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: Jan Vaclavik

6th May 2016

Acknowledgements

THANKS

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 6th May 2016 .....

Czech Technical University in Prague
Faculty of Information Technology

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

Summarize the contents and contribution of your work in a few sentences in English language.

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

Contents

List of Figures

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]

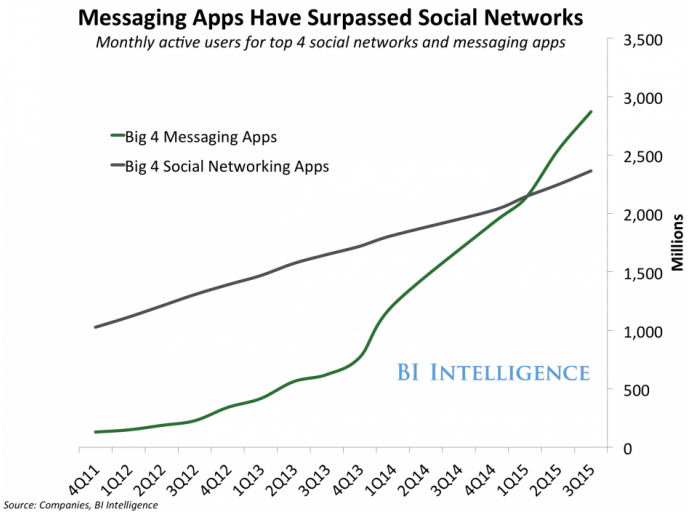

Figure 0.1: Messaging apps have surpassed social networks
Source: [1]

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

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 o 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

1.1 Messaging application

The main goal is to create a messaging application which will allow users to pin messages to GPS location. There will be two types of messages. Firstly users will post simple text messages with an option of attaching a picture. Secondly users are able to post a 3D object. Both types of messages are comment-able. 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.

This application serves as a real time companion. There is no overload of information you do not want to see at the moment as you have when scrolling through social media feeds. The app serves you relevant and fresh information left by other users based on your current location.

1.2 Functional requirements

1. Registration

   User can register using their email and password.

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. There is an option of adding an image to the message.

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 has 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 amount of message.

9. View and change user details

   The user has a choice of either showing details about himself or not. The only compulsory public detail is username. The detail which can be shared with other is user's country.

1.3 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.

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:

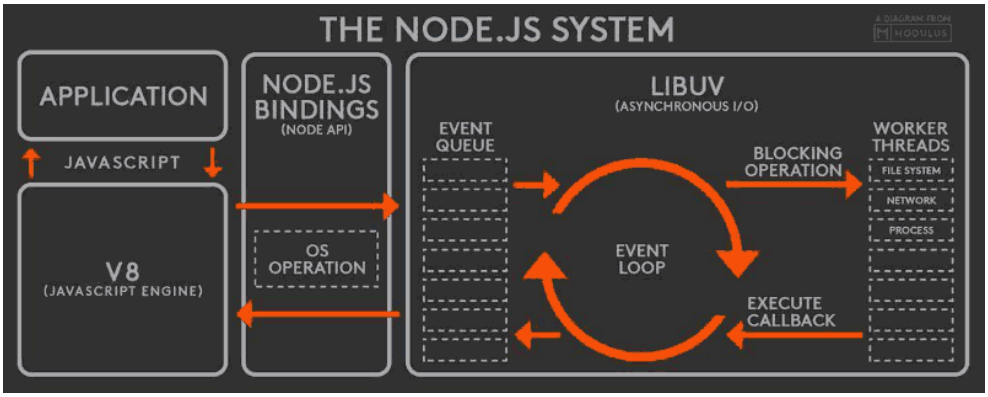

Figure 2.1: Node.js system

- 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]

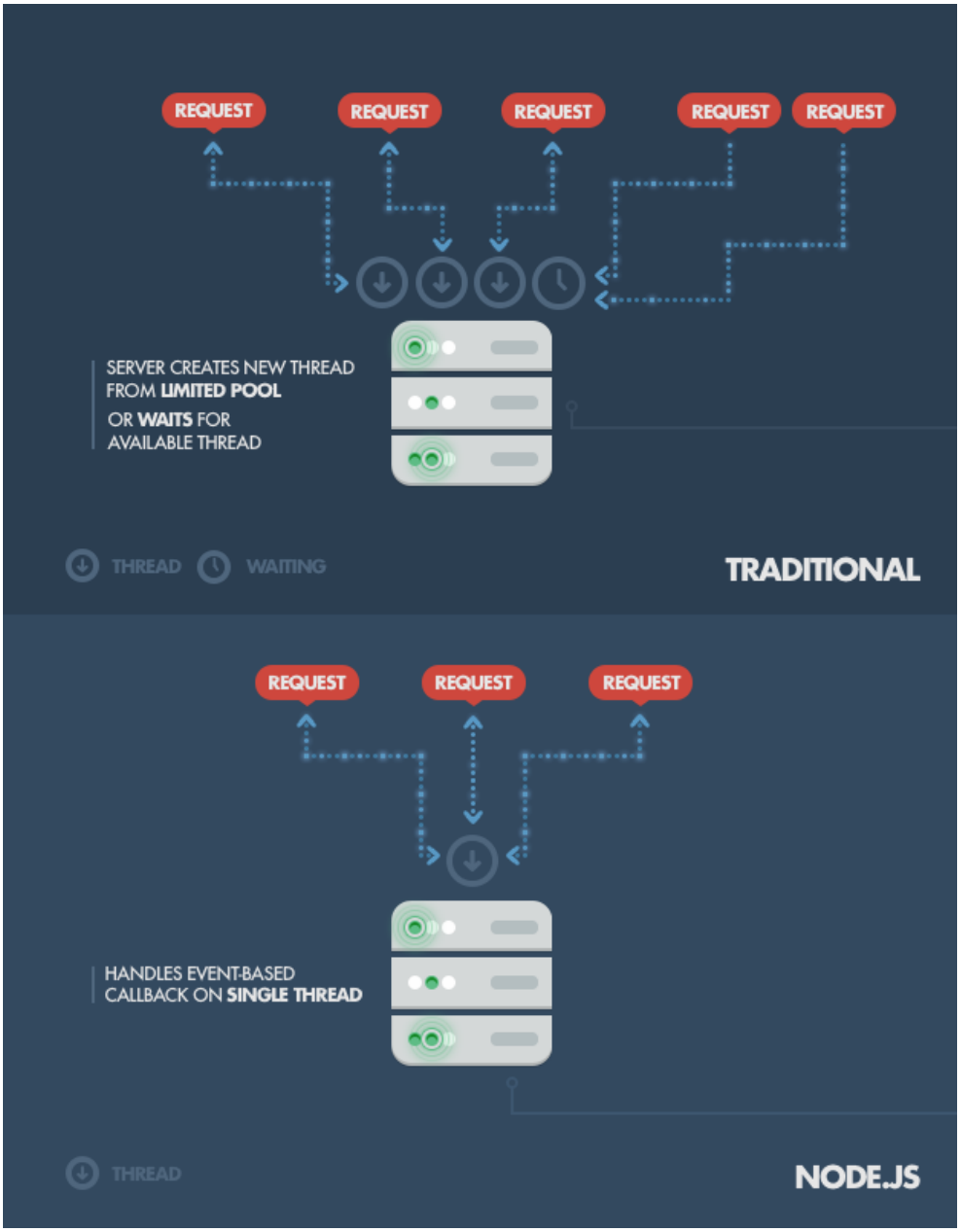

Figure 2.2: Node.js server processing compared with traditional web server processing of requests

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

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 it's 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.

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 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]

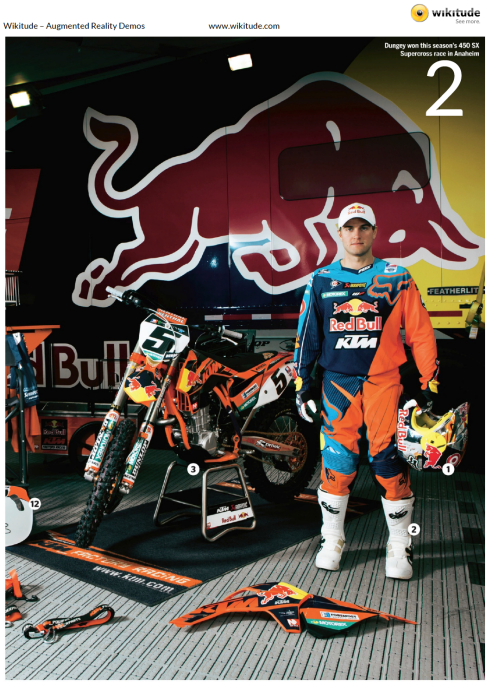

Figure 2.3: Photo marker example
Source: [2]

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.[19]

- 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 environment. Both methods map the environment creating feature maps using various feature detection methods and algorithms. It is not within

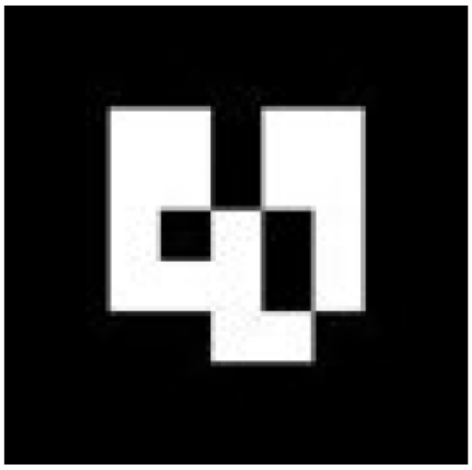

Figure 2.4: Black and white marker example
Source: [3]

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

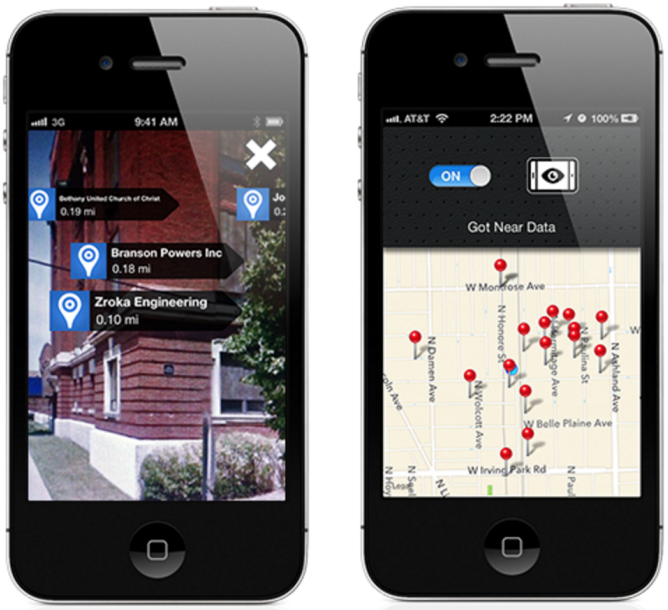

Figure 2.5: Points of interest
Source: [4]

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 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 a database design. It is essential to understand how will your data 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 scheme 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.

MongoDb has a different view on data than relational databases. Relational 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.

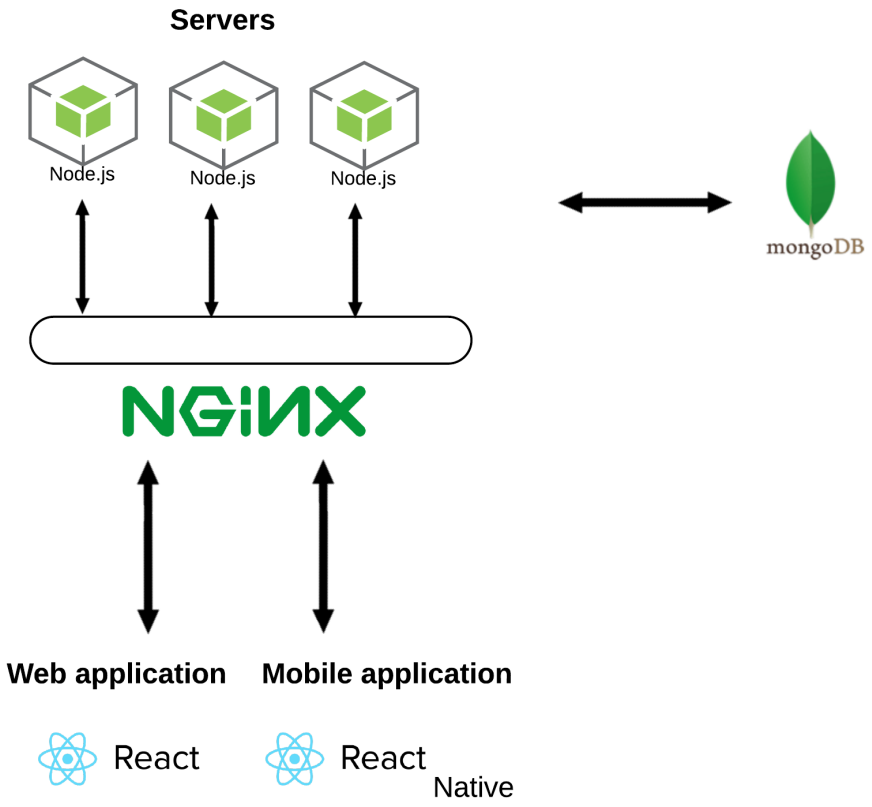

Figure 2.6: Interaction between parts of the system

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, 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 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.

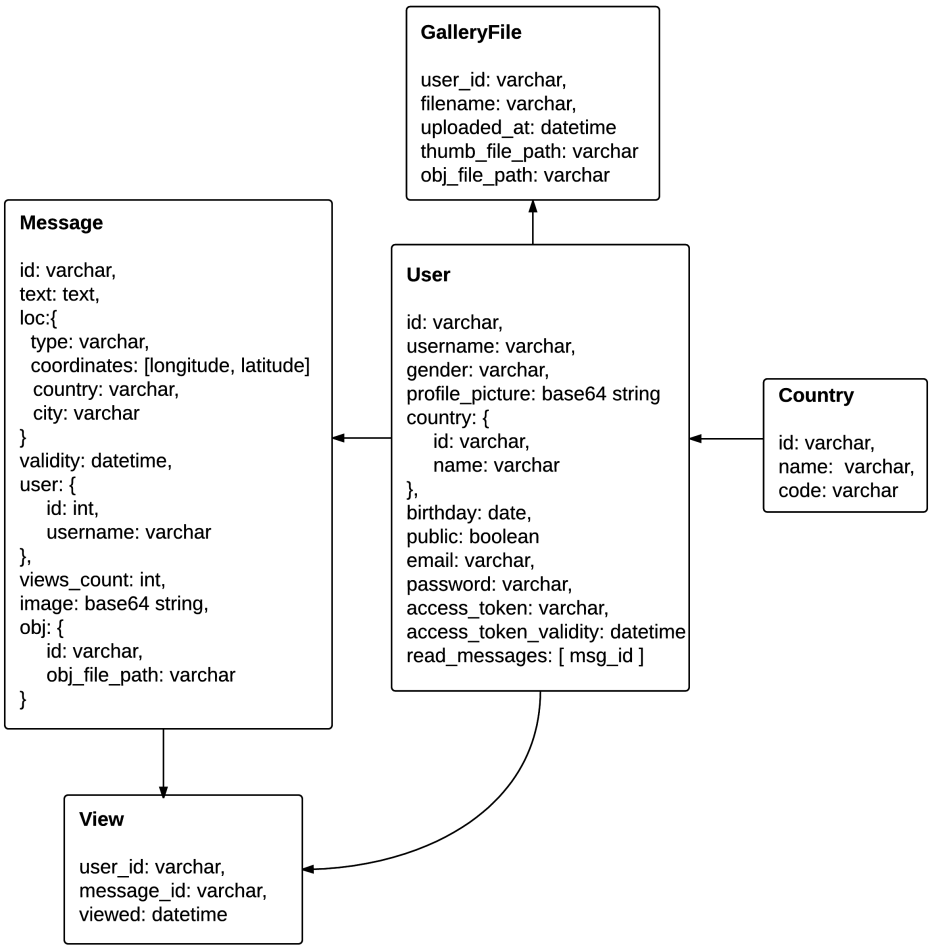

Figure 2.7: MongoDB model of GPS pinned messaging application

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 an array of read message ids. We need to note down individual

messages that user read so we can notify the user about comments on these messages. User document has a country object saved in it to save time querying.

- 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.

- Country

  I chose to create a separate country collection since I need to provide users with a country selection for their profile and it has to be unified for all users.

- 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.

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 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 is 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 experince using its Advanced tracking for both marker and markerless augmented realities. For more information about Kudan please see official website at [20]

  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 minimalistic 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 [21].

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

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

- 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 3d 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",
```

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

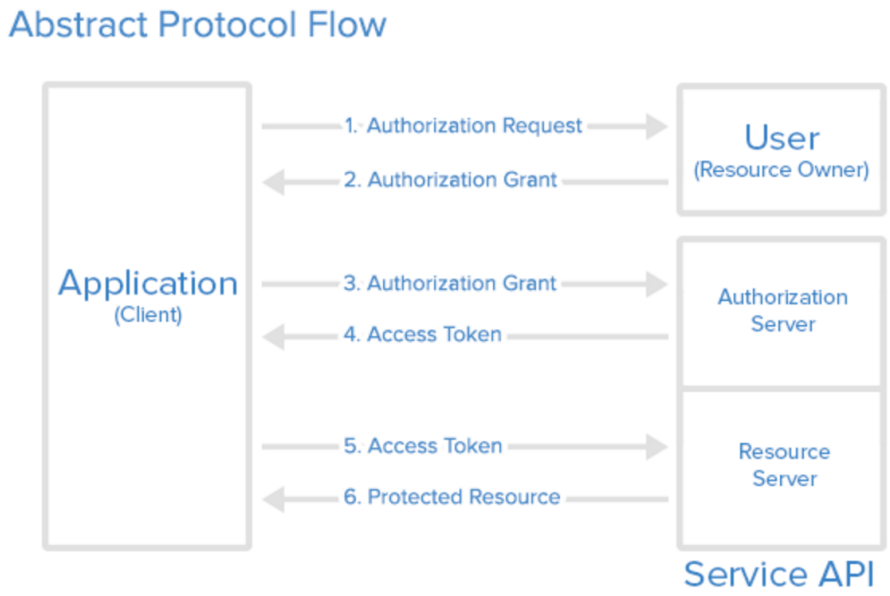

Figure 3.1: OAuth2 authentication protocol flow.
Source: [5]

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 [22]. 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 [23] was chosen, which uses original `ImageMagick` library availble at [24].

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.

```
"loc" : {
    "type" : "Point",
        "coordinates" : [
        14.422061,
        50.087657
    ]
}
```

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

```
loc: {
    $geoWithin: {
        $centerSphere: [
            [
                longitude,
                latitude
            ],
            radius
        ]
    }
}
```

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. [25] 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 *leact 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/|
               stylesheets/|flash/|media/|static/|robots.txt|humans.
               txt|favicon.ico) {
17             root /path/to/static/files;
18         }
19
20         location / {
21             proxy_pass http://air_servers;
22         }
23     }
24
25     # HTTPS server
26
27     server {
28         listen 443;
29
30         ssl on;
31         ssl_certificate     /etc/ssl/example.com/example.com.crt;
32         ssl_certificate_key /etc/ssl/example.com/example.com.key;
33
34        # plus the same configuration as above
35     }
36 }
```

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 [26], waiting for response, parsing the response and updating the document in the database. This slightly lengthy process could not take part

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 [27]

The complete documentation for `kue` is available at [28]

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 Libraries used

3.1.8 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
└── www ..... node start script
config
└── config.js ..... main config of the server
controllers
├── auth.controller.js ..... authorization strategies
└── oauth2.controller.js ..... OAuth2 flow
data ..... mongodb documents directory
docs
├── api.apib ..... api documentation api blueprint
└── output.html ..... api documentation output
kue ..... kue consumers
├── geoConsumer.js
└── validConsumer.js
models ..... mongoose models
├── client.js
├── comment.js
├── galleryFile.js
├── message.js
├── messageObj.js
├── refreshToken.js
├── token.js
├── user.js
└── view
public
└── gallery ..... 3D gallery files and thumbnails
routes ..... the thesis text directory
├── files.js
├── index.js
├── messages.js
└── users.js
tests
util ..... helpers
├── common.helper.js
├── file.helper.js
├── messages.helper.js
├── query.helper.js
├── request.helper.js
├── string.helper.js
└── view.helper.js
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

3.2.0.1 Redux

Redux is a predictable state container for Javascript application. It is a small library which helps control and structure application's state hierarchy. The main concern in javascript application is taking care of the state of application.

Redux was inspired by Flux, which is a way Facebook creates their applications. But Redux was made smaller and easier to use. Redux's biggest advantage it's the way it handles the data flow. Redux has unidirectional data flow, which provides the developer with a clear and easy to debug application.

The structure of the application using redux is shown on 3.3. Each part of the scheme plays a vital role.

- Presentational and Container Components

  Presentational components are components which do not hold any logic inside themselves. Everything they need for functioning they receive passed in their props from the Container Components. In the case presentational component needs to fire an event it uses callbacks which are passed to it in props. There is no logic in Presentational Components.

  Container Components are the ones who dispatch events when presentational components receive user interaction. They also take care of navigation. Sometimes they referred to as smart components and presentational components as dumb components.

- Action makers

  Action makers are the ones who return actual 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 cookies 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 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.

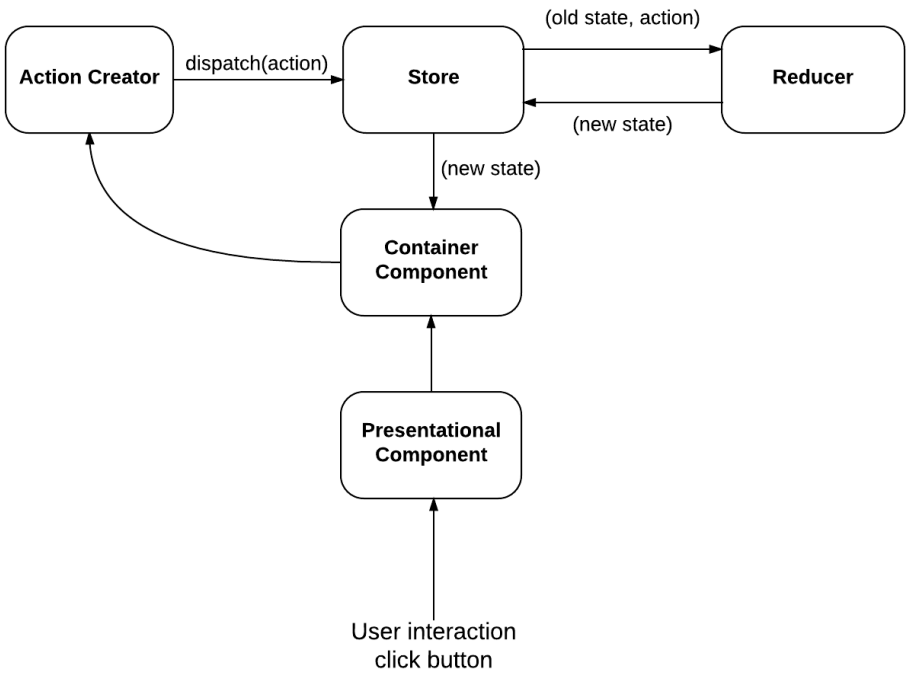

Figure 3.3: Redux structure. Unidirectional dataflow.

3.2.1 Structure

3.2.2 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 my 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 `http://www.wikitude.com/developer/documentation/ios`.

3.2.2.1 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 you application's functionality natively.

There are two macros which are used for this case:

- `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`.

```
// ExampleManager.h
#import "RCTViewManager.h"

@interface ExampleManager : RCTViewManager
@end
```

```
// ExampleManager.m
#import <YourCustomView.h>

#import "RCTViewManager.h"
#import "ExampleManager.h"

@implementation ExampleManager

RCT_EXPORT_MODULE()

- (UIView *)view
{
  return [[YourCustomView alloc] init];
}

@end
```

```
// Component.js
import React, { requireNativeComponent } from 'react-native'
    ;

var Example = requireNativeComponent('Example', Component);

class Component extends React.Component {
  render() {
    return <Example />;
  }
}

module.exports = Component;
```

`ExampleManager.h` is a header file and it solely defines 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 as stated 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.

- `RCT_METHOD_EXPORT(*method*)`

3.2.2.2 Integration of Wikitude and RN

To start using Wikitude SDK we first need to add it to our XCode project and create a ViewController as described in the official setup guide at `http://www.wikitude.com/developer/documentation/ios`. There is also a sample project available at `https://github.com/Wikitude/wikitude-sdk-basic-projects` which 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

3.2.3 Libraries used

All the libraries that were provided by the npm package manager.

3.3 Web Application

3.3.1 Structure

3.3.2 Libraries used

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.

[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

[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] Truong, A. Today's Most Innovative Company: IKEA Uses Augmented Reality to Show How Furniture Fits in a Room. *Fast Company*, volume 26, 2013.

[20] Kudan. *Express*. [Accessed: 2 Feb 2016]. Available from: https://www.kudan.eu/

[21] StrongLoop, IBM,. *Express*. [Accessed: 4 Apr 2016]. Available from: http://expressjs.com/

[22] Kelley, A. Multiparty. https://github.com/andrewrk/node-multiparty, 2015.

[23] Andersson, R. Node-Imagemagick. https://github.com/yourdeveloper/node-imagemagick, 2015.

[24] ImageMagick Studio LLC. *ImageMagick*. [Accessed: 24 Mar 2016]. Available from: http://www.imagemagick.org/

[25] NGINX Inc. *Nginx load balancing - http load balancer*. [Accessed: 18 Apr 2016]. Available from: https://www.nginx.com/resources/admin-guide/load-balancer/

[26] Google. *The Google Maps Geocoding API*. [Accessed: 15 Apr 2016]. Available from: https://developers.google.com/maps/documentation/geocoding/intro

[27] RedisLabs. *Redis*. [Accessed: 15 Apr 2016]. Available from: http://redis.io/

[28] Auttomatic. Kue. https://github.com/Automattic/kue, 2016.

Appendix A

Acronyms

GUI Graphical user interface

XML Extensible markup language

Appendix B

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,
       WTArchitectViewDebugDelegate>
9
10 /* Add a strong property to the main Wikitude SDK component */
11 @property (nonatomic, strong) WTArchitectView *architectView;
12
```

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

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

```
81    else {
82      NSLog(@"This device is not supported. Show either an alert or
            use this class method even before presenting the view
          controller that manages the WTArchitectView. Error: %@",
          [deviceSupportError localizedDescription]);
83      return nil;
84    }
85  }
86
87  #pragma mark - View Lifecycle
88  - (void)viewWillAppear:(BOOL)animated {
89    NSLog(@"%s", "will appear");
90    [super viewWillAppear:animated];
91
92    /* WTArchitectView rendering is started once the view
          controllers view will appear */
93    [self startWikitudeSDKRendering];
94  }
95
96  - (void)viewDidDisappear:(BOOL)animated {
97    NSLog(@"%s", "will dissappear");
98    [super viewDidDisappear:animated];
99
100   /* WTArchitectView rendering is stopped once the view
          controllers view did disappear */
101   [self stopWikitudeSDKRendering];
102 }
103
104 - (void)didReceiveMemoryWarning {
105   [super didReceiveMemoryWarning];
106   // Dispose of any resources that can be recreated.
107 }
108
109 #pragma mark - View Rotation
110 - (BOOL)shouldAutorotate {
111
112   return YES;
113 }
114
115 - (NSUInteger)supportedInterfaceOrientations {
116
117   return UIInterfaceOrientationMaskAll;
118 }
119
120 - (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)
        toInterfaceOrientation duration:(NSTimeInterval)duration {
121
122   /* When the device orientation changes, specify if the
          WTArchitectView object should rotate as well */
123   [self.architectView setShouldRotate:YES toInterfaceOrientation:
          toInterfaceOrientation];
124 }
125
126 #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
        isRunning property can be used */
132   if ( ![self.architectView isRunning] ) {
133     NSLog(@"architect running. started");
134     /* To start WTArchitectView rendering and control the startup
          phase, the -start:completion method can be used */
135     [self.architectView start:^(WTStartupConfiguration *
          configuration) {
136
137       /* Use the configuration object to take control about the
            WTArchitectView startup phase */
138       /* You can e.g. start with an active front camera instead
            of the default back camera */
139
140     } completion:^(BOOL isRunning, NSError *error) {
141
142       /* The completion block is called right after the internal
            start method returns.
143
144        NOTE: In case some requirements are not given, the
            WTArchitectView might not be started and returns NO
            for isRunning.
145        To determine what caused the problem, the localized error
            description can be used.
146        */
147       if ( !isRunning ) {
148         NSLog(@"WTArchitectView could not be started. Reason: %@"
              , [error localizedDescription]);
149       }
150     }];
151   }
152 }
153
154 - (void)stopWikitudeSDKRendering {
155
156   /* The stop method is blocking until the rendering and camera
        access is stopped */
157   if ( [self.architectView isRunning] ) {
158     [self.architectView stop];
159   }
160 }
161
162 /* The WTArchitectView provides two delegates to interact with.
      */
163 #pragma mark - Delegation
164
165 /* The standard delegate can be used to get information about:
166  * The Architect World loading progress
167  * architectsdk:// protocol invocations using document.location
       inside JavaScript
```

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

```
//  ARViewManager.h

#import "RCTViewManager.h"

@interface ARViewManager : RCTViewManager

@end
```

```
//  ARViewManager.m

#import "ARViewManager.h"
#import "ViewController.h"

@implementation ARViewManager

RCT_EXPORT_MODULE()

- (UIView *)view
{
    ViewController* controller = [[ViewController alloc] init];
    return [controller start];
}

@end
```

Appendix C

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 LaTeX 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