CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF MASTER’S THESIS

Title:	GPS pinned messaging application
Student:	Bc. Daria Mikhailova
Supervisor:	Ing. Jan Václavík
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of summer semester 2016/17

Instructions

Create a mobile application for iOS to leave messages pinned to GPS. It will communicate with a node.js server. Create a web application that will serve for viewing user’s statistics, messages and pictures or 3D objects.
A user leaves messages pinned to his current GPS location. To read these messages another user has to be within a short radius. Users will use the map mode to view messages in the area.
Explore possibilities of using Augmented reality to preview 3D objects uploaded through web interface and attached to messages.
The server will communicate with the mobile application and with the web application to provide data for saving messages and other details.
The web application will serve as a portal for previewing messages, statistics, locations visited by the user, and other details.
Use the following technologies: Mobile - React Native iOS, Augmented Reality SDK, Web - React.Js, Server - Node.Js server, REST API.

References

Will be provided by the supervisor.

L.S.

Ing. Michal Valenta, Ph.D.	prof. Ing. Pavel Tvrdík, CSc.
Head of Department	Dean

Prague February 17, 2016

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

10th May 2016

Acknowledgements

I would like to thank my supervisor Ing. Jan Vaclavik for giving me a chance to learn new technologies and bring one of my ideas to life.

Declaration

In Prague on 10th May 2016 ....

Citation of this thesis

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

Abstrakt

Cílem této práce je prozkoumat nový zpusob vytváření mobilních aplikaci pomoci technologii Javascript a implementovat systém pro výměnu zpráv. Systém se bude skládát z mobilní aplikace implementované v React Native, webové aplikace postavene na Reactu a serveru v node.js, ktery bude poskytovat data a sloužit jako autorizační autorita.

Klíčová slova Zprávy, GPS, React Native, React, Node.js, Redux, Rozna Realita, Wikitude.

Abstract

The goal of this thesis is to explore a new way of creating mobile applications using Javascript technologies and implement a complete system for messaging. The system will comprise of a mobile application created using React Native which will introduce a new way of communicating by leaving messages at GPS locations. Furthermore, the web application in React and a REST API server in node.js will be created.

Keywords GPS messaging, React Native, React, Node.js, Redux, Augmented Reality, Wikitude.

Contents

List of Figures

Introduction

The growth in usage 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' 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]

The facts mentioned above clearly indicate that messaging applications and social networks are a vital part of our life. Nevermind the rapid growth in functionality, we need 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 applications and social networks fulfill 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. The internet also offers old school forums and communities where people discuss their common interests such as movies and 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 or application. 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

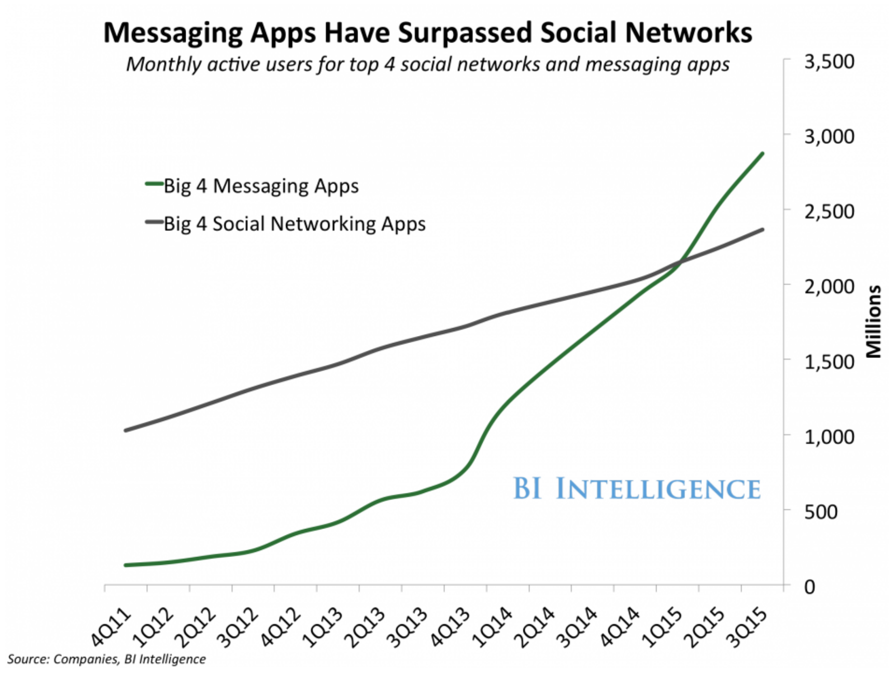


Figure 0.1: Messaging apps have surpassed social networks
Source: [1]

for friends and not long life partners. It's widely considered strange when a person is only looking for friends on dating websites. 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 company to do some sightseeing? What if you just really liked this museum and want to find like-minded people without having to be 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 traveling and you just moved into a hotel. You do not know anyone in the city and would like to have company for 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 to your own GPS location. Anyone could answer your message and it's plausible to quickly find yourself company for a walk in the city.

This GPS messaging application allows a user to leave messages pinned to their GPS location anywhere. It could be a review message saying how much someone liked a certain gallery or a message warning fellow travelers of scams or robberies happening in the area. University students could exchange messages with the whole lecture hall during their lectures. 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, in which you go through the city with new messages occasionally appearing on the map. This brings freedom in messaging. It is almost the same as if 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 the 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 a GPS location, a web application for uploading 3d objects and viewing statistics and a REST API server to communicate with both of them.

The 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 near 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.

The 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

   A user can register with their email and password using mobile application.

2. Authentication

   The user 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 the mobile application. There is an option of adding an image to the message.

4. Leaving a 3D object

   Users will be 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 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 the mobile application. He will be able to upload a profile picture, choose a country, and change his email or password.

1.1.1 System non functional requirements

1. Performance

   There must be a 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

   Both applications 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 registration

1. The user opens the application and taps the button *"Become a member"* on the main screen.

2. The user fills in his username, email and password.

3. If there are no validation errors, the user is redirected to the map view.

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 new message.

4. The user chooses *"text message"* and types in the text. He attaches a picture by tapping the "add image" button and then chooses the picture from his photo library. After the picture is selected, the user saves it and the app performs a redirect to the map view again.

5. The user can now see an icon symbolizing his message next to his current location.

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.

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 an 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 to 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 opens the 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. The 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 redirected 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 mesages

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 tap on the message and open the message detail. The user can read new comments and comment himself.

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 users taps on edit and now can edit his profile information. Each profile edit is confirmed with his password. The user can add a profile picture, change the 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 an open source and cross-platform runtime environment with mostly written in Javascript. Node.js uses non blocking event-driven I/O to remain flexible and lightweight when faced with an 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 the NPM packaging manager. 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 illustrated in figure 2.1. The whole action of Node.js happens in the Event loop. The event loop is a set of events defined in your application which node waits to fire. Once an application sends a request, an event is fired and is placed in the event queue. The event loop picks up the event and processes it. In the case that the event is a blocking event (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]

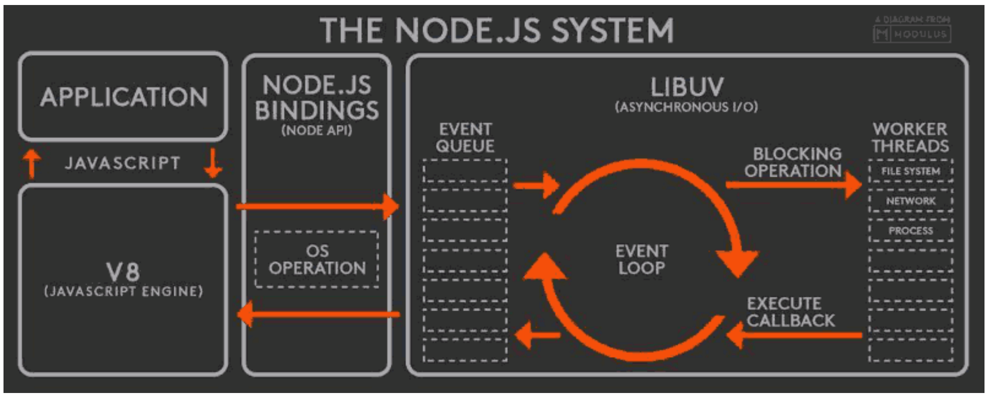


Figure 2.1: Node.js system
Source: [9]

The main advantages of running a node.js server:

- Node.js server can asynchronously process incoming requests. Therefore the response rate is faster.
- Node.js is event-driven, and uses a non blocking I/O by utilizing callbacks, event loop and event queue. This means that we use a browser style concurrency model on the web server.
- Since it is javascript we are able to take advantage of the V8 Javascript Engine. The 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 applications that run across various devices. It is perfect for rapid delivery 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 a new connection thread is created for each request 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]

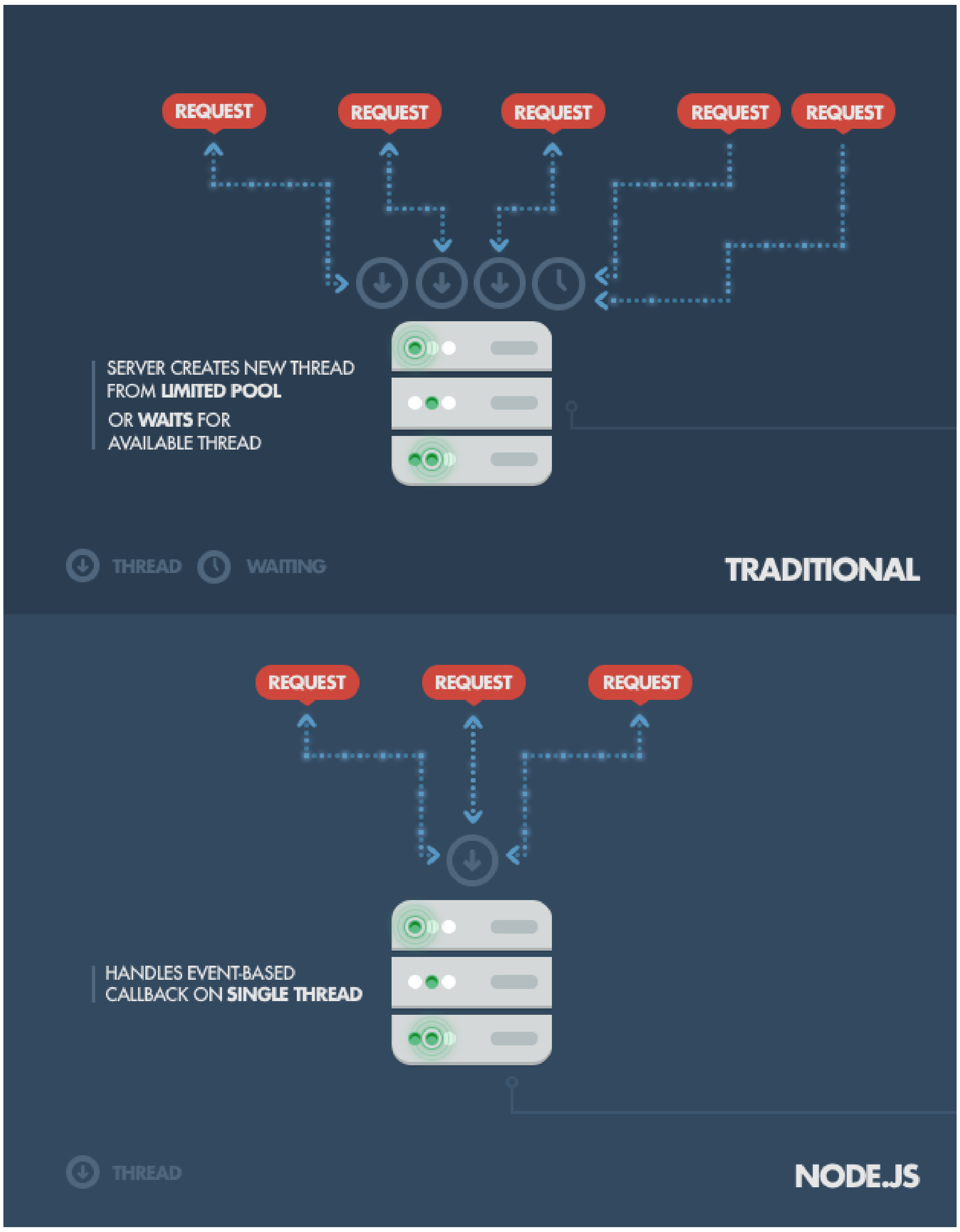

REQUEST
REQUEST
REQUEST
REQUEST
REQUEST
SERVER CREATES NEW THREAD FROM LIMITED POOL
OR WAITS FOR AVAILABLE THREAD
THREAD
WAITING
TRADITIONAL
REQUEST
REQUEST
REQUEST
HANDLES EVENT-BASED CALLBACK ON SINGLE THREAD
THREAD
NODE.JS

Figure 2.2: Node.js server processing compared with traditional web server processing of requests
Source: [9]

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 organized. It breaks down the application into components and makes it very easy to make changes without affecting the rest of the 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 program. 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 applications 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 always been avoiding. However there are quite a few issues 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 rerender 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 renders 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. [10][12][13]

2.1.3 ReactNative

2.1.3.1 Why not use Native?

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 font size. This takes a lot of time from the development process and slows down the whole development cycle. In 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 the 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 choose to 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 and more information is available at [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 a 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 even the smallest 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 parallelizing 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.

Finally, 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 a react native application is a condition 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 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 for 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 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 take 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 modularized it is. It is a good development approach to break your javascript application up 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 to leave 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 recognizes the marker it can define the correct scale and pose of the camera. This technique is commonly known as marker based tracking. A marker can be anything from a clear black and white image to a color photo. The main condition is that the marker has to be recognizable, 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 recognize 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 recognized. 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]

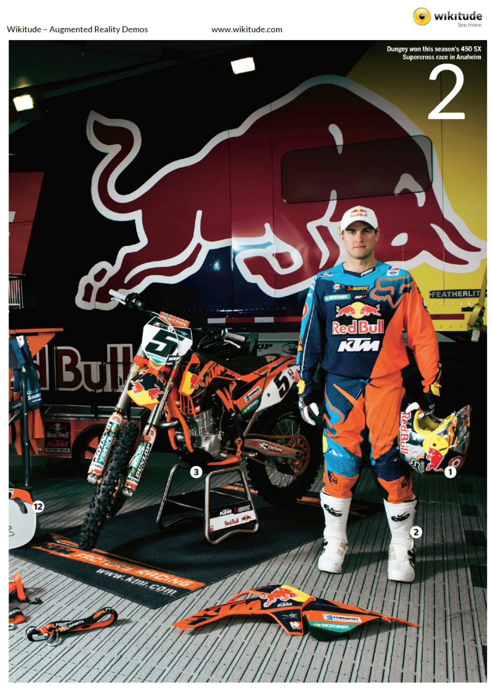

Wikitude – Augmented Reality Demos
www.wikitude.com
wikitude
Dungey won this season's 450 SX Supercross race in Anaheim
2

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

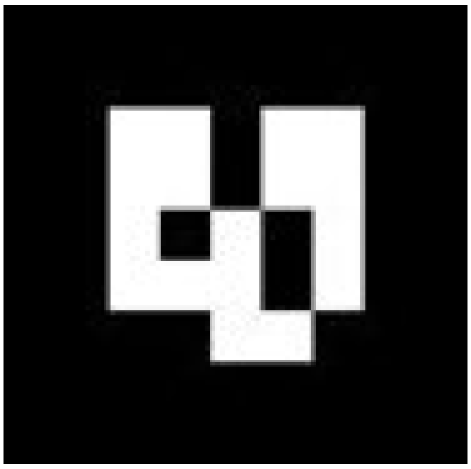


Figure 2.4: Black and white marker example
Source: [3]

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 hold 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 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.
  - GEO 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

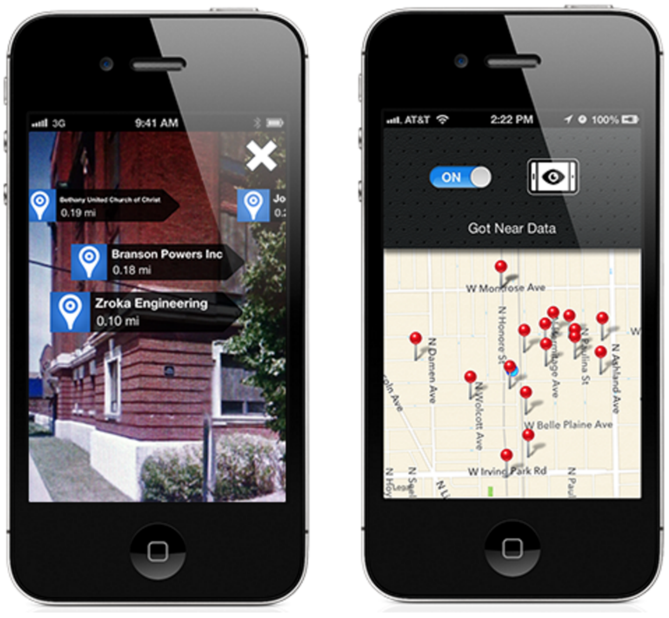


Figure 2.5: Points of interest
Source: [4]

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 a user clicking on links and objects to get more information about the scene, product or a building.

2.2 System design

The goal of this 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.

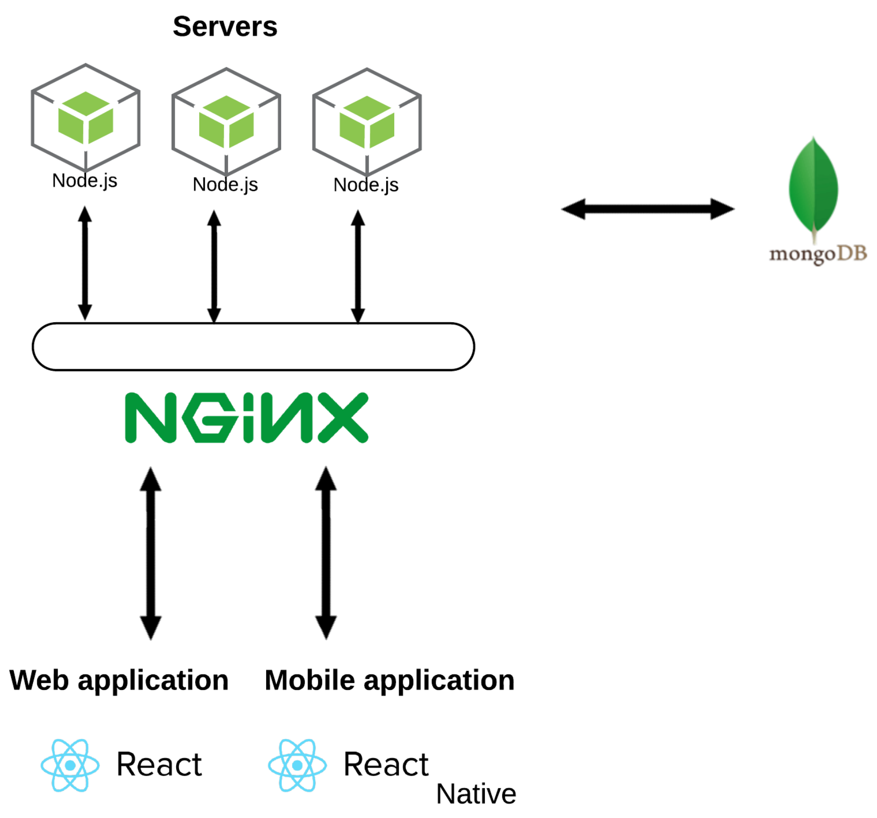


Figure 2.6: Interaction between parts of the system

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 organized 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 collection as the figure 2.7 shows. The arrows are the references, the way in which the arrow is pointing indicates the reference provided.

MongoDb has a different view on data than relational databases. Relational databases are focused on the answers we have when document based databases are more focused on the 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 the GPS pinned application, the main concern is the messages. We need to know, in one request, what is the message and all details about it.

- Message

  The message document contains message text, the description, which is a 150 charactes 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 the 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 to save the image as a base64 string right into the document. The 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. In case of bigger images it is also possible to save them in the collection. MongoDb has a tool which splits big documents into several smaller documents allowing any size to be safely saved in the collection.

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

- User

  The user collection contains standardized user data, authentication fields and a profile picture. It 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 selection in the 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 documents will have their 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 path saved.

- View

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

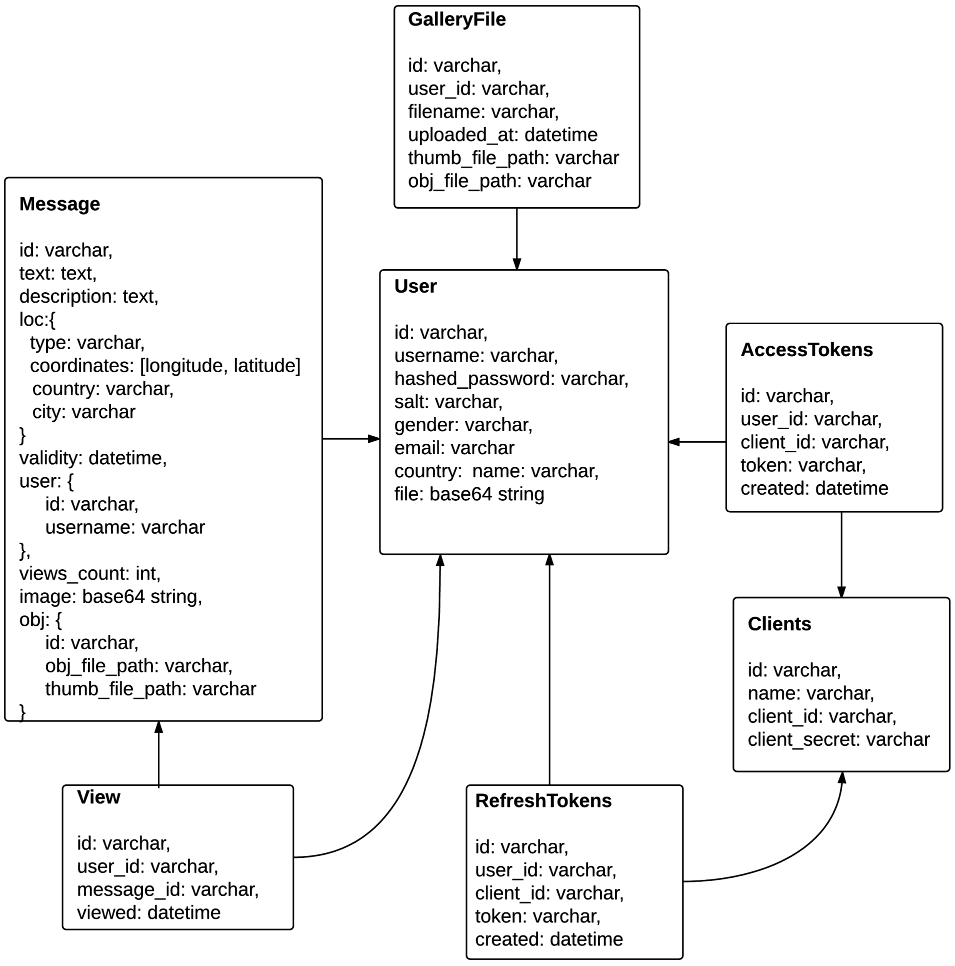


Figure 2.7: MongoDB model of GPS pinned messaging application

- 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

  The clients collection is a list of client application communicating with the API.

2.4 Augmented Reality Feasibility study

After conducting 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. These AR solutions are listed below.

- 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 [4]. This is a 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.

- Unity 3D - Kudan AR plugin

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

  Advantages:

  - SLAM - Simultaneous localization and mapping is a great technique displaying 3d objects. It has already been briefly 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 for my purposes because 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 a 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 issues to browse through. It has been already developed for several years, 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 with full support of all SDK's features. There is a watermark on the camera screen when using AR.

The first option did not seem to be suitable as it would require 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 ways of creating applications such as using javascript wrappers for objective C to create a pure native experience. Moreover this thesis is taking advantage of other different javascript techniques and I would mainly want to focus on possibilities offered by javascript frameworks.

The second option Augmented Reality-wise is a good choice, but it doesn't support location awareness and brings a lot of difficulties when trying to serve it with new resources.

After careful consideration I decided to choose the Wikitude AR SDK with the 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

Realization

3.1 Server

The server is implemented as a REST API Node.js server using the 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 APIs 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 realization 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.

- Statistics

  `GET /stats/countries`

  returns top 10 countries by the total amount of messages

  `GET /stats/cities`

  returns top 10 cities by the total amount of messages

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 the OAuth 2 protocol. It fits our needs well because we will be communicating with our server through an 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 Authorization 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 authorize it's activity.

The authentication flow is the following:

1. The client application sends the user an authorization 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 authorization grant, which is used for further verification of user's identity

3. The Client application uses obtained authorization grant to form another request to the Authorization/Resources server, which contains it's client id, client secret, authorization grant type and the details of authorization 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. The authorization process is complete at this stage. Both the client and user were verified by the server.

5. Once the 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 resources. In case the token has expired the client can always use the refresh token in order to receive a new access token.

There are several different authorization 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 authorization 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, as it is in my 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 Authorization/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.

An example of the request with authorization grant which is described in the step 4 is showed below.

POST `https://oauth.example.com/api/oauth/token`
Request body:

```
{
    "grant_type": "password",
    "client_id": "public_client_id",
    "client_secret": "very_secret_client_secret",
    "username": "mark",
    "password": "secretpassword"
}
```

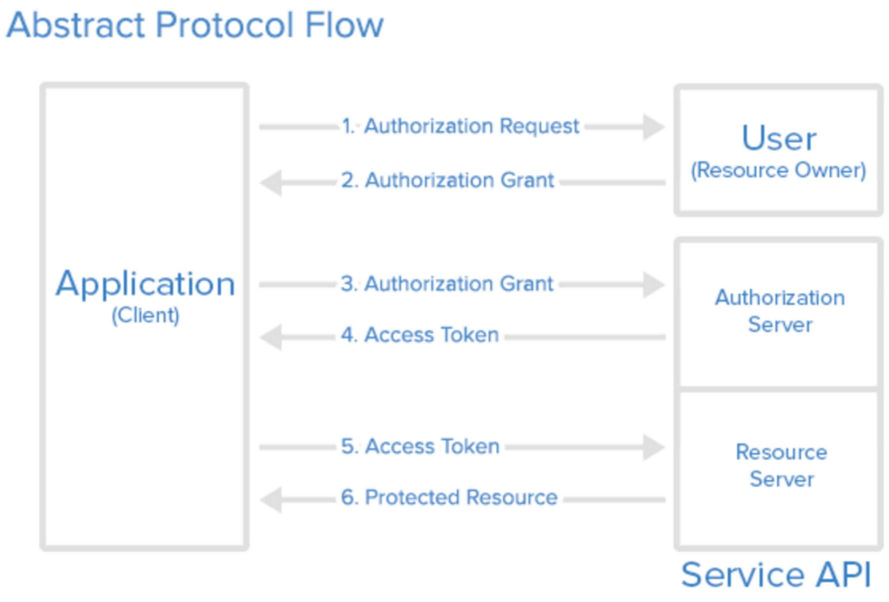


Figure 3.1: OAuth2 authentication protocol flow.
Source: [5]

3.1.3 Files processing

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

The 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 libraries, which parse `multipart/form-data` and works with files. After 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 a smaller size in order to refrain from wasting 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.

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

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

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

This is a simplified example of a 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 \verb`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 a load balancer fulfills 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.

The load balancer I used is called `Nginx`. It is an http server but is also a reverse proxy server and used for load balancing. The Load balancing strategy used on the server is the *least connectivity*. The *Least connectivity* strategy means that the server with the smallest number of connections will receive the 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 shown below. It is preferable to use *ssl* connection for running the server. The example contains the configuration for connecting over *ssl* and without. Lines 3 to 8 of the configuration file define the upstream and the servers which will be processing incoming requests. The load balancing strategy is defined on line 4. Lines 13 and 14 indicate paths to log files. The static content serving is configured on line 16. Line 20 defines the proxy pass, which name matches the name of the upstream. This configuration allows all requests coming to `http://127.0.0.1:8080` to be forwarded to one of the servers 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 [27], 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, considering 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 a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperlogs 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.

```
function addToValidQueue(queue, message){
    var job = queue.create('valid-queue', {
            title: 'unset valid',
            message_id: message.id,
        })
        .delay(message.validity)
        .save(function (err) {
            if (err) {
              log(err);
            }
         });
}
```

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 shown 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 Supervisor

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

Supervisor is a client/server system that allows easy monitoring and manipulations with UNIX processes. It has 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 an example of configuring a server to run. The full configuration file can be found in Appendix E

```
[program:airServer]
directory=/server/directory
command=npm start
autostart=true
stdout_logfile=/logs/server_out.log
stderr_logfile=/logs/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 strategies to define different authenticating techniques. There are around 300 strategies including authenticating through social networks and enterprise generation and more. [31]

- Async

  Async is a powerful library providing functionality for working with asynchronous Javascript. Working with asynchronous functionality get tedious sometimes and this library helps to make it easy. You only need to provide your asynchronous function with a callback which will symbolize 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 the latest versions of Javascript using syntax transformers. This lets developers use new features before the browser support comes out. [33]

- Mongoose unique validator

  This small library is extremely helpful. It is connected as middleware to a model in which unique indexes are defined. The 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 analyzing 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 the '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.

```
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

The `controllers` folder consists of two authorization controllers. The `auth.controller.js` defines authorization strategies of library `passport` such as *BasicStrategy*, *ClientPassword Strategy* and *BearerStrategy*. The `oauth2.controller.js` takes care of token generation, verification, and issuing new tokens on refresh token.

3.2 Mobile Application

3.2.1 Functionality

Mobile application has the following functionality:

- Registration

  On the main view there are options for log in and becoming a member. When the latter is chosen, the user is redirected to the registration view. He fills in his username, email, and password. If no validation errors occur, such as the username or email are already in the database or the password is too short (must be at least 8 symbols), the user is redirected to the map view.

- Login

  The user logs in on the main login view using his username and password. After successful login he is redirected to the map view. If login is unsuccessful the error message is shown.

- Map

  After the login the user is redirected to the map and the messages around him are loaded. The user can tap on different messages to see short descriptions. Each message description also contains a link to see the complete message. When the link is clicked and the user is too far from the message (more than 1-2 meters) an information modal appears informing the user that he is not able to view the message.

- Read a message

  Once the user is close enough to the message, he clicks on the icon on the map, sees the description, and clicks on the link to see the whole message. The message detail view shows up. In case there is an image, it is shown at the top, followed by the message text, the user who added the message and the date of creation. Below that are a number of comments, a comment box and a comments list. Only the 10 most recent comments are shown. There is a link to load more comments, which when clicked, loads an additional 10 comments and appends them to the existing list.

- Comment on a message

  On the message detail the user types in his comment in the comment box, presses post, and immediately sees his comment added as the newest one.

- Side menu

The user can access a side menu by pulling the left side of the screen or tapping on the menu sign in the top left corner. The option in the menu are My Messages, Profile and Logout.

- Add a new message

  The user, while being on the map, can add a new message by tapping Add message in the top right corner. When on the message creation view, the user can choose between text message and object message. If the user chooses a text message, he can attach a picture to it by tapping on "Add image." Next the user is offered a choice of camera or photo gallery. If he chooses the camera, the user can take a picture and it will be attached to the message. Choosing photo gallery allows the user to choose an image from the photo gallery of their phone. If the object message is chosen, the user can choose an object from his gallery. The user taps "Choose an object," and his object gallery is showed to him. In the case his object gallery is empty the user is shown a warning message telling him to go to the website and add objects to his gallery.

- View my messages

  The user pulls the left side of the screen or taps on the menu sign in the top left corner to open the menu. Within the menu is the my messages option, which when pressed redirects the user to the view with his messages. The user sees a list with his messages, next to a short description of each message is a number of views and in the case there are any new comments he will see a small comment icon with a number of new comments. The user can click on each message and view the details. The user can also comment on the message.

- Edit the profile

  The user pulls the left side of the screen or taps on the menu sign in the top left corner to open the menu. There is a profile option, which when pressed redirects the user to his profile. The user can edit his profile by changing his email, adding a country, changing password or uploading a picture.

- Logout

  The user pulls the left side of the screen or taps on the menu sign in the top left corner to open the menu. There is a logout option, which when pressed logs the user out and redirects him to the login view.

The screenshots of the application with description are available in the Appendix C.

3.2.2 Redux

Redux is a predictable state container for Javascript applications. It is a small library which helps control and structure an 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 shown 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 case a 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 actions when presentational components receive user interaction. They also take care of navigation 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, and 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.

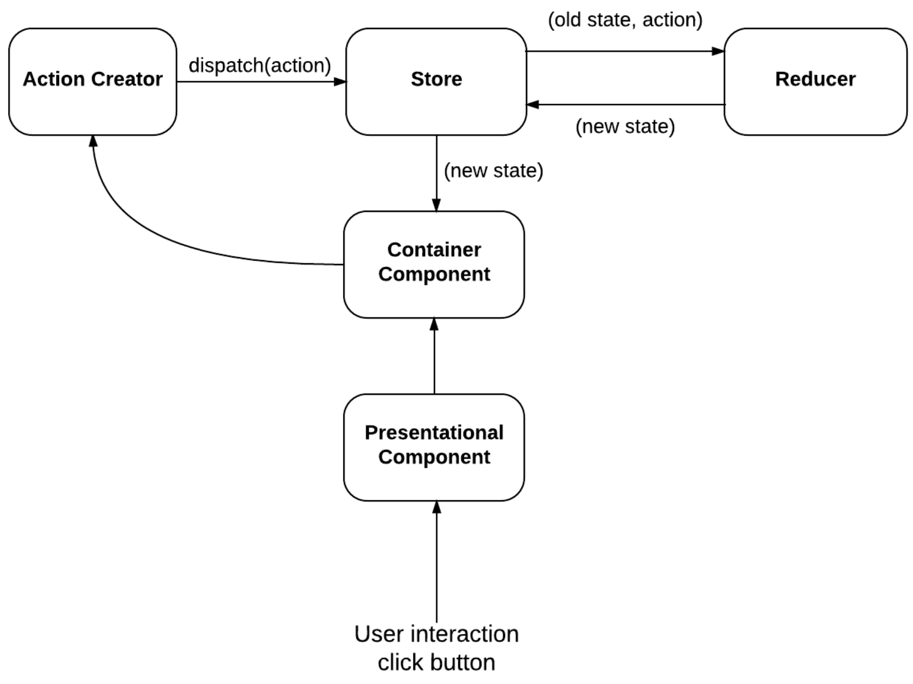


Figure 3.3: Redux structure. Unidirectional dataflow.

3.2.3 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 utilized. Redux has different middleware tools written to take care of that. I chose `redux-sagas` middleware.

Redux-saga is a middleware tool for taking care of asynchronous actions such as calling the 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 [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 then 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 midlleware and any other middleware goes in between "action creator" and "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.

3.2.4 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 summarize the whole process in the end. The following setup is for the application which would serve only one purpose: to create 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 and add it to the middleware on line 6. Lines 8 to 16 initialize 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.

```
/** Actions */
export const MESSAGE_POST = 'MESSAGE POST';
export const MESSAGE_POST_SUCCESS = 'MESSAGE POST SUCCESS';
export const MESSAGE_POST_FAILURE = 'MESSAGE POST FAILURE';

/** Action creators */
export function postMessage(data) {
    return {
        type: MESSAGE_POST,
        payload: {
            message: data
        }
    };
}

export function postMessageSuccess(data) {
    return {
        type: MESSAGE_POST_SUCCESS,
        payload: {
            new_message: data
        }
    };
}

export function postMessageFailure(error, action) {
    return {
        type: MESSAGE_POST_FAILURE,
        payload: {
            error: error
        }
    };
}
```

`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.

```
import {
    MESSAGE_POST_SUCCESS,
    MESSAGE_POST_FAILURE
} from "../actions/messages.actions";

const index = (state = [], action) => {
    switch (action.type) {
        case MESSAGE_POST_SUCCESS:
            // place for some processing logic
            return {
              new_message: action.payload
        };
        case MESSAGE_POST_FAILURE:
            return action.payload;
        default:
            return state;
    }
};

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. Thanks 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));
```

```
    }
}

/** watchers */
export function* watchMessagePost() {
    yield* takeEvery(MESSAGE_POST, postMessage);
}
```

`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 25. And finally on line 31 both of these functions are connected with our container. On line 27 I map postMessage action from `messages.actions.js` to the container properties. On the line 21 i make part of the state available to my container to access through its properties. Once I call function from line 27 the 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.postM(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,
                dispatch)
28      };
29  };
30
```

```
CreateMsgContainer = connect(mapStateToProps, mapDispatchToProps)
    (CreateMsgContainer);
module.exports = CreateMsgContainer;
```

There is no example of component since it is just a simple form with a button, the button would have an 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 the initial state null and creates the application, passes the store to the application through properties, and renders the `create.container.js`. The container render function renders `createComponent` which has a form for user interaction. The 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.

3.2.5 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]).

3.2.5.1 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 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 below: `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.

3.2.5.2 Object format

The supported format for 3d files by Wikitude is *wt3*, which is their own 3d format. For now there is no support for programatic conversion which I could have implemented in my web application. This bring a certain overhead. However wikitude provides their own converter which converts *fbx* file to *wt3*. It is free and available at wikitude website [2].

3.2.5.3 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 behavior for the Wikitude. The simulator only throws a warning and is not affected by the memory leak. On the contrary when the application is tested on a 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 earlier because my initial research didn't show any signs of incompatibility of React Native and Wikitude. Furthermore I 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 update the Augmented Reality logic part of the application and replace it with the plugin.

3.2.6 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

  This React Native module allows access to the phone's photo gallery and the camera. Also it sends the photo taken to the camera back to javascript in base64 format. [39]

- react-native-autocomplete

  Neat module, wrapping around native autocomplete module. It shows a dropdown of suggested autocomplete matches as user types. It is also highly configurable, from styles to the actual function *onTyping()*. It provides callback on every event fired by the native module. [40]

- react-native-maps

  The official facebook documentation for React Native offers this map as a better alternative to their own map implementation. This library offers different map manipulations such as controlling map events, tracking location, programmatically changing region, using animated API and many others. [41]

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 Functionality

The main functionality is access through the menu sections: 3D gallery, My Gallery, Upload 3D and Statistics.

- 3D Gallery

  Web application allows user to browse through a common object gallery to choose 3d objects. Each object can be added to user's gallery.

- Upload 3D

  The user can upload his own 3D images in the request format following the guidelines next to the upload form.

- My Gallery

  In this part the user can see which object were added to his gallery.

- Statistics

  The user can view statics for TOP 10 messaging country and city by accessing Statics in the menu. Furthermore the user can see statistics on his messages. How many times each message has been viewed and how many comments was left on each message, when was the last comment.

Screenshots of each part of the web application are found in the Appendix D.

3.3.2 Routing

I used `redux-router` [42] for handling the routing in the web application. It is a routing library for the React which allowed me to create user friendly, pretty urls without hashes, and to use browser history. Since React is a view only framework, the navigation is purely on the client side and 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. 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 unless you specifically created it. 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 [43].

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 redirects 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

3.3.3 Libraries used

Here are a few of the most helpful libraries I came across while working on the web application.

- react-bootstrap

  Famous Bootsrap library wrapped with React Components. As the original Bootstrap library allows easy styling. [44]

- react-masonry-component

  Neat masonry library, which allows easy organizing of objects on the gird. [45]

- react-d3

  D3 chart utilities enabled for React. The library supports various types of charts, for instance: line chart, scatter chart, bar chart and area chart. [46]

3.4 Testing

3.4.1 Unit tests

Unit testing is a type of automated testing focused on testing functionality of separate units of system in isolation. Targets of unit testing could be functions, React components and classes. Unit tests are tests closest to the code itself, they do not test any business logic but pure functionality. To make sure the tested code is isolated a developer mocks subcomponents and services used by the class/component he is testing. [47] Unit tests help during development. If they are frequently written the developer can make bold changes to the code and easily check whether any of the previous functionality is broken by simply running the tests. Tests should be maintained and changed with the growing code base to make sure they have a solid code coverage.

3.4.1.1 Framework

When looking for a testing framework my main concern was to find one framework I could easily use for all the technologies involved in my project. I needed a framework with support for testing node, easily calling API endpoints, traversing React tree, accessing React's components, and being able to parse, and deal with React Native components.

There is a plethora of javascript testing frameworks. However, not that many of them support or have a smooth way of working with React or React Native. The main issue with React Native is that it does not render html but native components. Therefore, each component needs to be mocked separately. While some like `Text` are easy to mock others like `Animate` leave you puzzled. Therefore, a big part of testing React Native is mocking which should not be the focus of the testing process. The developer needs to focus on the tests, not on mocks. Until recently the best way to test React and React Native was `Jest` [48]. It is a very good choice for testing since it mocks everything automatically and the developer can focus on writing tests rather than creating mocks for components. I tried Jest. However, I was dissatisfied with its speed and after reading about possible other issues that might arise such as mocking everything, event the libraries you do not want to have mocked, I decided to look for another solution. [47]

The next testing framework I tried was Mocha [49]. Mocha is not created for testing React but there are complimentary libraries which help with that. For instance `enzyme` [50] was designed specifically for this purpose.

`Enzyme` is a javascript testing utility which allows traversing React DOM, assert and manipulate with React's components with ease. Moreover, recently `react-native-mock` [51] came out, which is a mocker for all the React Native components written by Leland Richardson. Combining `react-native-mock`, `mocha` and `enzyme` provides a powerful setup with easy and elegant testing process.

3.4.1.2 Execution

Before running unit tests first of all make sure all the dev dependencies are installed. Test dependencies are included for the development environment only. That saves time when installing the project and disk space.

```
 npm install --only=dev
```

To run unit tests in any of my applications simply run the following in the command line.

```
npm test
```

3.4.2 Redux testing

Unit testing on its own is not enough to test the mobile and web applications. It is also important to test the data flow system I am using - Redux. Redux tests can be broken down into several categories:

- Action Tests

  These tests will make sure the action creators return the correct actions upon being called.

- Reducer Tests

  These test take care of verifying the correct transition between states. The test checks if the reducer returns the correct state upon receiving dispatched action.

- Middleware Tests

  The middleware in Redux flow is responsible for catching the dispatched action before it reaches reducer, performing some logic and then dispatching another action based on result of the performed logic. These tests are very important for both web and mobile applications since they are testing communication and data retrieval.

Redux testing will be performed using `mocha`, `enzyme` and `react-native-mock`.

3.4.2.1 Execution

These tests are run the same way as the unit tests.

Conclusion

The main task of using only Javascript technologies for creating a messaging system with mobile application, web application and a server was accomplished. Both non functional requirements and functional requirements have been fulfilled. The system is highly performant due to the chosen technologies: `node.js`, `React` and `React Native`. It would easily recover in case of a crash. External processes such as `supervisor` make sure the server and the queue consumers are restarted in case of failure. The server API is well documented with documentation easily accessible from the index page. Moreover, the system is scalable and is ready for high request load. Load balancer is configured to serve that purpose. In case of the server overloading it only takes a minute to add a new server to load balancer's configuration to scale the system. The system is easily extendable due to the nature of chosen implementation technologies.

It must be also mentioned that Javascript is a very powerful technology, which is underestimated by many. It is flexible and has a lively community with many contributors. React has already transformed the way web front-end is made. React Native, an amazing creation of Facebook engineers, is definitely worth while and has a big future ahead of it. It revolutionises the way mobile applications are created and gives access to mobile development to wider audience. And last but not the least is `node.js` which makes writting server side code pleasant.

To conclude I would like to say that I am very fortunate for having an opportunity to explore these amazing technologies and to have a chance to create a solid, highly functional system using them.

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

[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] Breshenan, 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.

[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/`

[36] Elouafi, Y. *Redux-saga documentation.* [Accessed: 20 Mar 2016]. Available from: `http://yelouafi.github.io/redux-saga/docs`

[37] Wiktude. wikitude-sdk-basic-projects. `https://github.com/Wikitude/wikitude-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] Shilling, M. react-native-image-picker. `https://github.com/marcshilling/react-native-image-picker`, 2016, [Accessed: 20 Apr 2016].

[40] Borja, E. rMLPAutoCompleteTextField. `https://github.com/EddyBorja/MLPAutoCompleteTextField`, 2016, [Accessed: 20 Apr 2016].

[41] Richardson, L. react-native-maps. `https://github.com/lelandrichardson/react-native-maps`, 2016, [Accessed: 20 Apr 2016].

[42] Community, R. react-router. `https://github.com/reactjs/react-router`, 2016, [Accessed: 20 Apr 2016].

[43] Nagayama, K. Deprecating our AJAX crawling scheme. `https://webmasters.googleblog.com/2015/10/deprecating-our-ajax-crawling-scheme.html`, 2015, [Accessed: 20 Apr 2016].

[44] react bootstrap. React Bootstrap. `https://github.com/react-bootstrap/react-bootstrap/`, 2016, [Accessed: 24 Apr 2016].

[45] Vullum, E. L. React Masonry Component. `https://github.com/eiriklv/react-masonry-component`, 2016, [Accessed: 24 Apr 2016].

[46] Bullington, E. S. React D3. `https://github.com/esbullington/react-d3`, 2016, [Accessed: 24 Apr 2016].

[47] Chen, Y.-J. Unit Testing a Redux App. `https://www.codementor.io/reactjs/tutorial/redux-unit-test-mocha-mocking` [Accessed: 13 Feb 2016], December 2015.

[48] Facebook. *Jest.* [Accessed: 22 Feb 2016]. Available from: `https://facebook.github.io/jest/`

[49] Mochajs.org. *Mocha.* [Accessed: 22 Feb 2016]. Available from: `https://mochajs.org/`

[50] Richardson, L. enzyme. `https://github.com/airbnb/enzyme`, 2016, [Accessed: 22 Apr 2016].

[51] Richardson, L. react-native-mock. `https://github.com/lelandrichardson/react-native-mock`, 2016, [Accessed: 22 Apr 2016].

Appendix **A**

Acronyms

AR Augmented Reality

DOM Extensible markup language

JS Javascript

MB Megabyte

POI Point of Interest

SDK Software Development Kit

SQL Structured Query Language

3D Three dimensional

Appendix B

Installation guide

B.1 Server

B.1.1 Dependencies

- Npm
- Redis
- MongoDb
- Supervisor
- Nginx

Start installing dependencies by running the following command.

```
npm install
```

The rest of dependecies are available for download at their official websites. Configuration files for `Supervisor` and `Nginx` are provided in folder `configuration` on the enclosed CD.

B.1.2 Execution

Start the Redis server.

```
  redis-server
```

Start the MongoDb in the server directory. It will be using directory data, in case u want mongo to use another directory, just specify it as the command argument.

```
  mongod --dbpath=data
```

Now you configure supervisor, by adding both consumers and a server in the configuration. Run supervisor and specify the configuration file path. You can either configure your commands to run at the start or run the command line utility `supervisorctl` and start them from there. If you used provided supervisor config the commands will run on start.

```
    supervisord --configuration=FILE
```

The server is ready to take requests now. It is running on `http://localhost:3000`

B.2 Mobile application

B.2.1 Dependencies

- XCode
- Npm
- Rnpm

B.2.2 Execution

Install XCode from the official website or through the AppStore.

Start installing dependencies by running the following command.

```
npm install
```

Link the external dependencies to the Objective C project in XCode.

```
rnpm link
```

Open `AppDelegate.m` from the `ios` folder and make sure there is a correct address set for the *jsCodeLocation*.

There is a folder `/app/config` which has configuration for the server address. Set the correct address of the server in there.

Compile the project. You can either run it on the in-built simulator or on a connected device.

B.3 Web application

B.3.1 Dependencies

- Npm

B.3.2 Execution

Start installing dependencies by running the following command.

```
npm install
```

The web application needs to have a server started to run. In folder `server` in the web application implementation run the following command.

```
npm run
```

An express server is ready to take requests now.
In your browser navigate to `http://localhost:3001`. You should be able to interact with the web application now.

There is a folder `/app/config` which has configuration for the server address. In case the web and the server will not be running on the same machine that configuration will have to be replaced with the actual server address.

Appendix C

Mobile application screenshots

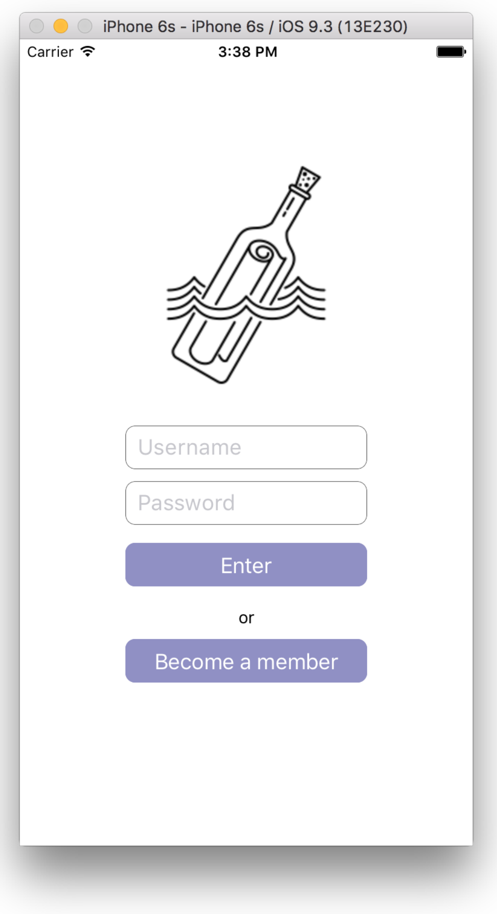


Figure C.1: Mobile application. Login

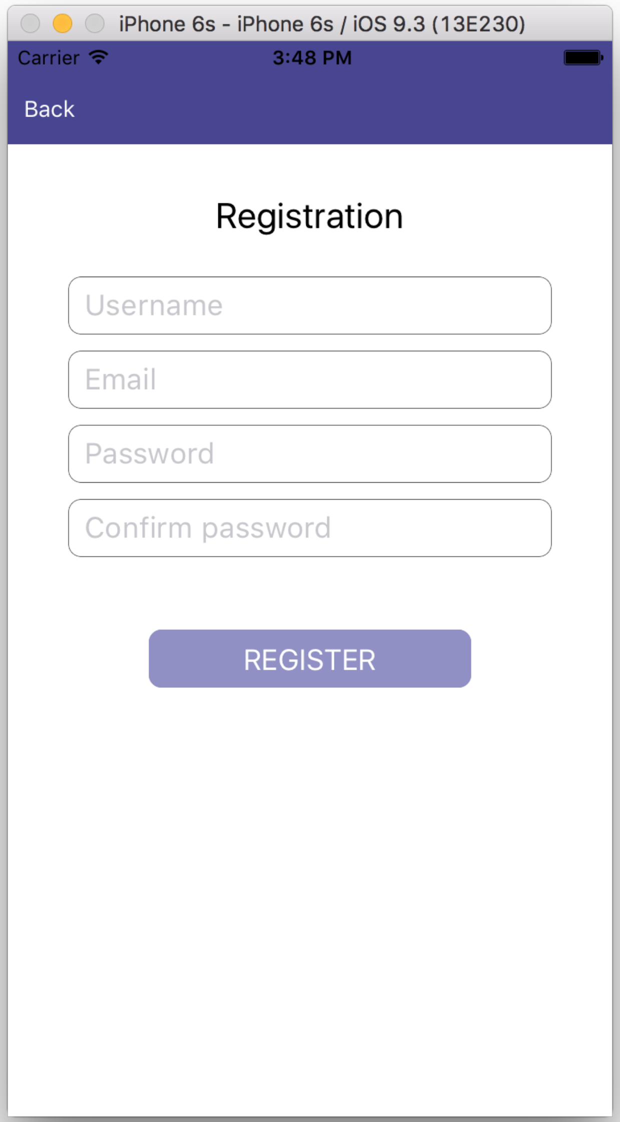
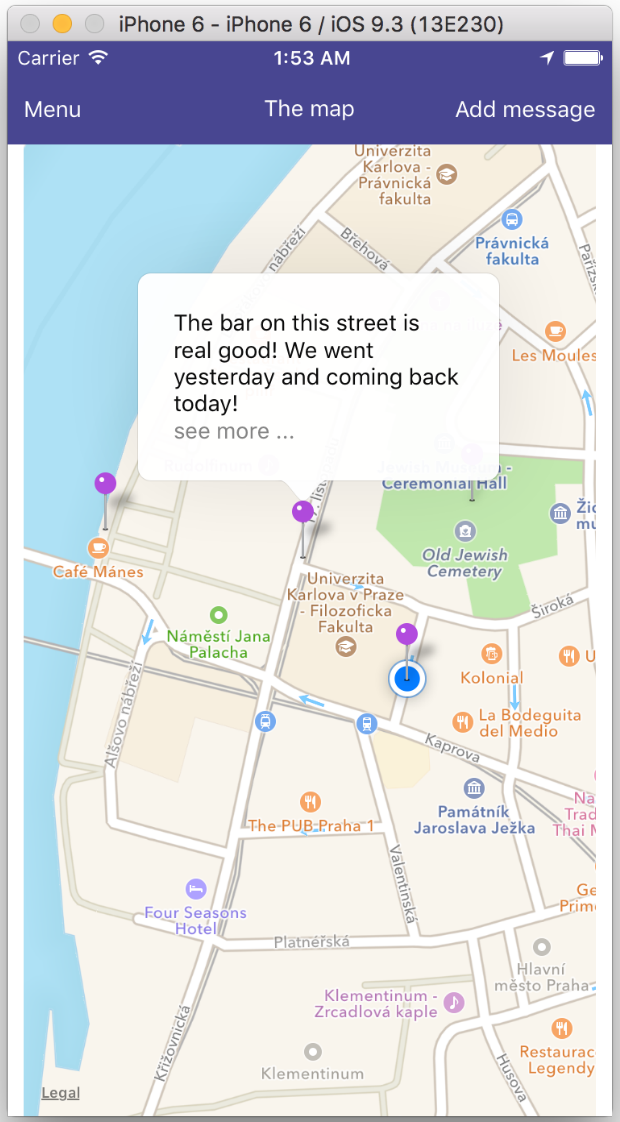


Figure C.2: Mobile application. Registration



The bar on this street is real good! We went yesterday and coming back today!
see more ...

Figure C.3: Mobile application. Messages map

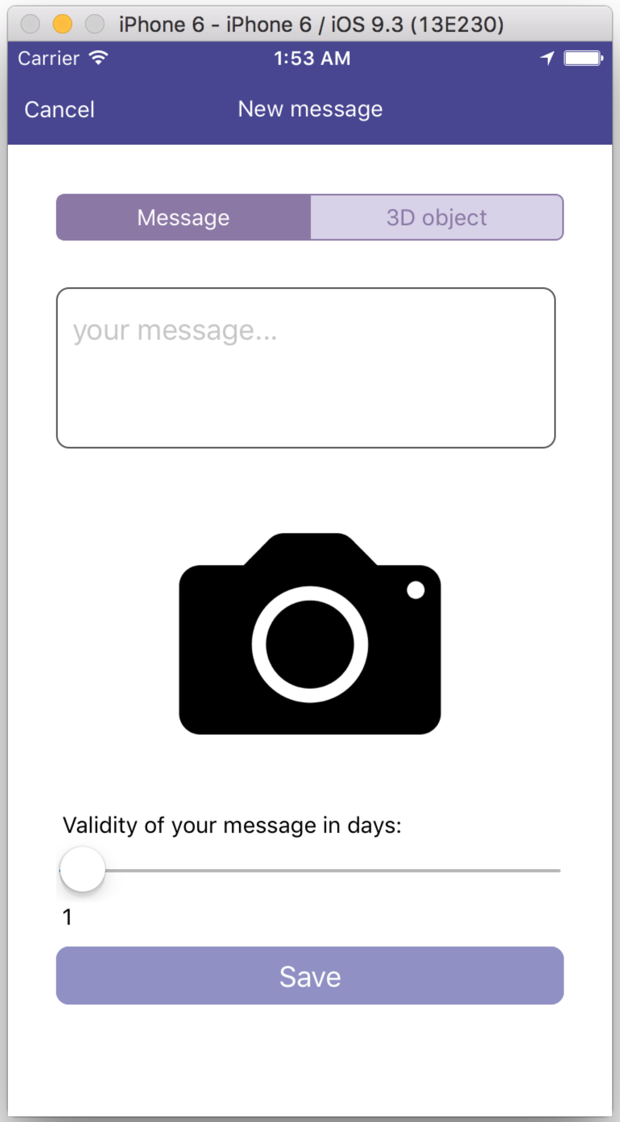

iPhone 6 - iPhone 6 / iOS 9.3 (13E230)
Carrier
1:53 AM
Cancel
New message
Message
3D object
your message...
Validity of your message in days:
1
Save

Figure C.4: Mobile application. Adding a text message

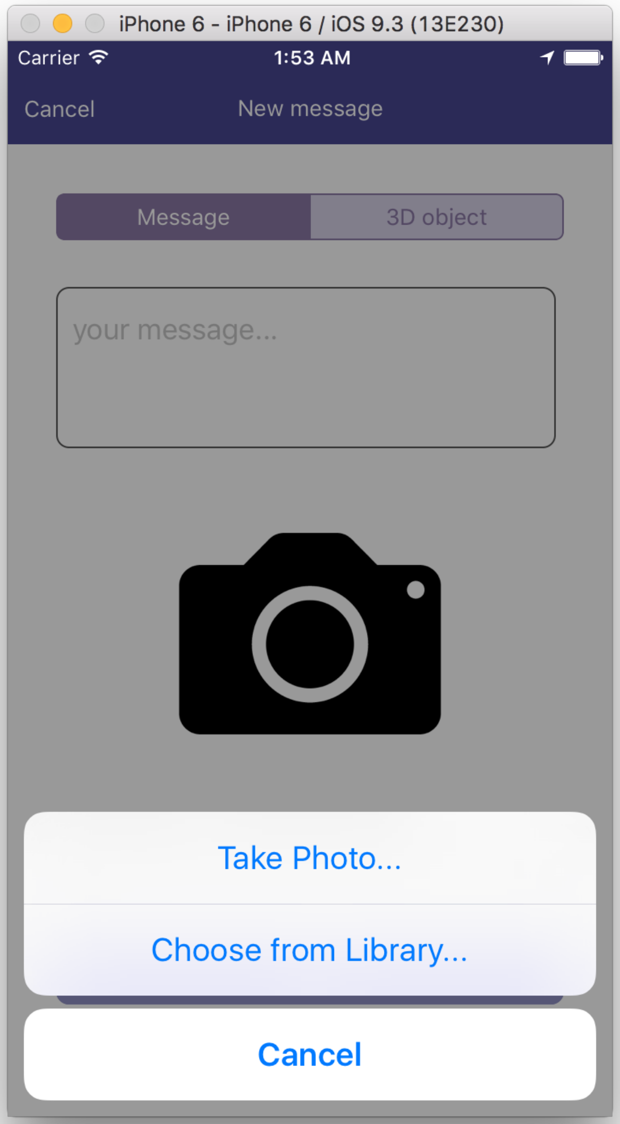
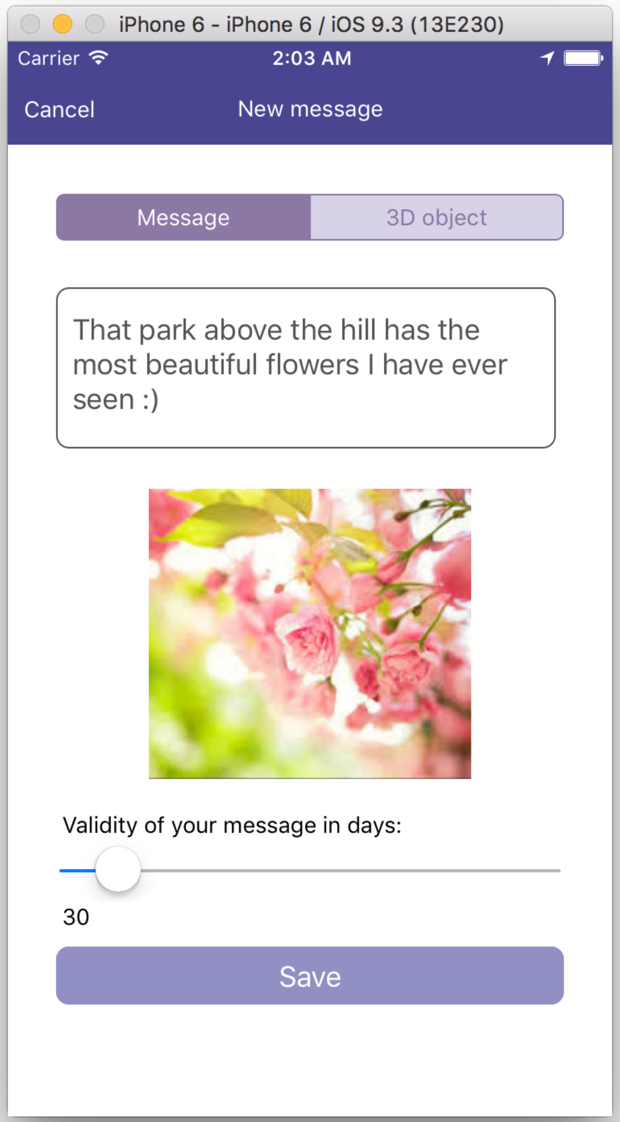


Figure C.5: Mobile application. Adding image to text message



That park above the hill has the most beautiful flowers I have ever seen :)

Validity of your message in days:

30

Figure C.6: Mobile application. Message with image and text

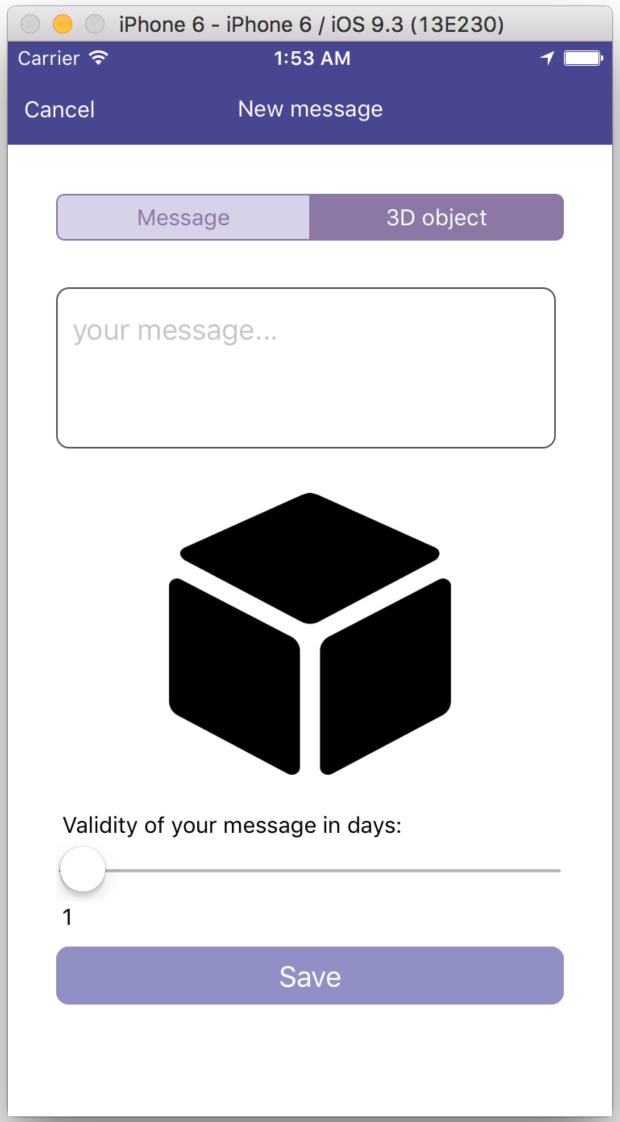

iPhone 6 - iPhone 6 / iOS 9.3 (13E230)
Carrier
1:53 AM
Cancel
New message
Message
3D object
your message...
Validity of your message in days:
1
Save

Figure C.7: Mobile application. 3D Message option chosen

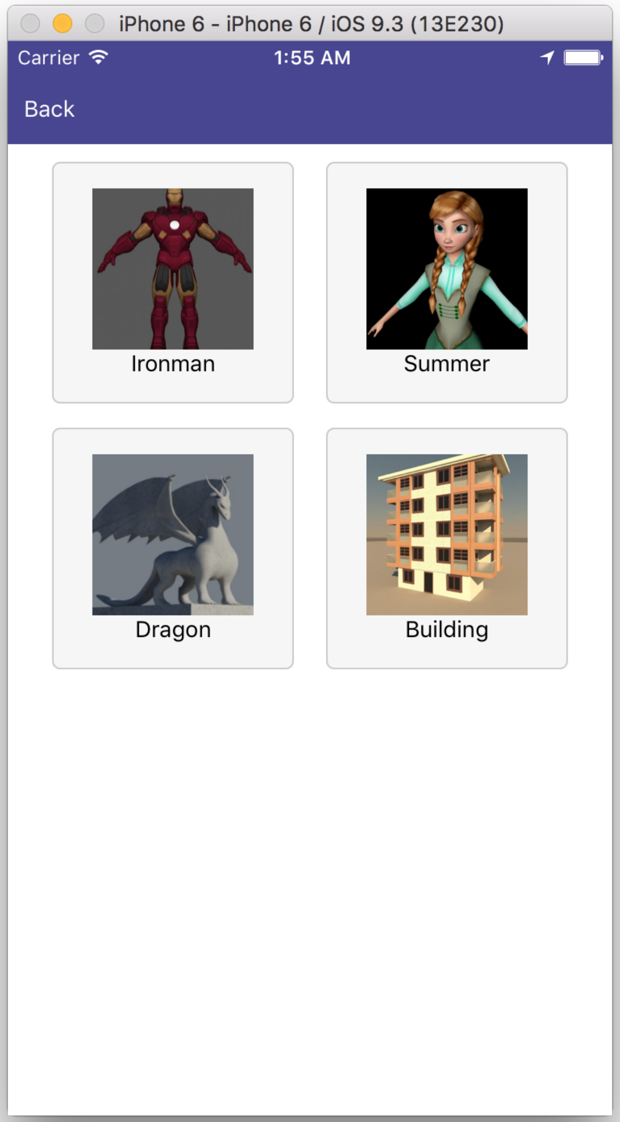
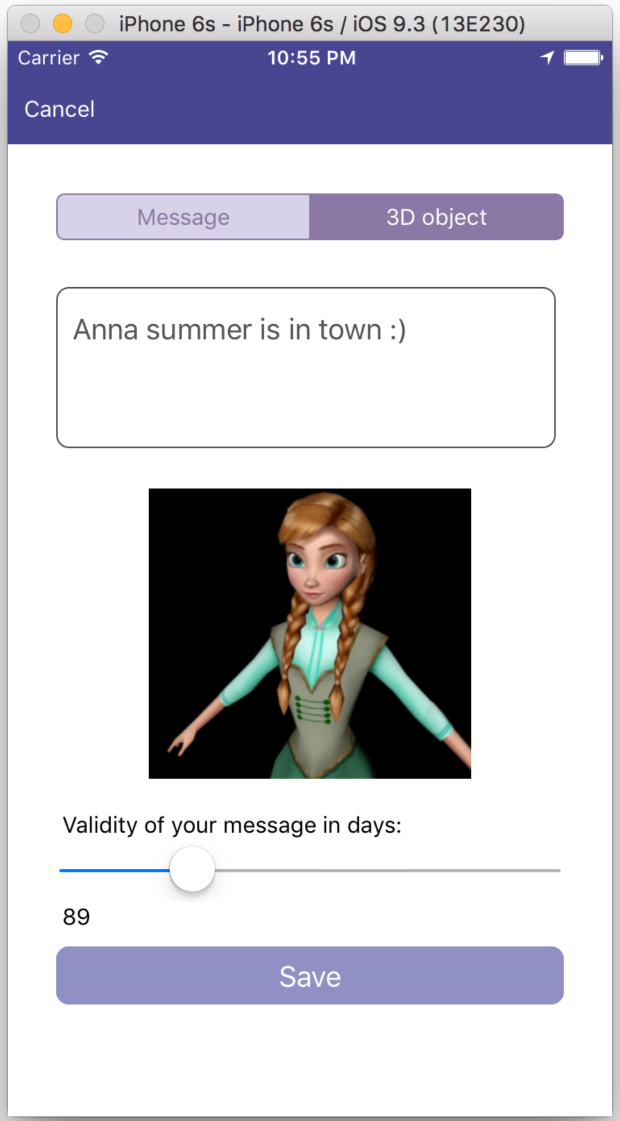


Figure C.8: Mobile application. User's object gallery. It is accessible when attaching an object to the message.



Anna summer is in town :)

Validity of your message in days:

89

Figure C.9: Mobile application. 3D Message thumbnail preview

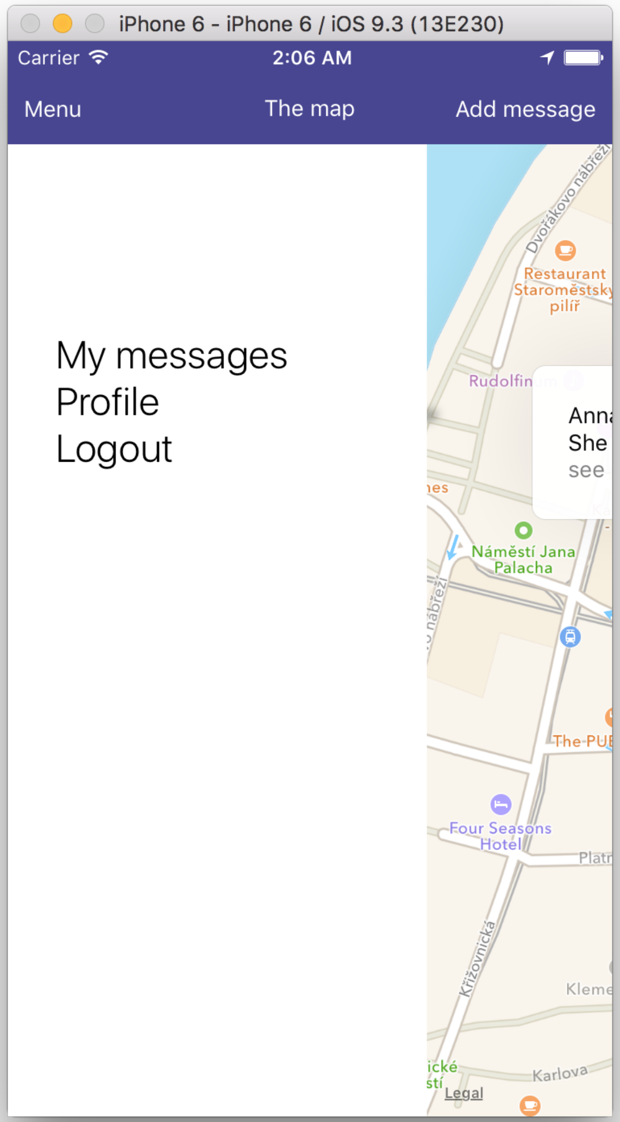
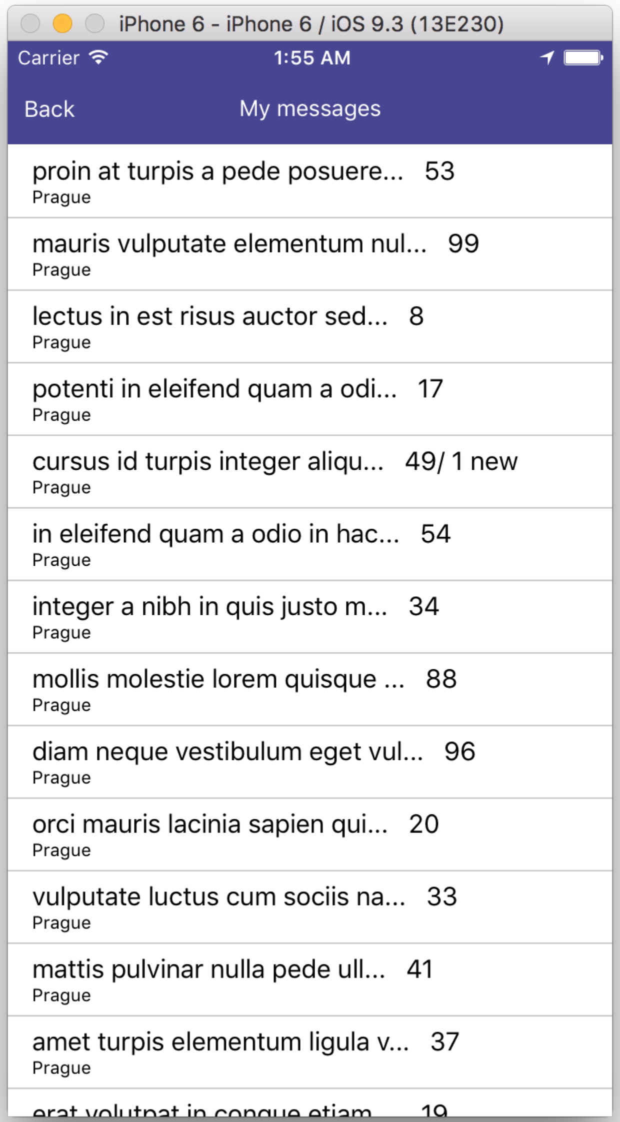


Figure C.10: Mobile application. Menu



iPhone 6 - iPhone 6 / iOS 9.3 (13E230)

Carrier 1:55 AM

Back My messages

proin at turpis a pede posuere... 53
Prague

mauris vulputate elementum nul... 99
Prague

lectus in est risus auctor sed... 8
Prague

potenti in eleifend quam a odi... 17
Prague

cursus id turpis integer aliqu... 49/ 1 new
Prague

in eleifend quam a odio in hac... 54
Prague

integer a nibh in quis justo m... 34
Prague

mollis molestie lorem quisque ... 88
Prague

diam neque vestibulum eget vul... 96
Prague

orci mauris lacinia sapien qui... 20
Prague

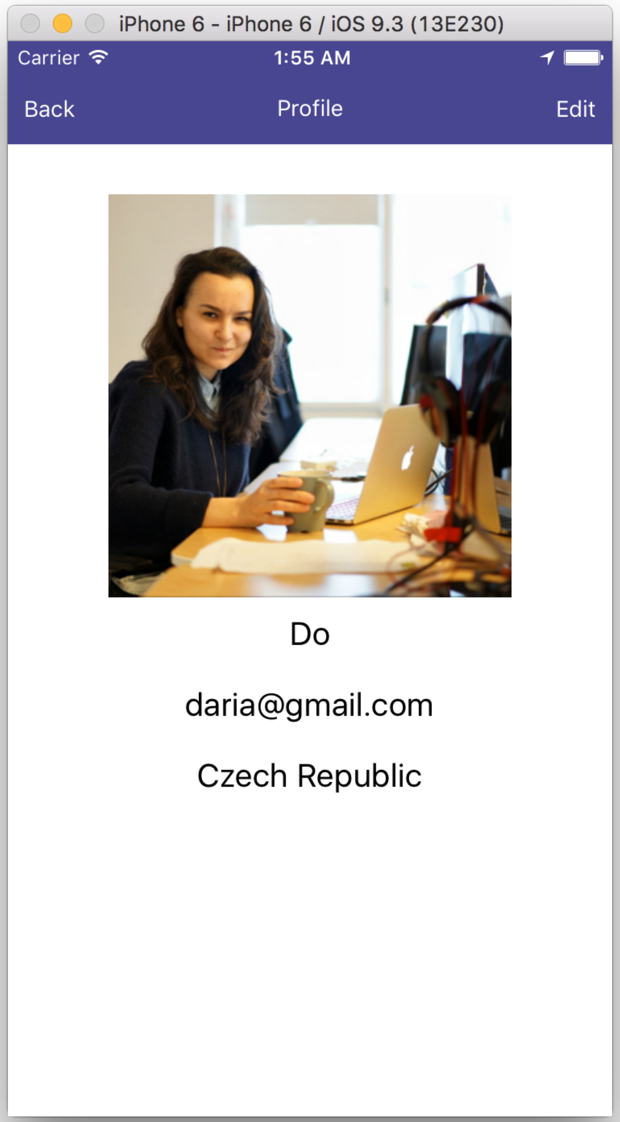
vulputate luctus cum sociis na... 33
Prague

mattis pulvinar nulla pede ull... 41
Prague

amet turpis elementum ligula v... 37
Prague

erat volutpat in congue etiam... 19

Figure C.11: Mobile application. My messages

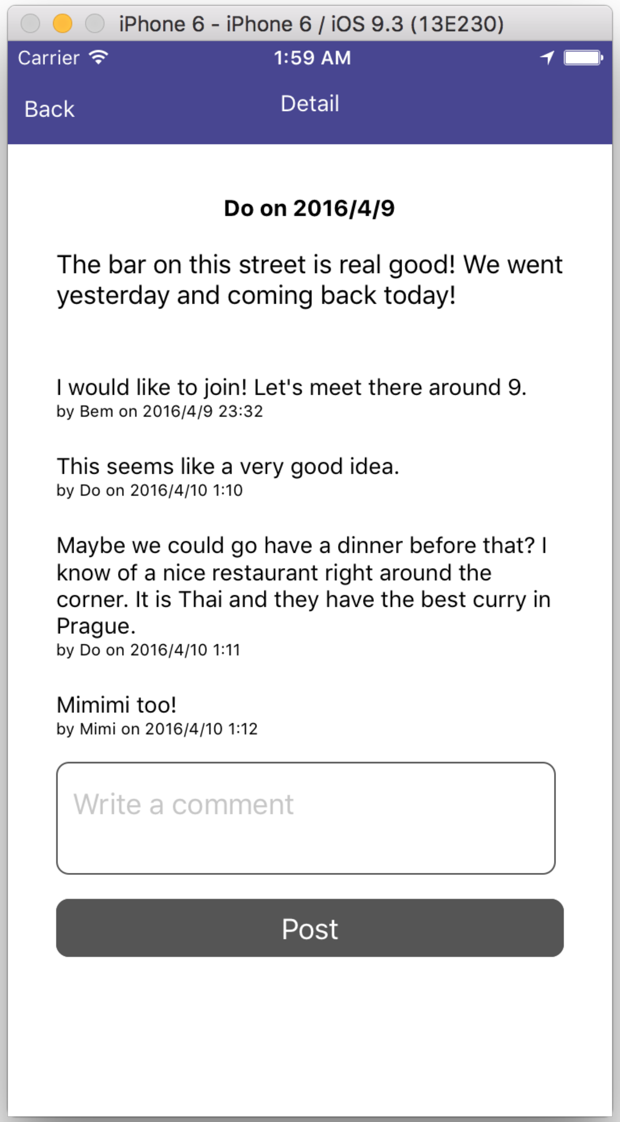


Do

daria@gmail.com

Czech Republic

Figure C.12: Mobile application. Profile



Do on 2016/4/9

The bar on this street is real good! We went yesterday and coming back today!

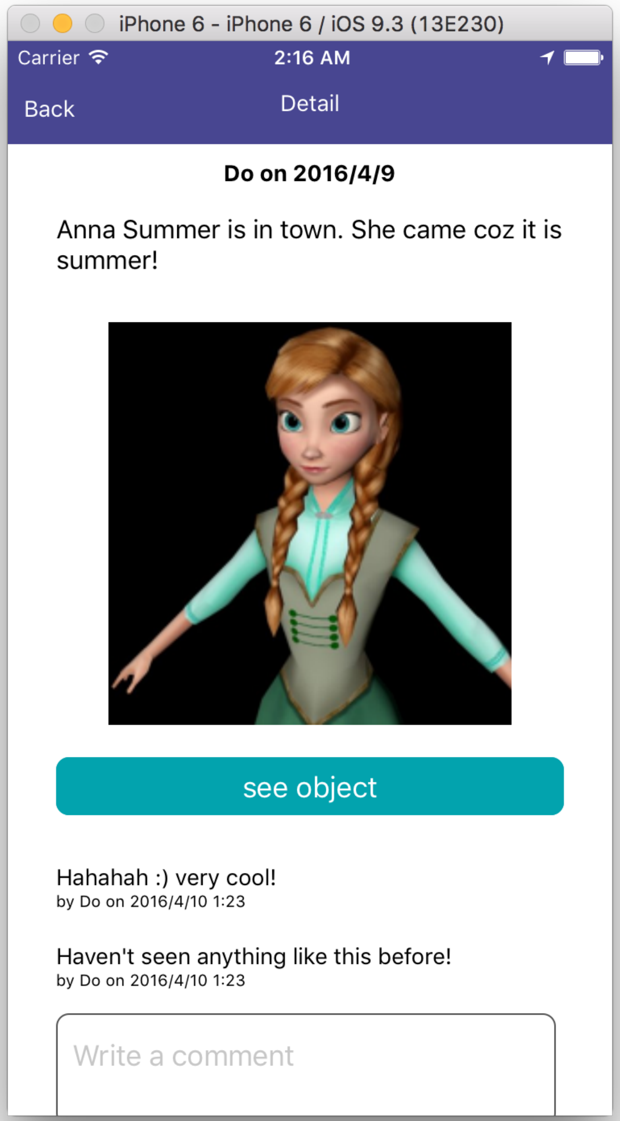
I would like to join! Let's meet there around 9.
by Bem on 2016/4/9 23:32

This seems like a very good idea.
by Do on 2016/4/10 1:10

Maybe we could go have a dinner before that? I know of a nice restaurant right around the corner. It is Thai and they have the best curry in Prague.
by Do on 2016/4/10 1:11

Mimimi too!
by Mimi on 2016/4/10 1:12

Figure C.13: Mobile application. Message detail



Do on 2016/4/9

Anna Summer is in town. She came coz it is summer!

Hahahah :) very cool!
by Do on 2016/4/10 1:23

Haven't seen anything like this before!
by Do on 2016/4/10 1:23

Figure C.14: Mobile application. 3D Message detail. Augmented Reality starts when "see object" button is tapped.

Appendix D

Web application screens

Web application is made as a complimentary portal mainly to allow user to choose 3d objects from the common gallery, add these obejcts to his gallery or upload his own 3d objects, see information about his messages and view statistics by country and city.

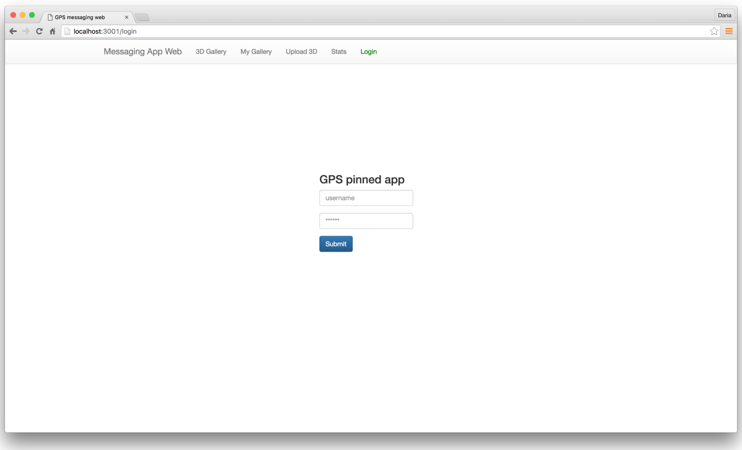
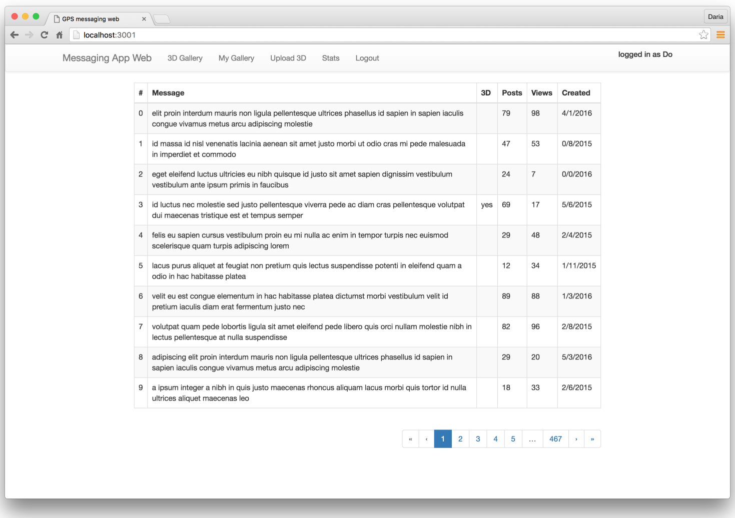


Figure D.1: Web application. Login



#	Message	3D	Posts	Views	Created
0	elit proin interdum mauris non ligula pellentesque ultrices phasellus id sapien in sapien iaculis congue vivamus metus arcu adipiscing molestie		79	98	4/1/2016
1	id massa id nisl venenatis lacinia aenean sit amet justo morbi ut odio cras mi pede malesuada in imperdiet et commodo		47	53	0/8/2015
2	eget eleifend luctus ultricies eu nibh quisque id justo sit amet sapien dignissim vestibulum vestibulum ante ipsum primis in faucibus		24	7	0/0/2016
3	id luctus nec molestie sed justo pellentesque viverra pede ac diam cras pellentesque volutpat dui maecenas tristique est et tempus semper	yes	69	17	5/6/2015
4	felis eu sapien cursus vestibulum proin eu mi nulla ac enim in tempor turpis nec euismod scelerisque quam turpis adipiscing lorem		29	48	2/4/2015
5	lacus purus aliquet at feugiat non pretium quis lectus suspendisse potenti in eleifend quam a odio in hac habitasse platea		12	34	1/11/2015
6	velit eu est congue elementum in hac habitasse platea dictumst morbi vestibulum velit id pretium iaculis diam erat fermentum justo nec		89	88	1/3/2016
7	volutpat quam pede lobortis ligula sit amet eleifend pede libero quis orci nullam molestie nibh in lectus pellentesque at nulla suspendisse		82	96	2/8/2015
8	adipiscing elit proin interdum mauris non ligula pellentesque ultrices phasellus id sapien in sapien iaculis congue vivamus metus arcu adipiscing molestie		29	20	5/3/2016
9	a ipsum integer a nibh in quis justo maecenas rhoncus aliquam lacus morbi quis tortor id nulla ultrices aliquet maecenas leo		18	33	2/6/2015

Figure D.2: Web application. Index page with messages

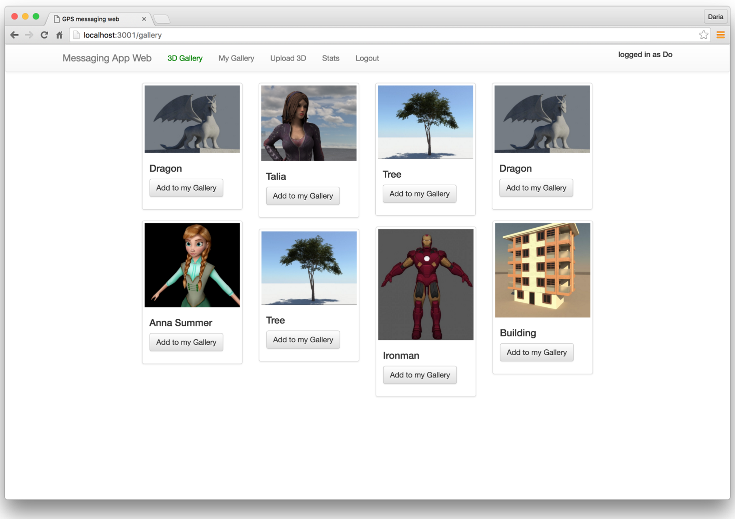


Figure D.3: Web application. 3D gallery with objects

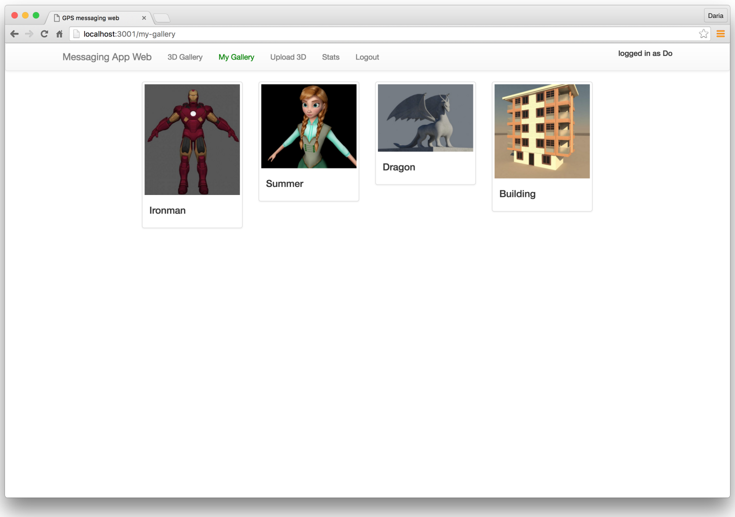


Figure D.4: Web application. User's 3d gallery with objects

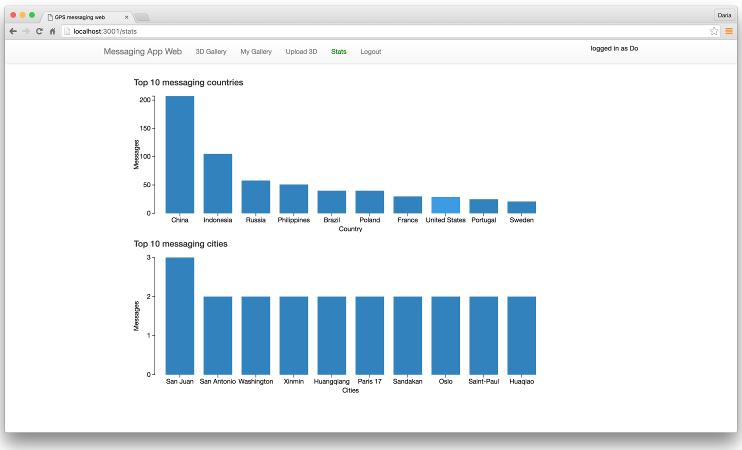


Figure D.5: Web application. Statistics

Appendix E

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
```

```

[program:validConsumer]
directory=/kue
command=node validConsumer.js
autostart=true
stdout_logfile=/logs/supervisor/validConsumer_out.log
stderr_logfile=/logs/supervisor/validConsumer_err.log
```

Appendix F

Express server for React routing

```
// required for converting JSX syntax
require("node-jsx").install();
require("babel-core/register")({"presets": ["es2015"]});
var express = require('express');
var path = require('path');
var app = express();

// serving static files
app.use(express.static(path.join(__dirname, 'public')));

app.get('*', function (request, response){
  response.sendFile(path.resolve(__dirname, 'public', 'index.html'))
});

// development error handler - will print stacktrace
if (app.get('env') === 'development') {
  app.use(function(err, req, res, next) {
    res.status(err.status || 500);
    res.render('error', {
      message: err.message,
      error: err
    });
  });
}

// production error handler - no stacktraces leaked to user
app.use(function(err, req, res, next) {
  res.status(err.status || 500);
  res.render('error', {
    message: err.message,
    error: {}
  });
});

module.exports = app;
```

Appendix G

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

```
//  ViewController.h

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@property (nonatomic, assign) NSString *objSrc;

@end
```

```
//  ViewController.m

#import "ViewController.h"

#import "View.h"
#import <WikitudeSDK/WikitudeSDK.h>
/* Wikitude SDK debugging */
#import <WikitudeSDK/WTArchitectViewDebugDelegate.h>

@interface ViewController () <WTArchitectViewDelegate,
    WTArchitectViewDebugDelegate>

```

```
/* Add a strong property to the main Wikitude SDK component, the
    WTArchitectView */
@property (nonatomic, strong) WTArchitectView                 *
    architectView;

/* And keep a weak property to the navigation object which
    represents the loading status of your Architect World */
@property (nonatomic, weak) WTNavigation                      *
    architectWorldNavigation;

@end

@implementation ViewController


- (void)dealloc
{
  /* Remove this view controller from the default Notification
      Center so that it can be released properly */

  [[NSNotificationCenter defaultCenter] removeObserver:self];
}

- (void)loadView
{
  View *_view = [[View alloc] init];
  self.view = _view;
}


- (void)viewDidLoad {
  [super viewDidLoad];
  // Do any additional setup after loading the view, typically
      from a nib.

  /* It might be the case that the device which is running the
      application does not fulfil all Wikitude SDK hardware
      requirements.
   To check for this and handle the situation properly, use the -
       isDeviceSupportedForRequiredFeatures:error class method.

   Required features specify in more detail what your Architect
       World intends to do. Depending on your intentions, more or
        less devices might be supported.
   e.g. an iPod Touch is missing some hardware components so that
        Geo augmented reality does not work, but 2D tracking does
       .

   NOTE: On iOS, an unsupported device might be an iPhone 3GS for
        image recognition or an iPod Touch 4th generation for Geo
        augmented reality.
   */
  NSError *deviceSupportError = nil;

```

```
51   if ( [WTArchitectView isDeviceSupportedForRequiredFeatures:
         WTFeature_Geo error:&deviceSupportError] ) {
52
53     /* Standard WTArchitectView object creation and initial
          configuration */
54     self.architectView = [[WTArchitectView alloc] initWithFrame:
          CGRectZero motionManager:nil];
55     self.architectView.delegate = self;
56     self.architectView.debugDelegate = self;
57
58     /* Use the -setLicenseKey method to unlock all Wikitude SDK
          features that you bought with your license. */
59     [self.architectView setLicenseKey:@"YOUR_KEY"];
60
61     /* The Architect World can be loaded independently from the
          WTArchitectView rendering.
62
63      NOTE: The architectWorldNavigation property is assigned at
           this point. The navigation object is valid until another
            Architect World is loaded.
64      */
65
66     NSURL *baseURL = [NSURL URLWithString:(@"http
          ://192.168.0.3:3000/ArchitectWorld/index.html")];
67     self.architectWorldNavigation = [self.architectView
          loadArchitectWorldFromURL:baseURL withRequiredFeatures:
          WTFeature_Geo];
68     /* Because the WTArchitectView does some OpenGL rendering,
          frame updates have to be suspended and resumend when the
          application changes it's active state.
69      Here, UIApplication notifications are used to respond to the
            active state changes.
70
71      NOTE: Since the application will resign active even when an
           UIAlert is shown, some special handling is implemented
           in the UIApplicationDidBecomeActiveNotification.
72      */
73     [[NSNotificationCenter defaultCenter] addObserverForName:
          UIApplicationDidBecomeActiveNotification object:nil queue
          :[NSOperationQueue mainQueue] usingBlock:^(NSNotification
           *note) {
74
75       /* When the application starts for the first time, several
            UIAlert's might be shown to ask the user for camera and
            /or GPS access.
76        Because the WTArchitectView is paused when the application
              resigns active (See line 86), also Architect
             JavaScript evaluation is interrupted.
77        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.
78        This loading state/interruption can be detected using the
             navigation object that was returned from the -
             loadArchitectWorldFromURL:withRequiredFeatures method.
          */
         if (self.architectWorldNavigation.wasInterrupted) {
           [self.architectView reloadArchitectWorld];
         }

         /* Standard WTArchitectView rendering resuming after the
            application becomes active again */
         [self startWikitudeSDKRendering];
       }];
       [[NSNotificationCenter defaultCenter] addObserverForName:
          UIApplicationWillResignActiveNotification object:nil
          queue:[NSOperationQueue mainQueue] usingBlock:^(
          NSNotification *note) {

         /* Standard WTArchitectView rendering suspension when the
            application resignes active */
         [self stopWikitudeSDKRendering];
       }];

       /* Standard subview handling using Autolayout */
       [self.view addSubview:self.architectView];
       self.architectView.translatesAutoresizingMaskIntoConstraints
          = NO;

       NSDictionary *views = NSDictionaryOfVariableBindings(
          _architectView);
       [self.view addConstraints: [NSLayoutConstraint
          constraintsWithVisualFormat:@"|[_architectView]|" options
          :0 metrics:nil views:views] ];
       [self.view addConstraints: [NSLayoutConstraint
          constraintsWithVisualFormat:@"V:|[_architectView]|"
          options:0 metrics:nil views:views] ];
     }
     else {
       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]);
     }
  }

  #pragma mark - View Lifecycle
  - (void)viewWillAppear:(BOOL)animated {

     [super viewWillAppear:animated];
     View* test_view = [self view];
     NSString *src = [NSString stringWithFormat:@"%@%@%@%@%@", @"x("
        ,@"\'" ,[test_view objSrc], @"\'", @");"];
     [self.architectView callJavaScript:(@"@a", src)];

     /* WTArchitectView rendering is started once the view
        controllers view will appear */
     [self startWikitudeSDKRendering];
```

```objc
}

- (void)viewDidDisappear:(BOOL)animated {
  [super viewDidDisappear:animated];

  /* WTArchitectView rendering is stopped once the view
      controllers view did disappear */
  [self stopWikitudeSDKRendering];
}

- (void)didReceiveMemoryWarning {
  [super didReceiveMemoryWarning];
  // Dispose of any resources that can be recreated.
}

#pragma mark - View Rotation
- (BOOL)shouldAutorotate {

  return YES;
}

- (NSUInteger)supportedInterfaceOrientations {

  return UIInterfaceOrientationMaskAll;
}

- (void)willRotateToInterfaceOrientation:(UIInterfaceOrientation)
   toInterfaceOrientation duration:(NSTimeInterval)duration {

  /* When the device orientation changes, specify if the
      WTArchitectView object should rotate as well */
  [self.architectView setShouldRotate:YES toInterfaceOrientation:
      toInterfaceOrientation];
}

#pragma mark - Private Methods

/* Convenience methods to manage WTArchitectView rendering. */
- (void)startWikitudeSDKRendering{

  /* To check if the WTArchitectView is currently rendering, the
      isRunning property can be used */
  if ( ![self.architectView isRunning] ) {
    /* To start WTArchitectView rendering and control the startup
        phase, the -start:completion method can be used */
    [self.architectView start:^(WTStartupConfiguration *
        configuration) {

      /* Use the configuration object to take control about the
          WTArchitectView startup phase */
      /* You can e.g. start with an active front camera instead
          of the default back camera */

//        configuration.captureDevicePosition =
    AVCaptureDevicePositionFront;

    } completion:^(BOOL isRunning, NSError *error) {

      /* The completion block is called right after the internal
          start method returns.

       NOTE: In case some requirements are not given, the
           WTArchitectView might not be started and returns NO
           for isRunning.
       To determine what caused the problem, the localized error
           description can be used.
       */
      if ( !isRunning ) {
        NSLog(@"WTArchitectView could not be started. Reason: %@"
            , [error localizedDescription]);
      }
    }];
  }
}

- (void)stopWikitudeSDKRendering {

  /* The stop method is blocking until the rendering and camera
      access is stopped */
  if ( [self.architectView isRunning] ) {
    [self.architectView stop];
  }
}

/* The WTArchitectView provides two delegates to interact with.
    */
#pragma mark - Delegation

/* The standard delegate can be used to get information about:
 * The Architect World loading progress
 * architectsdk:// protocol invocations using document.location
     inside JavaScript
 * Managing view capturing
 * Customizing view controller presentation that is triggered
     from the WTArchitectView
 */
#pragma mark WTArchitectViewDelegate
- (void)architectView:(WTArchitectView *)architectView
    didFinishLoadArchitectWorldNavigation:(WTNavigation *)
    navigation {
  /* Architect World did finish loading */
}

- (void)architectView:(WTArchitectView *)architectView
    didFailToLoadArchitectWorldNavigation:(WTNavigation *)
    navigation withError:(NSError *)error {

```

```
  NSLog(@"Architect World from URL '%@' could not be loaded.
     Reason: %@", navigation.originalURL, [error
     localizedDescription]);
}

/* The debug delegate can be used to respond to internal issues,
    e.g. the user declined camera or GPS access.

 NOTE: The debug delegate method -architectView:
     didEncounterInternalWarning is currently not used.
 */
#pragma mark WTArchitectViewDebugDelegate
- (void)architectView:(WTArchitectView *)architectView
    didEncounterInternalWarning:(WTWarning *)warning {

  /* Intentionally Left Blank */
}

- (void)architectView:(WTArchitectView *)architectView
    didEncounterInternalError:(NSError *)error {

  NSLog(@"WTArchitectView encountered an internal error '%@'", [
     error localizedDescription]);
}


@end
```

```
//  ARViewManager.h

#import "RCTViewManager.h"

@interface ARViewManager : RCTViewManager

@end
```

```
//  ARViewManager.m

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

@implementation ARViewManager

RCT_EXPORT_MODULE()

RCT_EXPORT_VIEW_PROPERTY(objSrc, NSString);

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

@end
```

Appendix **H**

Contents of enclosed CD

```
│ readme.txt ........................ the file with CD contents description
├── src ....................................... the directory of source codes
│   ├── messaging_system ........................... implementation sources
│   │   ├── server ........................... server implementation sources
│   │   ├── web_application ...................... web mplementation sources
│   │   └── mobile_application ............... mobile 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
```