will be used for uploading 3d objects and their thumbnails to the server. The application will be sending a request with content type `multipart/form-data` which needs to be parsed in order to retrieve information and files from it. Node.js offers a number of handy middleware which parses `multipart/form-data` and works with files. After a thorough research I chose `node-multiparty` available at [23]. This library parses the form data and makes files attached accessible.

Thumbnails for 3d objects have to be resized to smaller size not to waste space on the server. For resizing a `node-imagemagick` library available at [24] was chosen, which uses original `ImageMagick` library available at [25].

Message attachment files which are not objects are resized on the side of the mobile application. Therefore no further server side processing is needed.

3.1.4 Distance and radius retrieval

One of the main functions of the server will be returning messages within a given radius of user's current GPS location. The calculation of the distance from the user's GPS location will be done using sphere geometry.

In the case of distance calculation there are two ways one can choose - either using Euclidean geometry or spherical geometry. Euclidean geometry is also known as the 'plane geometry'. It is a type of geometry where the internal angles of triangles add up to 180 degrees, parallel lines do not ever cross and everything is flat. Spherical geometry on the contrary does not take account of flat world and works with 3d. Since the earth is shaped in a form of a sphere it would be better to use spherical geometry for distance calculations, since they will be more precise.

The messages endpoint returns messages within the preset radius from given GPS coordinates. Retrieval of messages needed to be fast and effective with the expected growing message collection. Simple search with comparing distances between given GPS and message's location was not suitable.

MongoDb has a powerful system of geo queries. It supports both 2d and sphere geo queries. To use these queries document data have to be saved in GeoJSON format. Documents in this format consist of an array of coordinates (longitude, latitude) and a type. There is a vast selection of types such as for instance "Polygon" or "Geometry Collection". My geo location data are representing a single point therefore I chose type "Point". The example of data in preferred GeoJSON format is shown below.

```

1 "loc" : {
2     "type" : "Point",
3     "coordinates" : [
4         14.422061,
5         50.087657
6     ]
7 }
```

The following is an example of geo query. Another optimisation of search performance is creation of a 2dsphere index on location field.

```

1 loc: {
2     $geoWithin: {
3         $centerSphere: [
4             [
5                 longitude,
6                 latitude
7             ],
8             radius
9         ]
10    }
11 }
```

3. REALISATION

This is a simplified example of mongoose message model with index definition. The actual example of the model is more complicated and can be found in the server implementation in folder `models` under name `message.js`.

Firstly you include `mongoose` library on line 1 and define Schema on line 2. Specific schema definition follows on lines 4 to 9. The line 11 defines index `2dsphere` over `loc` field of the model. Lines 12 and 13 register and export the model.

```
1 var mongoose = require('mongoose');
2 var Schema = mongoose.Schema;
3
4 var messageSchema = new Schema({
5     loc: {
6         coordinates: {type: [Number], required: true},
7         type: {type: String, required: true}
8     }
9 });
10 messageSchema.index({loc : '2dsphere'});
11 var Message = mongoose.model('Message', messageSchema);
12 module.exports = Message;
```

3.1.5 Load balancer

Among the non functional requirements there are Scalability and Performance. In other words the server has to be able to cope with growing requests load and it should be easily scalable if needed. Using load balancer fulfils both of these requirements. Load balancer will forward requests to the chosen server instance in case others are too busy to process new requests. Scalability of the server can be achieved with running more instances and configuring load balancer to register these new server instances.

Load balancer I used is called `Nginx`. It is an http server but is also a reverse proxy server and used for load balancing. Load balancing strategy used on the server is *least connectivity*. *Least connectivity* strategy means that the server with the smallest number of connections will receive new request. [26] Another advantage of having nginx taking care of incoming requests is that it can also take care of serving static content instead of the actual server.

The main part of the Nginx configuration file with registered load balancer is showed below. It is preferable to use `ssl` connection for running the server. The example contains configuration for connecting over `ssl` and without. Lines 3 to 8 of the configuration file define the upstream, the servers which will be processing incoming requests. The load balancing strategy is defined on line 4. Lines 13 and 14 indicate path to log files. The static content serving is configured on line 16. Line 20 defines the proxy pass, which name matches name of the upstream. This configuration allows all requests coming to `http://127.0.0.1:8080` be forwarded to one of the server from upstream according to *least connectivity* strategy. The configuration on lines 27 to 34 does the same but for the secure connection via `https://`.

```

1
2 http {
3     upstream air_servers {
4         least_conn;                      # Least Connections strategy
5         server 127.0.0.1:3000;          # NodeJS AirServer 1
6         server 127.0.0.1:3001;          # NodeJS AirServer 2
7         server 127.0.0.1:3002;          # NodeJS AirServer 3
8     }
9
10    server {
11        listen      8080;
12
13        access_log /var/log/nginx/access.log;
14        error_log  /var/log/nginx/error.log error;
15
16        location ~ ^/(images/|img/|javascript/|js/|css/|
17                      stylesheets/|flash/|media/|static/|robots.txt|humans.
18                      txt|favicon.ico) {
19            root /path/to/static/files;
20        }
21
22        location / {
23            proxy_pass http://air_servers;
24        }
25
26    # HTTPS server
27
28    server {
29        listen 443;
30
31        ssl on;
32        ssl_certificate      /etc/ssl/example.com/example.com.crt;
33        ssl_certificate_key  /etc/ssl/example.com/example.com.key;
34
35        # plus the same configuration as above
36    }
37 }
```

3.1.6 Queue

Priority job queue was used in this project to solve the following problems.

Firstly when creating a new message the user has to set message's validity in days, from 1 up to 365 days. When a message is created it is valid immediately and becomes invalid after the validity time expires. I was looking for a way to set the valid flag to false when the message expires. Job queue seemed like the perfect solution.

Secondly I needed to update message's location details with city and country for the statistics endpoints. This involved calling The Google Maps Geocoding Api [27], waiting for response, parsing the response and updating the document in the database. This slightly lengthy process could not take part

3. REALISATION

while saving the message as it would affect user experience negatively and the application could seem slow. Furthermore in the case of potential problems which could arise from calling 3d party API the whole message saving process could be sabotaged.

3.1.6.1 Kue

Kue is a very simple priority job queue backed up by Redis and created specifically for node.js .

”Redis is an open source (BSD licensed), in-memory data structure store, used as database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.” The description of Redis is taken from its official website found at [28]

The complete documentation for kue is available at [29]

3.1.6.2 Configuration

Kue as every job queue needs to have consumers and producers created. The producer part of the code was put in the processing of message POST request. Here is the example of the producer code.

```
1 function addToValidQueue(queue, message){  
2     var job = queue.create('valid-queue', {  
3         title: 'unset valid',  
4         message_id: message.id,  
5     })  
6     .delay(message.validity)  
7     .save(function (err) {  
8         if (err) {  
9             log(err);  
10        }  
11    });  
12}
```

I created a `queue.helper.js` which took care of putting jobs in the queues. There are two queues for two types of jobs: 'valid-queue' and 'geo-queue'. Valid flags jobs had to be delayed so they had set delayed time period calculated from the number of valid days input by the user. The delaying of a job is part of the kue and is implemented by using `delay()` function on line 6 in the example above. I also created two consumers - `geoConsumer.js` and `validConsumer.js`. The example of geo consumer is showed below. The `geoCode` function in the example on line 5 further processes received job data.

```
1 var kue = require('kue');  
2 var queue = kue.createQueue();
```

```

3
4 queue.process('geo-queue', function (job, done) {
5   geoCode(job.data, done);
6 });

```

3.1.7 Supervisord

Nodejs server needs supervision and fast recovery in the case it fails. For this reason I needed a tool which would keep my server running and restarts it in the case of fatality.

Supervisor is client/server system that allows easy monitoring and manipulations with UNIX processes. It has a very detailed and well readable documentation available at [30]. Once started it runs as a daemon process. It also has a command line utility for easy manipulation with processes. They can be restarted, stopped or started from the command line. To specify the programs one needs to run the configuration file is created. Below is showed an example of configuring a server to run. The full configuration file can be found in Appendix E

```

1 [program:airServer]
2 directory=/server/directory
3 command=npm start
4 autostart=true
5 stdout_logfile=/logs/server_out.log
6 stderr_logfile=/logs/supervisor/server_err.log

```

Queue consumers were also added to the supervisor configuration.

Supervisor provides standard output and error logging, which eliminates the need to setup a logging flow yourself.

3.1.8 Libraries used

I have used the npm package manager for taking care of the dependencies in the project. Among all the libraries I have used there are a few I would like to mention.

- *Passport*

It is a middleware used for authenticating requests. It uses so called strategies to define different authenticating techniques. There is around 300 strategies including authenticating through social networks and enterprise generation and more. [31]

- *Async*

Async is a powerful library providing function for working with asynchronous Javascript. Working with asynchronous functionality get tedious sometimes and this library helps to make it easy. You only need to

3. REALISATION

provide you asynchronous function with a callback which will symbolise its completion.[32]

- **Babel**

I needed a compiler to transform some of the Javascript syntax that I was using, for instance JSX and ES2015. Babel supports latest versions of Javascript using syntax transformers. This lets developer use new features before the browser support comes out. [33]

- **Mongoose unique validator**

This small library is extremely helpful. It is connected to as a middleware to a model in which unique indexes are defined. Mongoose unique validator does a great job in returning readable and user friendly validation messages. MongoDb unfortunately fails drastically in this. Its validation messages have to go through some serious parsing and analysing before being used. [34]

3.1.9 Structure

The directory structure of the server is shown on figure 3.2. The base for this structure was generated by the Express directory generator.

The main logic happens in `app.js`, which defines the whole server, includes important dependencies and connects routes. The `route` folder describes different endpoint operations. The endpoints in `route` folder use helpers from `util`.

The `models` folder consists of `mongoose` models. `Mongoose` is an ORM wrapper for talking to MongoDB. MongoDB is very flexible and does not enforce the developer to have a database scheme defined however it is easier and better to have at least defined models for different collections. Mongo allows changing 'database scheme' on the fly, by adding new keys without any overhead. This advantage might serve as a weak point for an attacker in the case models are not defined, which could allow unwanted and unexpected malicious values get in collections.

The `views` folder consists only of one view because our server serves strictly as a REST API server. The only template shows server's API documentation on the index page.

The `controllers` folder consists of two authorization controllers. The `auth.controller.js` defines authorization strategies of library `passport` (for more info please fsee section Libraries user) such as *BasicStrategy*, *ClientPassword Strategy* and *BearerStrategy*. The `oauth2.controller.js` takes care of token generation, verification and issuing new token on refresh token.

```

bin
├── config ..... server configs
├── controllers ..... authorization controllers
├── data ..... mongodb documents directory
├── docs ..... api documentation api blueprint
├── kue ..... kue consumers
└── logs
  └── models ..... mongoose models
  └── public
    └── routes ..... the thesis text directory
  └── tests
  └── util ..... helpers
  └── views
    └── index.jade ..... server index file showing api docs
  └── app.js ..... main server file
  └── package.json ..... npm packages
└── README.md

```

Figure 3.2: Server directory structure

3.2 Mobile Application

3.2.1 Redux

Redux is a predictable state container for Javascript applications. It is a small library which helps control and structure application's state hierarchy. Redux was inspired by Flux, which is an architecture Facebook uses for their client side applications. It is not within the scope of this thesis to explore Flux but if interested please refer to [35].

Redux's biggest advantage is the way it handles the data. It has a unidirectional data flow, which provides the developer with a clear and easy way to develop and debug applications.

The structure of the application using redux is showed on figure 3.3. The following is the description of each actor of Redux flow.

- Presentational and Container Components

Presentational components are components which do not hold any logic. Everything they need for functioning they receive passed in their props from the Container Components. In the case presentational component needs to dispatch an action or perform any other logic it uses callbacks that are passed to it in its properties during initialization.

Container Components are the ones that dispatch events when presentational components receive user interaction. They also take care of nav-

3. REALISATION

igation and other application logic. Sometimes they are referred to as smart components and presentational components as dumb components.

- Action makers

Action makers return action objects for the actions dispatched in Container Components.

- Store

Store holds the application state. The application state changes with different actions and throughout the whole run of the application it is held by the store. Application state can be defined as a set of objects, collections of objects, cached data from the server, there could be booleans, integer or string values which are requested by the application. For instance the state can hold an authentication token value if the user was successfully logged in the system.

- Reducers

Reducers are pure functions. Pure functions are functions which do not modify their arguments. Reducers accept the current state of the application and an action which was dispatched as arguments. They return a new state of the application, by returning a completely new object. It can be the clone of the current state but it can never be a modified current state. Reducers are basically state machines which tell which state follows after the state you are in if you dispatch the given action.

If you need to perform some preprocessing of data before it reaches the reducer you can use middleware, which will be registered in the store. The middleware part could be placed in between action dispatched and reducer decision about changing the state.

3.2.2 Redux-saga middleware

The above mentioned Redux flow does not specify where 3d party API calls can be taken care of. It doesn't make much sense to place API calls in container components because then the action dispatching structure will not be utilised. Redux has different middlewares written to take care of that. I chose `redux-sagas` middleware.

Redux-saga is a middleware for taking care of asynchronous actions such as calling 3d party API. Sagas are invoked once at the start of the application and run in the background. Each saga is registered in the store for watching or waiting for a certain action. Once the action is fired saga notices it and performs further logic based on the action. Sagas are created using generator functions. For more information about generator functions please refer to

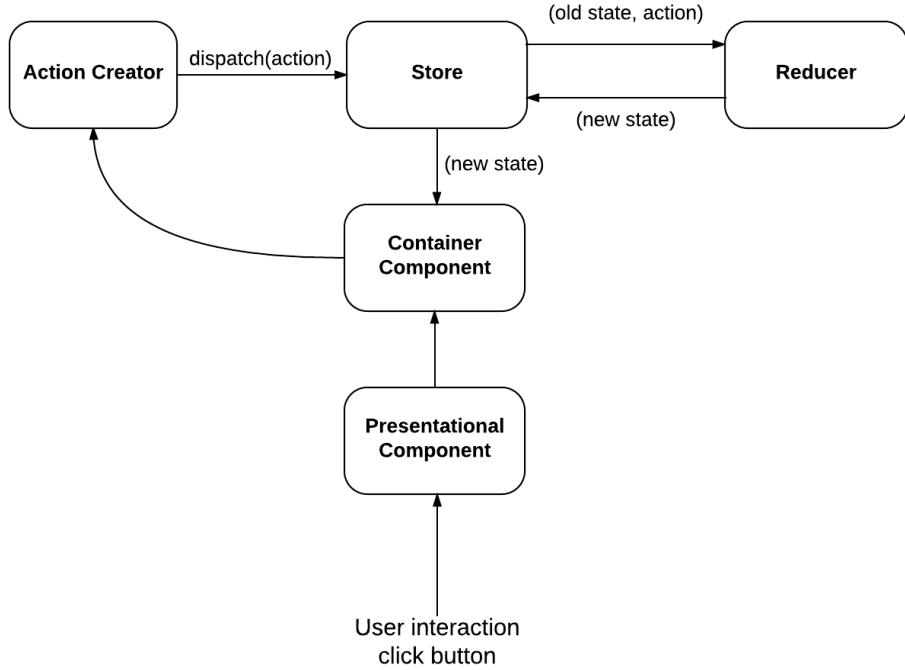


Figure 3.3: Redux structure. Unidirectional dataflow.

[36] chapter External Resources. The generator nature of redux sagas allows writing code in a simple synchronous way. That is a big advantage especially when it comes to API calls and is also easier than creating Promises or using callbacks. Using redux-saga allows to have application logic in two places - reducers which decide on state changes and sagas which take care of data retrieval. [36]

The redux-saga middleware and any other middleware goes in between action creator and the store from the figure 3.3. Sagas basically catch the action before it reaches the reducers, make some changes and dispatch another action which if not used by any other sagas reaches the reducer.

3.2.3 Redux flow example

Here, I would like to give a simple demonstration of one Redux round for posting a message. The examples are purely for demonstration of the Redux principle. They are inspired by the mobile application implementation but simplified in order not to confuse with extra details. I will describe each snippet and then will summarised the whole process in the end. The following setup is for the application which would serve only one purpose to create

3. REALISATION

messages.

The following is a store registration. `Redux` is imported on line 1, followed by `redux-saga` middleware import. Root reducer is the state changer logic, its example is below. On line 4 we import a watcher from sagas files. Lines 8 to 16 initialise the store.

```
1 import { createStore, applyMiddleware } from 'redux'
2 import createSagaMiddleware from 'redux-saga'
3 import rootReducer from '../reducers/index.reducer'
4 import { watchMessagePost } from '../sagas/sagas'
5
6 const sagaMiddleware = createSagaMiddleware(watchMessagePost);
7
8 export default function configureStore(initialState) {
9     return createStore(
10         rootReducer,
11         initialState,
12         applyMiddleware(
13             sagaMiddleware
14         )
15     )
16 }
```

`Index.js` - the entry point of the application. We create the store that was registered in the snippet above here on line 6. The `Provider` import on line 2 enables our application to have access to the store from anywhere. This is very important, since the store holds the application state. You can see that on line 10 the store is passed as the property to the `Provider` which wraps our `CreateContainer`.

```
1 import React, {AppRegistry, Component} from 'react-native';
2 import { Provider } from 'react-redux'
3 import configureStore from './app/store/configureStore'
4 import CreateContainer from './app/containers/create.container'
5
6 const store = configureStore();
7 class AirMsg extends Component {
8     render() {
9         return (
10             <Provider store={store}>
11                 <CreateContainer />
12             </Provider>
13         );
14     }
15 }
16
17 AppRegistry.registerComponent('AirMsg', () => AirMsg);
```

`messages.actions.js` - defines actions and action creators. Each action creator returns a new object. The advised structure of the returned object is as on the following example. It must contain properties type and payload. Each action has its own action creator.

```

1  /** Actions */
2 export const MESSAGE_POST = 'MESSAGE_POST';
3 export const MESSAGE_POST_SUCCESS = 'MESSAGE_POST_SUCCESS';
4 export const MESSAGE_POST_FAILURE = 'MESSAGE_POST_FAILURE';
5
6  /** Action creators */
7 export function postMessage(data) {
8     return {
9         type: MESSAGE_POST,
10        payload: {
11            message: data
12        }
13    };
14}
15
16 export function postMessageSuccess(data) {
17    return {
18        type: MESSAGE_POST_SUCCESS,
19        payload: {
20            new_message: data
21        }
22    };
23}
24
25 export function postMessageFailure(error, action) {
26    return {
27        type: MESSAGE_POST_FAILURE,
28        payload: {
29            error: error
30        }
31    };
32}

```

`index.reducer.js` can be implemented as a simple switch. It receives state and action in its parameters. The default part of the switch should always return the current state. The initial state is an empty array. Reducer is a place where you could insert some logic which would process action data.

```

1 import {
2     MESSAGE_POST_SUCCESS,
3     MESSAGE_POST_FAILURE
4 } from "../actions/messages.actions";
5
6 const index = (state = [], action) => {
7     switch (action.type) {
8         case MESSAGE_POST_SUCCESS:
9             // place for some processing logic
10            return {
11                new_message: action.payload
12            };
13        case MESSAGE_POST_FAILURE:
14            return action.payload;
15        default:
16            return state;

```

3. REALISATION

```
17     }
18 };
19
20 export default index;
```

Part of `sagas.js`. The watcher on line 19 to 21 listens to `MESSAGE_POST` action. It will react on every action of this type fired. Once the action is fired the watcher will call `postMessage` function on line 9. The function will perform the api call on line 11. Thank to the generator nature of sagas we can write code synchronously. Then line 12 dispatches a success action in the case no errors were thrown during the api call. Otherwise the errors are caught and the failure action is dispatched on line 14.

```
1 import {takeEvery} from "redux-saga";
2 import {put, call} from "redux-saga/effects";
3 import {MESSAGE_POST} from "../actions/messages.actions";
4
5 var api = require('../api');
6 var messageActions = require('../actions/messages.actions');
7
8 /** workers */
9 function* postMessage(data) {
10     try {
11         const resp = yield call(api.post, data.payload);
12         yield put(messageActions.postMessageSuccess(resp));
13     } catch (error) {
14         yield put(messageActions.postMessageFailure(error));
15     }
16 }
17
18 /** watchers */
19 export function* watchMessagePost() {
20     yield* takeEvery(MESSAGE_POST, postMessage);
21 }
```

`create.container.js` is a container which renders a `createComponent` on line 10. It is connected to the store and has full access to the state. The function which maps the state to container properties is on line 19. Another function mapping functions that dispatch actions to the store is on line 23. And finally on line 29 both of these functions are connected with our container. On line 25 I map `postMessage` action from `messages.actions.js` to the container properties. Once I call that function `POST_MESSAGE` action will be dispatched to the store. On line 13 this function is passed as a property to the Component, keeping the logic on the side of containers and providing component with a callback to react on user interaction.

```
1 var React = require('react-native');
2 import {connect} from "react-redux";
3 import {bindActionCreators} from "redux";
4 import * as messageActions from "../actions/messages.actions";
5
6 var CreateComponent = require('../createComponent');
```

```

7
8 class CreateMsgContainer extends React.Component {
9
10    render() {
11        return (
12            <CreateComponent
13                postMessage={(m) => this.props.postMessage(m)}
14            />
15        );
16    }
17}
18
19 const mapStateToProps = (store) => {
20    return {
21        message: store.new_message
22    };
23};
24
25 const mapDispatchToProps = (dispatch) => {
26    return {
27        postM: bindActionCreators(messageActions.postMessage,
28            dispatch)
29    };
30};
31 CreateMsgContainer = connect(mapStateToProps, mapDispatchToProps)
32 (CreateMsgContainer);
33 module.exports = CreateMsgContainer;

```

There is no example of component since it is just a simple form with a button, the button would have onclick event set to the function passed to it in properties.

Now I would like to sum up all of the above snippets in order to form a whole picture. The application starts and `index.js` is called. It initializes the store with initial state null and creates the application, passes the store to the application through properties and renders the `create.container.js`. Container render function renders—`createComponent`—which has a form for user interaction. User submits a message. The `onClick` event fires (it was passed from the container through properties). The event fires `POST_MESSAGE` action. The saga watcher is waiting for that action. It grabs the action and passes it to its worker. The worker calls api to save the message. If everything goes ok and the returned response status is 200 the `POST_MESSAGE_SUCCESS` action is fired. The reducer receives the action and changes the state according to implementation. By changing the state the reducer forces re-render of container and component. Container has state `store.new_message` mapped to its properties. Now once the whole cycle is finished the container can do something with the received data.

3. REALISATION

3.2.4 Augmented Reality

As I have previously stated in chapter Analysis and design section Augmented Reality Feasibility study I chose Wikitude SDK for implementation of Augmented Reality in the mobile application.

Wikitude SDK has a plugin for IOS development with Javascript API which I decided to use. There is no direct support for React Native, but the plugin provides a developer with Wikitude framework and there is a set of instructions how to integrate AR in the native iOS application on their website (for reference see official setup guide [2]).

3.2.4.1 Connecting React Native and Objective C

React Native is a set of javascript functions wrapped around native code. Therefore you can make your application communicate with the native code using Native Modules or Native UI Components. This is particularly useful when you want to implement part of your application's functionality natively.

There are two macros which one can use: `RCT_METHOD_EXPORT(*method*)` and `RCT_MODULE_EXPORT()`. Further is described the one that I used.

- `RCT_MODULE_EXPORT()` This macro says that the given class is exported to React Native and can be referenced by its name. It is considered to be good programming style to separate your files into header (.h extension) and implementation files (.m extension). For an example of exporting a module see the following classes: `ExampleManager.h`, `ExampleManager.m` and `Component.js`.

```
1 // ExampleManager.h
2 #import "RCTViewManager.h"
3
4 @interface ExampleManager : RCTViewManager
5 @end
```

```
1 // ExampleManager.m
2 #import <YourCustomView.h>
3
4 #import "RCTViewManager.h"
5 #import "ExampleManager.h"
6
7 @implementation ExampleManager
8
9 RCT_EXPORT_MODULE()
10
11 - (UIView *)view
12 {
13     return [[YourCustomView alloc] init];
14 }
15
16 @end
```

```
1 // Component.js
2 import {React, requireNativeComponent} from 'react-native';
3
4 var Example = requireNativeComponent('Example', Component);
5
6 class Component extends React.Component {
7   render() {
8     return <Example />;
9   }
10}
11
12 module.exports = Component;
```

`ExampleManager.h` is a header file and it solely defines the interface and possible attributes or parameters for the main class. For our purposes it extends `RCTViewManager` who takes care of views in React Native, hence the suffix `RCT`.

`ExampleManager.m` implements `view` function which returns `UIView*` on line 11. This function is called by default on any class representing a view in Objective C, when it is mounted in the view hierarchy.

`Component.js` is the actual component we use in our React Native application. In order to use Native UI Component which we defined in `ExampleManager.h` and `ExampleManager.m` we add an import on line 2 and require native component on line 4. After that the native view is ready to use as a simple react component as you can see on line 8.

3.2.4.2 Integration of Wikitude and RN

To start using Wikitude SDK you first need to add it to our XCode project and create a ViewController as described in the official setup guide at [2]. There is also a sample project available at [37] that shows an example of ViewController implementation. For my purposes I had to modify the sample implementation. The complete implementation for integration of Wikitude and React Native is found in Appendix: Integration of Wikitude and React Native.

However this implementation works while testing in simulator and with cable attached, it crashes the application once tested on device not connected with the cable. After thorough research I found out that React Native core mounts each view twice for styling reasons [38]. This creates a memory leak because the `ArchitectWorld` object in the application is allocated twice, which is an unwanted behaviour for the Wikitude. The simulator only throws warning and is not affected by the memory leak. On the contrary when the application is tested on device and the Augmented Reality is launched it crashes the application, shutting it down with no chance of continuing using it. I didn't come across that issue sooner because my initial research didn't show any signs of incompatibility of React Native and Wikitude. Furthermore I

3. REALISATION

didn't find any example of their integration so I simply thought due to the young age of React Native that it hasn't been tried out before.

To conclude Wikitude is not entirely suitable for using with React Native at the moment. However I believe with growing popularity of React Native there will be a plugin released for it soon. Once the plugin is out I could change the Augmented Reality logic part of the application and replace it with the plugin.

3.2.5 Libraries used

I have used npm package manager for taking care of the dependencies in the project. Among all the libraries I have used there are a few I would like to mention.

- react-native-image-picker
- react-native-autocomplete
- react-native-maps

3.3 Web Application

Web application is implemented using React and Redux, which has been already described in detail in the previous section Redux.

3.3.1 Routing

I used `redux-router` [39] for handling the routing in the web application. It is a routing library for the React which allows to create user friendly, pretty urls without hashes and use browser history. Since React is view only framework all the routes work only if you start navigating from the index route. Basically you have to have the application loaded and then you are able to move around changing routes. Because when your route is missing the hash, which is usually present in javascript application routes, the request is sent to the server. Once you refresh any of the pages which is not the index page the route will not be found since there is no such route on the server. Therefore for React routing to function properly there is a need for a server. There are several options of how to accomplish that.

Firstly there is server side rendering option. The server will be serving complete html pages with data and everything needed. In the case of high load and data heavy websites it is a very good choice since it brings high performance and is also SEO friendly. However Google seems to be enhancing their indexing and crawling mechanisms in order to be able to crawl through javascript pages [40].

Secondly a simple server which will serve as a proxy can be created. This option seemed more suitable for me. The web application is pretty basic so the performance increase from moving to server rendering will not be that essential. For the meantime for the purposes of my web application server with a proxy is enough.

I implemented a simple express server which on all incoming requests redirect to index.html allowing the application to load and serve the user requested route. The example of the express server can be found in Appendix F

3.3.2 Libraries used

- react-bootstrap
- react-masonry-component
- react-d3

3.4 Testing

3.4.1 Data testing

3.4.2 Unit testing

Conclusion

Bibliography

- [1] McKitterick, W. Messaging apps are now bigger than social networks. January 2016, [Accessed: 12 Mar 2016]. Available from: <http://www.businessinsider.com/the-messaging-app-report-2015-11>
- [2] Wikitude GmbH. *Wikitude developer documentation*. [Accessed: 24 Mar 2016]. Available from: <http://www.wikitude.com/developer/documentation/ios>
- [3] Siltanen, S. Theory and applications of marker-based augmented reality. Technical report, VTT Technical Research Centre of Finland, 2012.
- [4] Lesage, G. PRAugmentedReality. <https://github.com/promet/PRAugmentedReality>, 2013, [Accessed: 2 Feb 2016].
- [5] Anicas, M. *An Introduction to OAuth 2*. Digital Ocean, July 2014, [Accessed: 20 Mar 2016]. Available from: <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>
- [6] Anderson, M. K. Why We're Thinking About Messaging Apps All Wrong. February 2016, [Accessed: 12 Mar 2016]. Available from: <http://blog.hubspot.com/marketing/messaging-apps>
- [7] Capan, T. Why The Hell Would I Use Node.js? A Case-by-Case Tutorial. <https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js> [Accessed: 2 Feb 2016], 2013.
- [8] Peeples, K. What are the Benefits of Node.js? <https://dzone.com/articles/what-are-benefits-nodejs> [Accessed: 3 Feb 2016], May 2015.
- [9] Node.js Foundation. *Node.js*. [Accessed: 3 Feb 2016]. Available from: <https://nodejs.org>

BIBLIOGRAPHY

- [10] McKeachie, C. React.js and How Does It Fit In With Everything Else? July 2014, [Accessed: 12 Mar 2016]. Available from: <http://www.funnyant.com/reactjs-what-is-it/>
- [11] Facebook. *Why React?* [Accessed: 14 Mar 2016]. Available from: <https://facebook.github.io/react/docs/why-react.html>
- [12] Occhino, T. React Native: Bringing modern web techniques to mobile. March 2015, [Accessed: 14 Mar 2016]. Available from: <https://code.facebook.com/posts/1014532261909640/react-native-bringing-modern-web-techniques-to-mobile/>
- [13] Rosa, A. D. Introduction to the React JavaScript Library. February 2015, [Accessed: 14 Mar 2016]. Available from: <http://developer.telerik.com/featured/introduction-to-the-react-javascript-framework/>
- [14] MongoDB Inc. *Introduction to MongoDB*. [Accessed: 20 Mar 2016]. Available from: <https://docs.mongodb.org/manual/introduction/>
- [15] Parker, Z.; Poe, S.; Vrbsky, S. V. Comparing NoSQL MongoDB to an SQL DB. In *Proceedings of the 51st ACM Southeast Conference*, ACMSE '13, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-1901-0, pp. 5:1–5:6, doi:10.1145/2498328.2500047. Available from: <http://doi.acm.org/10.1145/2498328.2500047>
- [16] Bperry. You have no SQL inj... sorry, NoSQL injections in your application. June 2014, [Accessed: 14 Mar 2016]. Available from: <https://community.rapid7.com/community/metasploit/blog/2014/06/12/you-have-no-sql-inj--sorry-nosql-injections-in-your-application>
- [17] MongoDB Inc. *How does MongoDB address SQL or Query injection?* [Accessed: 14 Mar 2016]. Available from: <https://docs.mongodb.org/manual/faq/fundamentals/#how-does-mongodb-address-sql-or-query-injection>
- [18] Ron, A.; Shulman-Peleg, A.; Bronshtein, E. No SQL, No Injection? Examining NoSQL Security. *CoRR*, volume abs/1506.04082, 2015. Available from: <http://arxiv.org/abs/1506.04082>
- [19] Breshean, J. Diving into Webpack. <https://web-design-weekly.com/2014/09/24/diving-webpack/>, 2014, [Accessed: 25 Apr 2016].
- [20] Truong, A. Today's Most Innovative Company: IKEA Uses Augmented Reality to Show How Furniture Fits in a Room. *Fast Company*, volume 26, 2013.

Bibliography

- [21] Kudan. *Express*. [Accessed: 2 Feb 2016]. Available from: <https://www.kudan.eu/>
- [22] StrongLoop, IBM,. *Express*. [Accessed: 4 Apr 2016]. Available from: <http://expressjs.com/>
- [23] Kelley, A. Multiparty. <https://github.com/andrewrk/node-multiparty>, 2015, [Accessed: 18 Apr 2016].
- [24] Andersson, R. Node-Imagemagick. <https://github.com/yourdeveloper/node-imagemagick>, 2015, [Accessed: 18 Apr 2016].
- [25] ImageMagick Studio LLC. *ImageMagick*. [Accessed: 24 Mar 2016]. Available from: <http://www.imagemagick.org/>
- [26] NGINX Inc. *Nginx load balancing - http load balancer*. [Accessed: 18 Apr 2016]. Available from: <https://www.nginx.com/resources/admin-guide/load-balancer/>
- [27] Google. *The Google Maps Geocoding API*. [Accessed: 15 Apr 2016]. Available from: <https://developers.google.com/maps/documentation/geocoding/intro>
- [28] RedisLabs. *Redis*. [Accessed: 15 Apr 2016]. Available from: <http://redis.io/>
- [29] Auttomatic. Kue. <https://github.com/Automattic/kue>, 2016, [Accessed: 10 Apr 2016].
- [30] Agendaless Consulting and Contributors. *Supervisor: A Process Control System*. [Accessed: 28 Apr 2016]. Available from: <http://supervisord.org/>
- [31] Hanson, J. Passport. <https://github.com/jaredhanson/passport>, 2016, [Accessed: 18 Apr 2016].
- [32] McMahon, C. Async. <https://github.com/caolan/async>, 2016, [Accessed: 20 Apr 2016].
- [33] Babel. Babel. <https://babeljs.io/>, 2016, [Accessed: 20 Apr 2016].
- [34] Haswell, B. Mongoose-unique-validator. <https://github.com/blakehaswell/mongoose-unique-validator>, 2016, [Accessed: 20 Apr 2016].
- [35] Facebook. Flux. [Accessed: 18 Apr 2016]. Available from: <https://facebook.github.io/flux/>

BIBLIOGRAPHY

- [36] Elouafi, Y. *Redux-saga documentation*. [Accessed: 20 Mar 2016]. Available from: <http://yelouafi.github.io/redux-saga/docs>
- [37] Wikitude. *wikititude-sdk-basic-projects*. <https://github.com/Wikitude/wikititude-sdk-basic-projects>, 2016, [Accessed: 20 Apr 2016].
- [38] Facebook. *RCTUIManager.m*. <https://github.com/facebook/react-native/blob/master/React/Modules/RCTUIManager.m#L801>, 2016, [Accessed: 24 Apr 2016].
- [39] Community, R. *react-router*. <https://github.com/reactjs/react-router>, 2016, [Accessed: 20 Apr 2016].
- [40] Nagayama, K. Deprecating our AJAX crawling scheme. <https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html>, 2015, [Accessed: 20 Apr 2016].

APPENDIX A

Acronyms

GUI Graphical user interface

XML Extensible markup language

APPENDIX **B**

Installation guide

B.1 Server

B.2 Mobile application

B.3 Web application

APPENDIX **C**

Web application screens

APPENDIX **D**

Mobile application screens

Supervisor configuration

Default directory for the supervisor is where the configuration file is. Most of the configuration is default, the only changed lines are 20 to 39, the program configuration.

```

1 [unix_http_server]
2 file=/tmp/supervisor.sock ; (the path to the socket file)
3
4 [supervisord]
5 logfile=/supervisor/supervisord.log ; (main log file)
6 logfile_maxbytes=50MB ; (max logfile bytes b4 rotation)
7 logfile_backups=10 ; (main logfile rotation backups)
8 loglevel=info ; (log level; default info;)
9 pidfile=/tmp/supervisord.pid ; (supervisord pidfile)
10 nodaemon=false ; (start in foreground if true)
11 minfds=1024 ; (min. startup file descriptors)
12 minprocs=200 ; (min. process descriptors)
13
14 [rpcinterface:supervisor]
15 supervisor.rpcinterface_factory = supervisor.rpcinterface:
   make_main_rpcinterface
16
17 [supervisorctl]
18 serverurl=unix:///tmp/supervisor.sock ; use a unix:// URL for a
   unix socket
19
20 [program:airServer]
21 command=npm start
22 autostart=true
23 stdout_logfile=/logs/supervisor/server_out.log
24 stderr_logfile=/logs/supervisor/server_err.log
25
26 [program:geoConsumer]
27 directory=/kue
28 command=node geoConsumer.js
29 autostart=true
30 stdout_logfile=/logs/supervisor/geoConsumer_out.log
31 stderr_logfile=/logs/supervisor/geoConsumer_err.log

```

E. SUPERVISOR CONFIGURATION

```
32 [program:validConsumer]
33 directory=/kue
34 command=node validConsumer.js
35 autostart=true
36 stdout_logfile=/logs/supervisor/validConsumer_out.log
37 stderr_logfile=/logs/supervisor/validConsumer_err.log
```

Express server for React routing

```
1 // required for converting JSX syntax
2 require("node-jsx").install();
3 require("babel-core/register")({ "presets": ["es2015"] });
4 var express = require('express');
5 var path = require('path');
6 var app = express();
7
8 // serving static files
9 app.use(express.static(path.join(__dirname, 'public')));
10
11 app.get('*', function (request, response) {
12   response.sendFile(path.resolve(__dirname, 'public', 'index.html'
13     ))
14 });
15
16 // development error handler - will print stacktrace
17 if (app.get('env') === 'development') {
18   app.use(function(err, req, res, next) {
19     res.status(err.status || 500);
20     res.render('error', {
21       message: err.message,
22       error: err
23     });
24   })
25 }
26
27 // production error handler - no stacktraces leaked to user
28 app.use(function(err, req, res, next) {
29   res.status(err.status || 500);
30   res.render('error', {
31     message: err.message,
32     error: {}
33   });
34 });
35 module.exports = app;
```


Integration of Wikitude and React Native

In order to integrate Wikitude and React Native I implemented the following files:

- ViewController.h
- ViewController.m - starting point for the Wikitude
- ARViewManager.h
- ARViewManager.m - exported view component for React Native

```
1 // ViewController.h
2
3 #import <UIKit/UIKit.h>
4
5 @interface ViewController : UIViewController
6
7 -(UIView*)start;
8
9 @end
```

```
1 // ViewController.m
2
3 #import "ViewController.h"
4 #import <WikitudeSDK/WikitudeSDK.h>
5 /* Wikitude SDK debugging */
6 #import <WikitudeSDK/WTArchitectViewDebugDelegate.h>
7
8 @interface ViewController () <WTArchitectViewDelegate,
9 WTArchitectViewDebugDelegate>
10 /* Add a strong property to the main Wikitude SDK component */
11 @property (nonatomic, strong) WTArchitectView *architectView;
12
```

G. INTEGRATION OF WIKITUDE AND REACT NATIVE

```
13 /* And keep a weak property to the navigation object which
14    represents the loading status of your Architect World */
15 @property (nonatomic, weak) WTNavigation *
16     architectWorldNavigation;
17
18 @implementation ViewController
19
20 - (void)dealloc
21 {
22     [[NSNotificationCenter defaultCenter] removeObserver:self];
23 }
24
25 - (UIView*)start {
26     NSError *deviceSupportError = nil;
27
28     if ( [WTArchitectView isDeviceSupportedForRequiredFeatures:
29          WTFeature_Geo error:&deviceSupportError] ) {
30
31         /* Standard WTArchitectView object creation and initial
32            configuration */
33         self.architectView = [[WTArchitectView alloc] initWithFrame:
34             CGRectMakeZero motionManager:nil];
35         self.architectView.delegate = self;
36         self.architectView.debugDelegate = self;
37
38         /* Use the -setLicenseKey method to unlock all Wikitude SDK
39            features that you bought with your license. */
40         [self.architectView setLicenseKey:@"licenceKey"];
41
42         /* The Architect World can be loaded independently from the
43            WTArchitectView rendering.
44
45             NOTE: The architectWorldNavigation property is assigned at
46             this point. The navigation object is valid until another
47             Architect World is loaded.
48
49 */
50
51         NSURL *baseURL = [NSURL URLWithString:(@"http://url-to-your/
52             architect-world/index.html")];
53         self.architectWorldNavigation = [self.architectView
54             loadArchitectWorldFromURL:baseURL withRequiredFeatures:
55             WTFeature_Geo];
56
57         /* Because the WTArchitectView does some OpenGL rendering,
58            frame updates have to be suspended and resumend when the
59            application changes it's active state.
60            Here, UIApplication notifications are used to respond to the
61            active state changes.
62
63         NOTE: Since the application will resign active even when an
64             UIAlertView is shown, some special handling is implemented
```

```

        in the UIApplicationDidBecomeActiveNotification.
51     */
52     [[NSNotificationCenter defaultCenter] addObserverForName:
53         UIApplicationDidBecomeActiveNotification object:nil queue
54         :[NSOperationQueue mainQueue] usingBlock:^(NSNotification
55         *note) {
56
56     /* When the application starts for the first time, several
57         UIAlert's might be shown to ask the user for camera and
58         /or GPS access.
59     Because the WTArchitectView is paused when the application
60         resigns active (See line 86), also Architect
61         JavaScript evaluation is interrupted.
62     To resume properly from the inactive state, the Architect
63         World has to be reloaded if and only if an active
64         Architect World load request was active at the time
65         the application resigned active.
66     This loading state/interruption can be detected using the
67         navigation object that was returned from the -
68             loadArchitectWorldFromURL:withRequiredFeatures method.
69     */
70     if (self.architectWorldNavigation.wasInterrupted) {
71         [self.architectView reloadArchitectWorld];
72     }
73
74     /* Standard WTArchitectView rendering resuming after the
75         application becomes active again */
76     [self startWikitudeSDKRendering];
77 ];
78     [[NSNotificationCenter defaultCenter] addObserverForName:
79         UIApplicationWillResignActiveNotification object:nil
80         queue:[NSOperationQueue mainQueue] usingBlock:^(
81         NSNotification *note) {
82
82     /* Standard WTArchitectView rendering suspension when the
83         application resignes active */
84     [self stopWikitudeSDKRendering];
85 ];
86
87     /* Standard subview handling using Autolayout */
88     [self.view addSubview:self.architectView];
89     self.architectView.translatesAutoresizingMaskIntoConstraints = NO;
90
91     NSDictionary *views = NSDictionaryOfVariableBindings(
92         _architectView);
93     [self.view addConstraints: [NSLayoutConstraint
94         constraintsWithVisualFormat:@"|[[_architectView]|" options
95         :0 metrics:nil views:views] ];
96     [self.view addConstraints: [NSLayoutConstraint
97         constraintsWithVisualFormat:@"V:|[[_architectView]|"
98         options:0 metrics:nil views:views] ];
99     return self.view;
100 }
```

G. INTEGRATION OF WIKITUDE AND REACT NATIVE

```
81     else {
82         NSLog(@"This device is not supported. Show either an alert or
83             use this class method even before presenting the view
84             controller that manages the WTArchitectView. Error: %@", [deviceSupportError localizedDescription]);
85         return nil;
86     }
87 }
88
89 #pragma mark - View Lifecycle
90 - (void)viewWillAppear:(BOOL)animated {
91     NSLog(@"%@", "will appear");
92     [super viewWillAppear:animated];
93
94     /* WTArchitectView rendering is started once the view
95      controllers view will appear */
96     [self startWikitudeSDKRendering];
97 }
98
99 - (void)viewWillDisappear:(BOOL)animated {
100    NSLog(@"%@", "will disappear");
101    [super viewWillDisappear:animated];
102
103    /* WTArchitectView rendering is stopped once the view
104     controllers view did disappear */
105    [self stopWikitudeSDKRendering];
106 }
107
108 #pragma mark - View Rotation
109 - (BOOL)shouldAutorotate {
110
111     return YES;
112 }
113
114 - (NSUInteger)supportedInterfaceOrientations {
115
116     return UIInterfaceOrientationMaskAll;
117 }
118
119 - (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)
120     toInterfaceOrientation duration:(NSTimeInterval)duration {
121
122     /* When the device orientation changes, specify if the
123      WTArchitectView object should rotate as well */
124     [self.architectView setShouldRotate:YES toInterfaceOrientation:
125         toInterfaceOrientation];
126 }
127
128 #pragma mark - Private Methods
```

```

127
128 /* Convenience methods to manage WTArchitectView rendering. */
129 - (void)startWikitudeSDKRendering{
130
131     /* To check if the WTArchitectView is currently rendering, the
132        isRunning property can be used */
133     if ( ![self.architectView isRunning] ) {
134         NSLog(@"architect running. started");
135         /* To start WTArchitectView rendering and control the startup
136            phase, the -start:completion method can be used */
137         [self.architectView start:^(WTStartupConfiguration *
138             configuration) {
139
140             /* Use the configuration object to take control about the
141                WTArchitectView startup phase */
142             /* You can e.g. start with an active front camera instead
143                of the default back camera */
144
145             } completion:^(BOOL isRunning, NSError *error) {
146
147                 /* The completion block is called right after the internal
148                    start method returns.
149
150                     NOTE: In case some requirements are not given, the
151                         WTArchitectView might not be started and returns NO
152                         for isRunning.
153
154                     To determine what caused the problem, the localized error
155                         description can be used.
156
157                     */
158                 if ( !isRunning ) {
159                     NSLog(@"WTArchitectView could not be started. Reason: %@",

160                         [error localizedDescription]);
161
162                 }
163             }];
164
165         - (void)stopWikitudeSDKRendering {
166
167             /* The stop method is blocking until the rendering and camera
168                access is stopped */
169             if ( [self.architectView isRunning] ) {
170                 [self.architectView stop];
171             }
172
173             /* The WTArchitectView provides two delegates to interact with.
174                */
175             #pragma mark - Delegation
176
177             /* The standard delegate can be used to get information about:
178             * The Architect World loading progress
179             * architectsdk:// protocol invocations using document.location
180                 inside JavaScript

```

G. INTEGRATION OF WIKITUDE AND REACT NATIVE

```
168 * Managing view capturing
169 * Customizing view controller presentation that is triggered
170     from the WTArchitectView
171 */
172 #pragma mark WTArchitectViewDelegate
173 - (void)architectView:(WTArchitectView *)architectView
174     didFinishLoadArchitectWorldNavigation:(WTNavigation *)
175         navigation {
176     /* Architect World did finish loading */
177 }
178
179 - (void)architectView:(WTArchitectView *)architectView
180     didFailToLoadArchitectWorldNavigation:(WTNavigation *)
181         navigation withError:(NSError *)error {
182
183     NSLog(@"Architect World from URL '%@' could not be loaded.
184             Reason: %@", navigation.originalURL, [error
185                 localizedDescription]);
186 }
187
188 /* The debug delegate can be used to respond to internal issues,
189    e.g. the user declined camera or GPS access.
190
191 NOTE: The debug delegate method -architectView:
192       didEncounterInternalWarning is currently not used.
193 */
194 #pragma mark WTArchitectViewDebugDelegate
195 - (void)architectView:(WTArchitectView *)architectView
196     didEncounterInternalWarning:(WTWarning *)warning {
197
198     /* Intentionally Left Blank */
199 }
200
201 - (void)architectView:(WTArchitectView *)architectView
202     didEncounterInternalError:(NSError *)error {
203
204     NSLog(@"WTArchitectView encountered an internal error '%@',
205             [error localizedDescription]);
206 }
207
208 @end
```

```
1 //  ARViewManager.h
2
3 #import "RCTViewManager.h"
4
5 @interface ARViewManager : RCTViewManager
6
7 @end
```

```
1 //  ARViewManager.m
2
3 #import "ARViewManager.h"
4 #import "ViewController.h"
5
6 @implementation ARViewManager
7
8 RCT_EXPORT_MODULE()
9
10 - (UIView *)view
11 {
12     ViewController* controller = [[ViewController alloc] init];
13     return [controller start];
14 }
15
16 @end
```


Contents of enclosed CD

readme.txt	the file with CD contents description
exe	the directory with executables
src.....	the directory of source codes
└── wbdcm	implementation sources
└── thesis.....	the directory of L ^A T _E X source codes of the thesis
text	the thesis text directory
└── thesis.pdf.....	the thesis text in PDF format
└── thesis.ps.....	the thesis text in PS format