Bachelor's Thesis

# Ecological Momentary Assessments from a Cross-Platform, Native Application

Jake Davison

July, 2018

supervised by:
Dr. Frank Blaauw
Dr. Ando Emerencia

university of groningen / faculty of science and engineering

# Contents

# 1 Introduction

In this thesis we will explore the suitability of different development platforms for creating a cross-platform mobile application for use by participants in psychological studies making use of the Ecological Momentary Assessment method. Areas of focus are: keeping the code base maintainable by a small team as well as generality and extensibility of application functionality. Suitability will be assessed by looking at the features of each platform, looking at existing technology for carrying out EMA and primarily through actual implementation of such an application.

## 1.1 Ecological Momentary Assessments

Research in many areas of psychology, such as developmental psychology and psychopathology, uses questionnaires completed by those taking part in the research. Self reported answers to questionnaires are liable to vary over time due to a wide range of factors. As such researchers are now looking to extend the scope of research by repeatedly measuring psychological factors of subjects over a more extended period of time, thereby building a series of data that can offer deeper insight. This kind of research usually takes the form of questionnaires that are repeatedly issued to participants but can also range from from real-time data capture to frequent diary entries. [1] This family of techniques is called the Experience Sampling Method[2] (ESM) or Ecological Momentary Assessments.

Another strength of this method is in identifying which factors vary over time and using the data to explore why instability occurs across certain mental data points, researchers to look at the change of psychological factors over a longer period of time. This way a study can achieve a more complex and personal analysis of a subject's psyche in relation to that person themselves, rather than to a wider sample of people[3].

EMA is valued for its ability to measure data from the real moment to moment fluctuations in mental wellbeing that psychology seeks to understand, rather than what is reported retrospectively in a lab environment. Asking subjects to report their feelings while in or immediately after a particular situation (known as ambulatory assessment) is something that can lead to a far more accurate model of the mental state in question. The ever-increasing integration of web-connected devices into the everyday lives of the population provides a platform for the conducting of EMA with ease and on a scale never before possible.

A research venture at the University of Groningen involving the Johann Bernoulli Institute, department of Developmental Psychology and the Interdisciplinary Center Psychopathology and Emotion regulation (ICPE) utilises the EMA method for multiple research projects, some spanning a national level. These projects include HoeGekIsNL? (HowNutsAreTheDutch?), a platform for measuring men-

tal health factors/symptoms within the Dutch general public[4] ; and u-can-act, a platform that aims to understand the causes of early school drop out by gathering data from students and their teachers/mentors.

## 1.2 u-can-act

Through weekly questionnaires, u-can-act aims to obtain data that could lead to a better understanding of how motivation in school can be affected by various factors. In contrast to many studies of this topic, and through the use of EMA techniques, u-can-act aims to understand the different individual-level causes of school drop out, rather than blanket factors across a sample group.

Currently, u-can-act is in the process of gathering data through distribution of their questionnaires to students and mentors. The currently implemented system for doing this uses web pages generated with the questionnaire content for each participant, these are made available to the user via SMS message with a unique URL. This is a system that the developers working on u-can-act are looking to phase out, instead introducing a system in which questionnaires for a user could be retrieved directly from a RESTful API that the user could access without the use of a unique link sent via SMS.

## 1.3 Project

Ideally, the u-can-act team want to transition the system toward a mobile application for iOS and Android that could achieve this goal functionality, allowing user login, retrieval of a user's questionnaires and push notifications to alert participants of a new questionnaire. Such an application would work with the questionnaire definitions already in use by u-can-act. These definitions utilise a scheme of writing different question types, such as checkbox, radio and range slider questions, in JavaScript Object Notation (JSON).

We set out, in a project that is the subject of this thesis, to provide an early implementation of a native application for Ecological Momentary Assessments implemented in the form of a Service Oriented Architecture. The project involves interacting with the already implemented questionnaire format in use with u-can-act through an API endpoint as well as further interaction with the API in such a way that allows the application to move away from the currently implemented SMS system.

Despite the application being built with the u-can-act architecture in mind, extensibility and generality is to be considered of much importance such that the application can be considered for use with other EMA projects.

# 2 Related Work and Background

This chapter will review a sample of the plethora of existing EMA applications available currently as well as the existing technology that can be used to implement such an application. Furthermore, we investigate the concepts and use cases within the field of Service Oriented Architecture.

## 2.1 Mobile EMA Applications

To better understand the requirements for such an application as would be necessary for the project, we select some of the already available EMA solutions for mobile and analyse functionality and the software solutions they implement. These applications are the commercially available *RealLife Exp* and *mEMA* in addition to the free *PIEL Survey*.

### 2.1.1 RealLife Exp

RealLife Exp, developed by LifeData, is a fully featured application for conducting EMA research. The application itself is available for Android and iOS and supports many of the same kinds of question as the u-can-act system uses[5] as well as allowing the submission of user GPS coordinates. The application can also schedule time-fixed or random notifications prompting the user to answer a questionnaire.

The RealLife Exp system provides a web application that allows the researchers behind a study to create and schedule questionnaires with a graphical user interface. The web application also performs some analytics on the data from participants and displays them to a researcher, with the option to download this data in various formats. The use of a web application makes it possible for research groups that include no members with a programming background to deploy EMA studies.

### 2.1.2 mEMA

The application with the most expansive features of the apps reviewed as part of this project is mEMA. The iOS and Android applications support much of the same question types as u-can-act uses for their questionnaires[6]. The mEMA system additionally supports the sending of GPS and accelerometer data, as well as ambient light levels.

mEMA's system uses a web application to define and deploy questionnaires to participants. This allows the scheduling of questionnaires for a study or for individuals within the study. Results from questionnaires are also displayed on the web application, with different download formats. Again, the use of a

web application allows researchers without specific expertise to conduct EMA projects.

### 2.1.3 PIEL Survey

PIEL Survey is a free alternative to the applications discussed thus far. The functionality of PIEL Survey is certainly more restricted than that of mEMA and RealLife Exp, but the application does still support many of the same question types as used in u-can-act. The most immediate drawback to PIEL Survey is that the application is only available for iOS, limiting participants to those with an Apple device.

Questionnaires are defined and scheduled through the use of a 'control file' which can be downloaded onto a device and opened with the PIEL Survey application[7]. This makes the process of deploying a questionnaire more involved for researchers than using a web application, but still doesn't require any specific knowledge. Results are sent to a researcher via email or exported via iTunes. These results are exported as a .csv file per participant which could make compilation of data intensive for larger studies.

### 2.1.4 Need for New Application

The lowest cost per month for RealLifeExp is an annual subscription of $290 USD per month[8], this price point supports 200 participants. For mEMA, pricing ranges from $280 to $1000 per month, depending on the level of functionality required. These prices for mEMA all support up to 1,500 participants[9].

Although PIEL Survey is free to use, its code is not open source, so it cannot be modified to account for functionality that it lacks. With this considered, PIEL Survey looks to be a suitable solution for very small-scale EMA studies requiring less advanced functionality, but would not meet the demands of u-can-act. The price of commercial applications and the lack of functionality of free applications highlights that an open-source application could be incredibly useful for EMA researchers.

## 2.2 Native Mobile Applications From a Single Code Base

In order to be able to offer cross-platform usability while maintaining the convenience and flexibility of a native mobile application and delivering an application within time-constraints, the project would benefit hugely from the ability to build separate native applications for iOS and Android from a single base of code. As such, it was imperative to explore the available options, for developing the application, that would offer such a feature.

Numerous options exist that allow the development of a web application that can be run on either mobile platform through the use of a Web View. Applications such as these have come to be known as "hybrid" applications. This kind of application simulates a native application by appearing as a native application would on a user's home screen but only actually providing a kind of browser-less rendering of the web page where the application can be reached. With this in mind, the two leading options that allow creation of apps from a single code base explored for the project were the hybrid Ionic framework, building upon Apache Cordova; and the relatively new React Native.

The use of one of these technologies would not only allow the creation of both applications simultaneously and therefore the most effective use of time, but would also make for more maintainable and extensible source code since future changes made to a single code base would manifest across both platforms.

### 2.2.1 React Native

React Native is a JavaScript library developed by Facebook and community contributors that allows the development of both Android and iOS native applications from a single source code base. Released in 2015, the technology is quite new and still very much in its infancy, but already is used for some of the most widely used mobile applications like Instagram, Facebook and Skype[10]. React Native was created as Facebook began to criticise the slower performance of HTML based hybrid apps.

React Native is built on top of the previously existing React platform, which is geared towards web based applications. As such, it inherits it's Document Object Model (DOM) structure for rendering the application on screen, this is similar to the way HTML defines a structure for a page and thus makes for intuitive front-end programming. This structure treats a single screen as a tree of components within other components.

React Native utilises JavaScript, specifically the JSX extension which facilitates the use of an XML style syntax within JavaScript that can then be compiled to pure JavaScript. React Native's use of JSX allows a developer to define components with XML style code, providing the ability to build components according to a DOM. Behaviour and styles for each component within the model are defined through JavaScript which React Native can compile into native code where necessary, such that this information can be used to control the DOM that is used by either operating system to structure the displayed content.

Types of components are defined separately from one another, by encapsulation, and then individual instances are created based on "props", variables that are passed to define certain aspects of a component instance or to update a component lower down in the tree. This way, information is primarily passed down the tree to define what the user sees and interacts with on screen. The

encapsulation of these components provides the opportunity to write highly extensible and generalised code.

The Domain Object Model used by React Native is a Virtual Domain Object Model (VDOM). This means that a representation of the DOM is stored in memory; as such, scripted changes or changes based on user input can be calculated in relation to this VDOM. The advantage to this approach is that the actual DOM being used to display content to the user is only modified where necessary. More exactly, only nodes within the VDOM that are changing are updated and subsequently re-rendered in the actual DOM.

React Native creates truly native apps, with a JavaScript engine controlling some of the behaviour and passing it to native code. Certain actions, however, that are possible on one mobile operating system are not fully translatable to an equivalent action on another. It is for this reason that React Native can involve the writing of separate React Native code for each OS and in certain cases, due to the immaturity of the platform, even Native Code. Facebook describe this in calling the React Native approach "learn once, write anywhere", meaning one can learn only React Native and program native apps that work for different operating systems. This phrase sums up well the fact that React Native prioritises native deployment over using a single code base.

### 2.2.2 Ionic / Apache Cordova

Apache Cordova is a technology that utilises the hybrid approach to application development. It provides a series of plugins that allow interface between various mobile platform features and and a web view. In this way, an HTML based web application can be run in a Web View with Cordova plugins that access native functionality such as file systems and notifications[11][12]. This architecture provides means to use a hybrid application to utilise functionality of a mobile OS that would otherwise require native code.

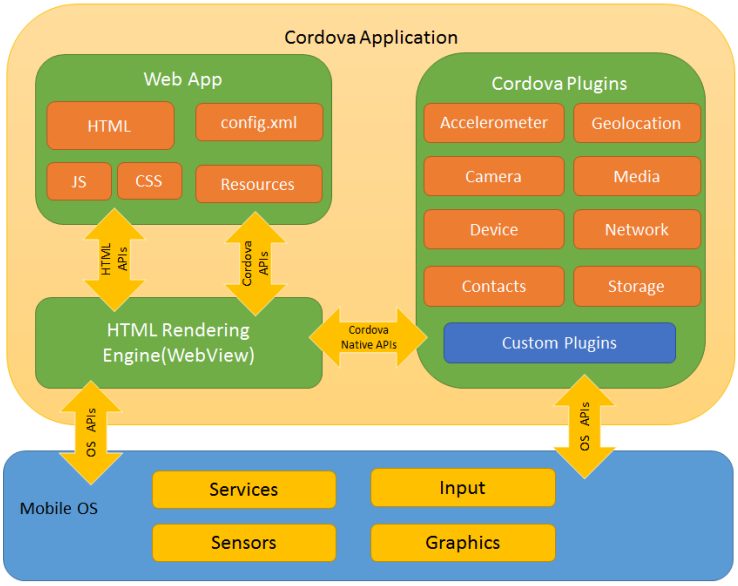


Figure 1: The Architecture of Cordova[11]

Apache Cordova works by detecting which operating system it is running on and then using functions called within the Web View application to call different native functions within the operating system. In this way, generic Cordova functions called by the web application lead to functions being called in native code. Even interaction with Cordova is wrapped by Ionic, so Cordova is abstracted from the user. This "write once, run anywhere" approach gives the platform a real focus toward cross-platform capability and speed of development.

Ionic is a popular framework that works alongside Apache Cordova by running in a Web View and interacting with the plugins of Apache Cordova. Ionic thus allows a developer to use a technology stack of HTML, CSS and JavaScript to create a web application with access to native functionality through the use of Apache Cordova as a bridge between the Ionic codebase and the operating system.

Ionic comes closer to providing a native feel than many other hybrid development frameworks due to its built-in library of user interface components that display differently based on the operating system the application is run on in order to mimic native UI components. These components, however are still composed of HTML, CSS and JS.

Ultimately, Ionic provides a strong framework for building cross-platform applications quickly and from a truly single code base. Due to the level of generality and abstraction from the operating system itself, this focus sacrifices the possi-

bility of writing truly native apps.

### 2.2.3 Choice of Technology

React Native and Ionic are technologies with different aims. Within the contradiction between cross-platform code and native deployment, the two platforms aim at different poles. React Native prioritises creating true native applications, while Ionic places a focus on using a single base of code to deploy across multiple platforms.

The u-can-act team was already migrating some of their web interfaces to ReactJS, so the use of React Native would enable the reduction of the u-can-act technology stack as well as possibly facilitating resuability of some code or easier interaction between application and web components. As well as this, there was a preference of the u-can-act team for a truly native application, so the quick scalability and maintainability of a hybrid application was superseded by the usability and speed of a native application.

Another possible benefit to the use of React Native is the React Primitives library, developed in-house at Airbnb. This JavaScript package generalises various components that vary between React Native and ReactJS such that code using React Primitives can be rendered by ReactJS or React Native, allowing web implementations to be supported from the same single code base as the mobile native applications[13].

## 2.3 Domain Specific Languages

Domain Specific Languages are programming languages dealing with specific purposes as opposed to General-Purpose Languages. DSLs are written in GPLs and seek to abstract the way that the general purpose code performs tasks defined by the code written in a DSL. This allows DSLs to be simple and understandable by non-programmers.

The JSON definitions of a questionnaire used by u-can-act do not constitute a domain specific language in the sense that they simply define an object in JavaScript, but they do follow much of the same principles of abstraction and can be used by a non-programmer. As such, domain specific languages were an interesting topic of research for this project.

Quby is a system that uses a DSL written in Ruby to allow the definition of questionnaires by domain experts who are not programmers[14]. It shares many aspects of how questionnaires are defined with the u-can-act system and can be used to define much of the same question types. Quby has shown promise over GUI based systems of writing questionnaires due to the ability to copy and paste example questions with advanced features or to duplicate similar questions[15]. This shows promise for the u-can-act system's method of defining

questionnaires, even when measured against the GUI based approaches of other EMA applications.

## 2.4 Service Oriented Architectures

A Service Oriented Architecture is a conceptual model for distributed systems in which the interactions between nodes in the system can be viewed as exchanges of services. The model characterises nodes as being service requesters or service providers for each interaction. With requesters generally being end-user applications requesting services from some provider[16].

The proposed u-can-act system can be viewed as a Service Oriented Architecture, with the API offering multiple services at any time while the end-user application can request services like retrieving a questionnaire or saving user input. HoeGekIsNL also utilised a Service Oriented Architecture for their EMA delivery system, citing that it helped support code reusability as well as keeping different areas of the system decoupled from one another[17].

# 3 System Overview

The project required working alongside existing technology, this concept introduces this technology more specifically, as well as providing an overview of the conceptual architecture for the final system.

## 3.1 Project Requirements

A requirements document based on the scope and expectations for the project is very intrumental in such a software development project. This was created with an iterative process of development in mind, where the application is built up from its most foundational elements, with more specific functionality added on top of this, as well as the tweaking of features and refactoring of code to create more cohesion between elements added at different times during the project.

The requirements document includes a rough timeline to which project progress could be compared. This basic idea of which features should be prioritised and what is a realistic time frame in which they could be implemented, provides a strong foundation to guide workflow from the beginning of the project. Requirements are shown below, the timeline is available in the appendix.

1. **Priority: High** - Single React Native code base must facilitate implementation of native Android and iOS applications.
2. **Priority: High** - Fetches, parses and displays questions from an existing REST API.
    - REST API must use HTTPS[1]
    - Focus on generality and extensibility.
3. **Priority: Moderate/High** - Pushes question responses via REST.
    - REST API must use HTTPS[1]
4. **Priority: Moderate/Low** - Login system allowing user accounts to access questions.
5. **Priority: Low** - Push notifications that inform user of new questionnaire and tap-through to it.
6. **Priority: Low** - Have complete working versions of application on both iOS and Android.

[1] Requirement of Apple for RESTful requests but also important for user data privacy.

## 3.2 System Architecture

A fundamental aspect of the project was the need to work within the context of an existing API for defining questionnaires used within the u-can-act infrastructure. Architecture was also already in place for scheduling questions as per the current implementation of u-can-act, but some of this would have to be changed to work alongside the desired functionality of the application, such as features facilitating the replacement of SMS messages used to send questionnaires to participants.

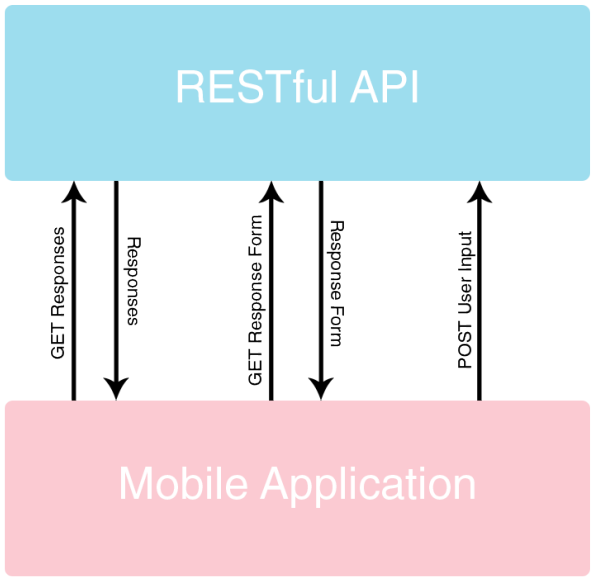


Figure 2: The service oriented architecture of the mobile application

It is important here to make a distinction between a *response* and a *questionnaire*. Questionnaires within the context of the u-can-act system are general definitions of questionnaires, responses are personalised forms (with information like names of student/mentor added) that are sent to each participant to fill out. The application does not deal with these general questionnaires, so for the purpose of clarity and readability, responses are sometimes referred to as questionnaires in this thesis.

As in the figure 2, the application interacts with multiple endpoints of the u-can-act API, namely:

- **GET - Responses for a user:** When a user logs in, the application looks

up this user's available responses, a personalised form of a more general questionnaire.

- **GET - Response form for a user:** After a user has selected a response from the list generated from the login request, this individual response question form is requested from the server.
- **POST - User Input:** When a user has completed a response form, their input is compiled into a JSON object and sent to the API server where it is stored.

## 3.3 Display of Questions

The most fundamental focus of the project is to develop code in React Native that can pull in a questionnaire, formatted in JSON, and then display these questions according to various possible parameters defined in the JSON code. The scheme, according to which which questions are defined in JSON, was that of which was already in use by u-can-act before the beginning of the application's development.

```
{
  id: :v1,
  type: :radio,
  title: 'What did the most important events relate to?',
  options: ['hobby/sport', 'work', 'friends', 'romantic relationship', 'home']
}

{
  id: :v2,
  type: :range,
  min: 0,
  max: 100,
  title: 'Was it important to you about whom you filled a questionnaire?',
  labels: ['completely unimportant', 'very important'],
}
```

Figure 3: Differences in formatting of a Radio and Range question type.

Questions differ in the way that they are defined, according to type. As such, the type of a question is the first thing to identify when reading a questions. This allows the rest of the question to be parsed according to parameter expected for that question type.

Types to be supported by the application are range, checkbox, radio, textfield, time, date, raw and unsubscribe.

### 3.3.1 Range

Range question types represent that of a very common question format in questionnaires. The question type displays a title question and a horizontal bar with slider that can be dragged between two labels displayed at each end of the bar.

### 3.3.2 Checkbox

Checkbox questions are also fundamental to most questionnaire formats. This question type displays a title question along with a list of checkboxes that represent different answers to the question. As many of these options as a user wishes can be selected.

### 3.3.3 Radio

Radio questions are similar to checkbox questions and also common in questionnaires. These questions display just like a checkbox question, but only one answer may be selected by a user. To differentiate between these questions and checkbox questions, the icon used to represent a selected and unselected option should be different to that of a checkbox question.

### 3.3.4 Textfield

Textfield questions consist of a title question and a text field that allows for user input. This allows the user to give their own answer as opposed to choosing from a predefined set.

### 3.3.5 Time

Time questions display a title question as well as an input allowing a user to select a number of hours in a predefined range with a step size between options also predefined.

### 3.3.6 Raw

Raw question types require no user input, they are simply a means of displaying some HTML formatted content, generally text. This question type is generally used as a preface to provide extra info before displaying a group of questions.

### 3.3.7 Unsubscribe

Unsubscribe questions are mostly intended to be displayed at the start of a questionnaire, but can appear anywhere. They display some HTML formatted information, along with a button (with predefined text) that allows the user to unsubscribe from receiving the questionnaires any longer.

## 3.4 Required and Supported Functionalities

There are certain required parameters for each question type that must be present in order to display the question. There are also parameters that can specify extra functionality that should also be handled in the displaying of questions.

```
{
  id: :v1,
  type: :checkbox,
  title: 'What did the most important events relate to?',
  options: ['hobby/sport', 'work', 'friends', 'romantic relationship', 'home']
}

{
  section_start: 'The main goals',
  hidden: true,
  id: :v2,
  type: :checkbox,
  required: true,
  title: 'On which goals did you work with the student during this week?',
  tooltip: 'some tooltip',
  options: [
   { title: 'Improving or maintaining the relationship.', shows_questions:
      %i[v2 v3], tooltip: 'relationship with the student' },
   { title: 'Gaining insight into their experience.',  hides_questions: %i[v4 v5] },
   'Gaining insight into their surroundings.',
   { title: 'Giving your own insight.', shows_questions: %i[v8 v9], stop_subscrip
      tion: true },
   { title: 'Developing skills.', shows_questions: %i[v10 v11] },
   { title: 'Changing or adjusting their environment.', shows_questions: %i[v12] }],
  show_otherwise: true,
  otherwise_label: 'No, because:',
  otherwise_tooltip: 'some tooltip',
  show_after: Time.new(2018, 5, 6).in_time_zone,
  section_end: true
}
```

Figure 4: Required and Supported Examples of a Checkbox Question

Examples of such parameters are:

- tooltip: Expects a string. If defined displays an icon that, when tapped, shows a popup containing the string.
- tooltip (defined for an option): As above but, if defined, displays an icon next to an option rather than for the entire question.
- required: Expects a boolean. Stops the user advancing past the question without input if true.
- shows_question/hides_question (defined for an option): Expects an array of question ID strings. Each option can be defined to, when selected, show or hide other questions specified by their IDs.
- hidden: Expects a boolean. Defines whether a question is hidden by default or not.
- stop_subscription (defined for an option): Expects a boolean. Specifies whether or not an option, if selected, will unsubscribe the user from the questionnaire.

# 4 Implementation

There are multiple aspects of the application requiring different considerations in order to begin implementation of each of these functionalities. Some components of the program allow for a more dynamic approach to problem solving, allowing problems to be solved iteratively, as they are encountered. Others, especially those requiring interaction with APIs, require a more preconsidered solution.

## 4.1 Receiving, Parsing and Displaying Question Data

The most vital part of the application is the ability to pull question data from a REST API, and display a set of questions with all of the features supported by the question API. To understand the domain of what would be required in terms of display presentation, we used two example questionnaires displayed within the already existing web application. This information was then used to devise a representation in the user interface.

The Question Screen is initialised by being passed a URL suffix as a prop to use for retrieving the right questionnaire. Thus, when the Question Screen is first mounted on the display, it sends a GET request to the u-can-act API. The URL for the endpoint itself is predefined but can be modified in a file *endpoints.js*. Once the request has completed, an response error code is shown in the case of a failure. Otherwise, the first question is mounted as per the usual functioning of a questionnaire as described below.

In terms of the DOM used within React Native, as well as the way in which questions are passed from the API, the most effective to represent questions seemed to be to display a single question screen with buttons to navigate through the array of questions parsed from the API. In this way, different question types are implemented as individual custom React Native components, one for each type of question (checkboxes, radio buttons, slider, etc.). This facilitates encapsulation, allowing the definition of the possible requirements of each kind of question within its respective component; while also decoupling question types from the questionnaire form, since any new question types could be implemented simply by writing a new component for that type to the application, making extending the application simple. This displays the strength of React Native's integration of the Component concept with its VDOM, in allowing for extensible, decoupled modules of code.

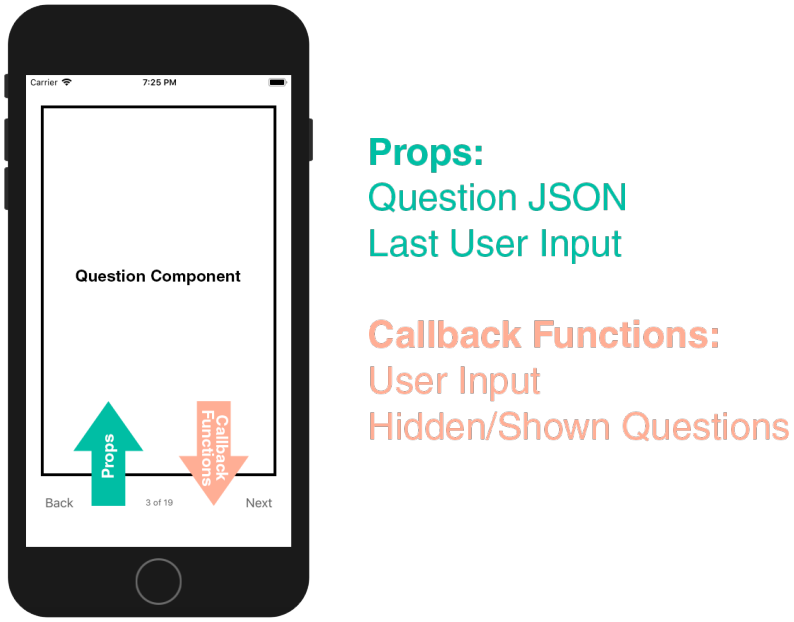


Figure 5: Question Screen showing interactions with each component.

The interaction between the Question Screen and the Question Components starts with the Question Screen determining the type of whichever Question has been navigated to by the user. Then the type's corresponding Question Component is passed the question JSON and the last user input recorded for that question as props and is subsequently rendered. If there is no previously recorded data, the Question Component is initialised as having the default input for that type.

Once the Question Component is rendered, any user input upon the component is immediately passed to the Question Screen which then, in turn updates the display of the Question Component itself. Any questions that should be hidden or shown due to the user's input is also passed to the Question Screen which either adds these question IDs to an array of hidden questions or removes them from it. When a user navigates to the next or previous question, the entry before or after the current question in the array of question parsed from JSON is passed to its respective Question Component. Before this, however, a question's ID is checked against the array of hidden questions. If the question should be hidden, it is skipped.

This information, such as user input and hidden questions, is stored in the state of the Question Screen component. State is a concept within React Native which refers to a set of variables that persist as a component is re-rendered. A component within the VDOM is only re-rendered if its state or props change.

All other variables within the component are lost after a re-rendered. Keeping these variables in state allows Question Screen to maintain these variables for the length of the questionnaire and also to re-render selectively, based upon changes in user input state caused by callback methods within a Question Component.

## 4.2 Logging In to Receive Questionnaires

Logging in is performed within an Authenication Component, mounted in the home screen of the application. Login is performed using a username consisting of 4 characters. These were already in use with the u-can-act system and were used as part of the unique URL sent to each user. Though a password field is present in the application's Authentication Component, passwords are not yet a feature of the u-can-act API as it is still transitioning from using SMS messages and unique links. Hence, the password field only enforces that the password is 4 characters long, otherwise any value will work as long as a valid username is entered.

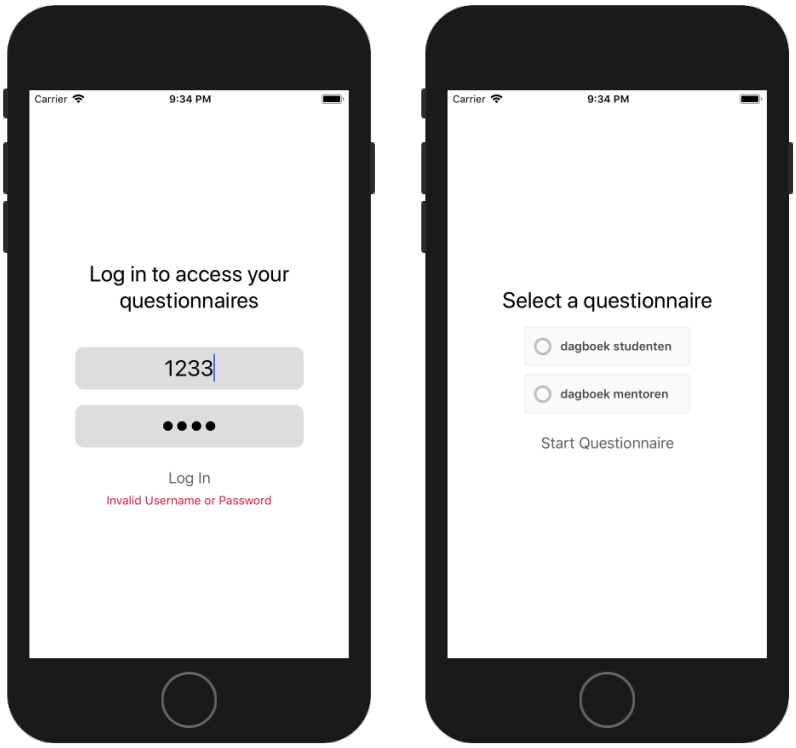


Figure 6: Authentication Component showing failed login and a retrieved response list following a successful login.

Logging in will request an array of response titles coupled with the Universally

Unique Identifier (UUID) for each response. The endpoint URL is predefined in the application code, as with in the Question Screen. The username entered is appended to the endpoint URL in order to request the list of responses available for that user. Should this response fail, the user is notified that the details they have entered are incorrect. Should it succeed, a list of radio buttons are displayed, labelled with the titles of the responses available from the API.

Once the user has selected a response that they wish to fill out, the Authentication Component performs the transition to the Question Screen, passing the UUID as a prop. This will then be used as a suffix for an endpoint URL to retrieve that specific questionnaire, as described in the previous section.

## 4.3 Sending User Input

User input is sent in a format already used by the u-can-act system. This is, again, a format defined in JSON, this time consisting of simply the UUID of the response being completed and the input for each question that was answered, an example is pictured below.


```
{
  "uuid":"caf455657-3037-4gc4-a791-85fc9f56788b",
  "content":{"v1":"Nee", "v7":"20", "v8":"56", "v9":"42", "v10":"21", "v11":"78",
    "v12":"80", "v13_hobby/sport":"true", "v13_thuis":"true", "v14":"30",
    "v15":"25","v16":"56", "v17":"100", "v18":"Nee", "v25":"53"}
}
```


Figure 7: Example JSON with user input for a completed questionnaire.

The JSON for each question is generated by that Question Component as the questionnaire is filled out. This JSON string is passed to the question screen each time the Question Screen is updated with user input from the Question Component. This is done due to the vary ways in which the JSON should be formatted, dependent on question type. As such, once the user reaches the end of the questionnaire, there is an array available of JSON strings representing user input for each question.

Once the user tries to advance beyond the final question in a questionnaire, a Submit Response component is mounted. This component displays a message to the user that they have completed the questionnaire and are free to go back and make changes to their answers but otherwise can submit their completed questionnaire. Should the user tap on the submission button, the array of JSON strings is compiled into a single string along with the UUID corresponding to the response that has been completed.

Once the JSON has been compiled for submission, a POST request is made to the API endpoint for user input. This endpoint is the same for each request, since the UUID that corresponds to the submission is part of the submission

itself. Success of this request displays a confirmation message, while failure displays an error response code. After a successful POST is made, the Question Screen closes and the user is returned to the Home Screen.

# 5 Conclusions

Completion of this project allows for the reflection upon two main areas: the suitability of the software created and its fulfillment of the requirements of the project; as well as the way the project was shaped by the use of React Native and how the platform affected the meeting of requirements.

## 5.1 Resulting Software

Following the completion of the project, the result is a single base of code that builds to two applications, for both Android and iOS, that are capable of: allowing a user to login and access their available questionnaires; retrieving their selected questionnaire from an API; displaying it according to u-can-act's definitions; compiling the user input into a form recognised by the u-can-act API and submitting this user data to the API. Here we will review the completed application with regards to the requirements specified at the beginning of the project.

### 5.1.1 Single Code Base

1. **Priority: High** - Single code base must facilitate implementation of native Android and iOS applications.

This requirement has been fulfilled. The use of React Native, with its focus on native functionality over the hybrid philosophy of having a strict single base of code with no considerations for cross-platform capability needed in code, ultimately required very little action to ensure cross-platform capability.

Within the written code for the application there is no section that is defined differently for either platform. Though the applications must be built to some native code for each platform, this is built from the exact same JavaScript code by React Native itself. Considering that the single base of code was required so as to keep the application maintainable and easily extensible by a small team such as u-can-act, this is ideal.

### 5.1.2 Pull in, Parse and Display Questions

1. **Priority: High** - Fetches, parses and displays questions from an existing REST API.
    - REST API must use HTTPS[2]
    - Focus on generality and extensibility.

[2]Requirement of Apple for RESTful requests but also important for user data privacy.

The application retrieves from a RESTful API endpoint in such a way that makes it very easy for the application to be modified to use another endpoint for the source of questions. The endpoints are all defined only once in the code which makes this very simple and time inexpensive.

As for the display of questions, the work needed to extend the application to accept new question types is also minimal. This is due to the way questions are displayed as a component within the Question Screen, meaning new questions can be added by simply creating a new component that renders such a question type and updating Question Screen to recognise its type parameter.

An areas of shortcoming for the application is that not all functionality of each question type is fully implemented. For example, questions that specify a date only after which the question should be shown will be shown regardless of this parameter. Functionality like this would not take long to implement, but there are a lot of less fundamental functionalities like this that seemed less relevant to the project at hand and so were sacrificed in exchange for time that could be spent implementing more fundamental features with the knowledge that these more minor features could be implemented at a later time without requiring substantial work.

### 5.1.3 Submitting User Input

- **Priority: Moderate/High** - Pushes question responses via REST.
  - REST API must use HTTPS[3]

User input is collected within each Question Component which also generates a JSON string containing the input data for that question, formatted as expected by the API for that question type. This, again, keeps the application extensible by keeping this JSON generation within the Question Component itself, meaning that any new forms of presenting user input to the API are handled within a new component.

As with the other API endpoints, the endpoint for posting data appears only once in the code and is easy to modify to use another endpoints. For POST requests, an authentication string is sent in the header, this can also be defined along with the endpoints for use by other projects.

### 5.1.4 User Login

- **Priority: Moderate/Low** - Login system allowing user accounts to access questions.

The login system was implemented in such a way that the existing infrastructure behind u-can-act did not need to be extensively modified. As such it uses

[3]Requirement of Apple for RESTful requests but also important for user data privacy.

4 character usernames that were already in use as part of the unique URLs sent to each user.

The system does not yet allow for true username-password authentication, this could be useful functionality to have in fully securing a study against sabotage or even user error in which they input the wrong user name. However, the application is extensible to do so given that it already saves the password to a variable when login is attempted, it just doesn't yet do anything with that variable.

The list of available questionnaires that is retrieved requires only very minimal data from an API, namely a title and a UUID that can be used to GET the questionnaire from an API endpoint. This leaves the API easy to set up for this feature as well as leaving it open to be extended.

#### 5.1.5 Push Notification

- **Priority: Low** - Push notifications that inform user of new questionnaire and tap-through to it.

This is the only feature in the list of requirements that was not met at the end of the project. There are multiples reasons that this is the case. Primarily, this feature being of low priority, this was one of the last areas of implementation to be considered. Thus once other requirements had been met, it seemed more important to look towards testing, refactoring and debugging of the code, especially in view of the timeline set out at the beginning of the project.

Push notifications would also mark a considerable step in departing from the SMS delivery system for questionnaires. As such, it would have required a more extensive reworking of the API system than many other features required but in a small timescale due to the late stage in the project.

Push notifications would also have been the first major use of native OS functionality in the application which brings complications in development and requires significant testing. React Native, due to its immaturity, does not yet offer a way to implement push notifications in its standard library without native code. An external library called *react-native-push-notifications*[18] may be of use for this purpose however.

#### 5.1.6 Applications Running on Device for iOS and Android

- **Priority: Low** - Have complete working versions of application on both iOS and Android.

This requirement mainly covered testing and debugging of the final code. The important cross-platform consideration for the project was ensuring that the

application *could* be used across platforms with a single code base. This requirement was more concerned with achieving a kind of minimal viable product that actually *does* function across both platforms.

This requirement actually required very little work. Once the application was built for devices on both platforms and tested, there were minimal issues that manifested differently depending on device. The application was developed for most of the project using Xcode's iOS Simulator and thus when deployed to an iOS device for the first time, exhibited no issues.

On Android only one issue arose, which was very minor. When the keyboard pops up on iOS it simply obscures about 40% of the screen. While on Android, the keyboard takes up a similar area of the screen, but actually resizes the entire screen to fit within the area not used by the keyboard. This caused a display issue when a user tried to log into Android where the text inputs for username and password overlapped one another which was confusing and unsightly. Solving this problem was simple and required only enforcing a minimum height for the container of these text input fields.

Overall, less than a full work day was spent resolving cross-platform issues and thus this low-priority requirement was completed, even at a late stage in the project.

## 5.2 Benefits and Drawbacks to Using React Native

Due to the focus of this work on creating cross-platform applications from a single code base, it is very useful to consider the implications that the use of React Native had for this project.

### 5.2.1 Benefits

As covered in the previous section, the experience of building the single code base to both devices in this project was one of few issues and little effort. React Native offers command line tools that can be used to deploy to a device or to build the application for deployment via Apple's Xcode or Android Studio.

Another benefit to using React Native is its thriving and expanding community. React Native is developed and maintained by an open-source community as well as a team of developers at Facebook. The heavy contribution from a large community results in issues being resolved extremely responsively, evidenced by the 12,174 issues (at time of writing) resolved since January 2015[19]. This community is also constantly building a wealth of external libraries for React Native that can often eliminate the need for using native code in areas where pure React Native has not yet eliminated this necessity.

During development, React Native uses a local development server to bundle the JavaScript for running on device or simulator. As such, updates made to code can be bundled immediately allowing a developer to see their changes reflected immediately in their code. Not only is this shortening of the feedback cycle between code and behaviour useful for the purposes of implementation and debugging, it is also valuable in speeding up the process of learning how React Native works.

### 5.2.2 Drawbacks

The primary drawback to using React Native at this point is its immaturity. React Native is 3 and a half years old and relies on much community contribution for progress. As a result, development of certain functionality can be convoluted or rely heavily on external libraries, especially when dealing with native OS interaction. These issues have been cited by Airbnb as a reason for their recent decision to cease use of React Native[20]. The immaturity of React Native is also evident in the documentation for the platform, which can be vague, incomplete and at times out of date. This of course can lead to confusion over how certain aspects of the platform should really be used. These things considered, the aforementioned strength of React Native's community does help to overcome these issues at times.

The need, at times, for use of native code represents another drawback. React Native is not created with only the intention of working from a single base of code, as summed up through the *"learn once, write anywhere"* principle it purports to follow. This is a philosophy that React Native clearly follows, but as of yet falls short of realising. For this project, interaction with native OS functionality was not a focus and so native code was not required. However, it became clear when making a decision as to how feasible the implementation of push notifications would be in the project's time scale, that accessing this kind of functionality is where React Native shows its current limitations in both working from a single code base and working without the use of native code.

# 6 Future Work

Since the completion of the project has led to what it aimed to achieve, namely a generalised application for performing EMA that is focused on being expandable, it is useful to assess possible improvements and extensions to the functionality of the application.

## 6.1 Extended Data Point Types

Inline with other valuable aspects of the EMA method, as well as other EMA applications looked at in this thesis, an valuable area of expansion for the application would be the capacity to capture data such as GPS location, accelerometer data and perhaps even biometric data from wearable technology. These data points would not currently be useful to the u-can-act project but would add much value to the application as a more general application for conducting Ecological Momentary Assessments.

In addition to collecting this data for submission to researchers, this could also be used to harness the ambulatory assessment potential of EMA by prompting the user to fill in questionnaires when certain conditions are met such as a high heart rate or entering a certain location.

## 6.2 JSON Web Token Authentication

JSON Web Tokens (JWT) are a standard used for providing authentication between two parties. The basic concept is that an authentication service can generate a token with three component parts: a header, a payload and a signature. The header contains information as to the hashing algorithm used for the signature; the payload contains the information that is being shared, such as log in details; and the signature is created by hashing the other two parts with some secret string known only by the senders and receivers. These components are all encoded in Base64URL and put together. The result is a token that can be used by the requester to prove to a provider that it is who it claims to be and that the token has not been modified.

Token based authentication would provide a more secure layer of authentication for the application in which the application would generate a token for a user that would be sent with any request made to the API. The API would then require that this token be received with any request for a service to perform that operation. This prevents the API endpoints that are not used for authentication from being targeted, as these do not currently require a user's log in details.

## 6.3 Push Notifications

The subject of push notifications has been well covered in this thesis and is the most immediately implementable of features presented in this section. Some modification of the API infrastructure supporting this application would be required to create and send the notification signals, but this is within the immediate scope of the u-can-act developers working on the API.

As previously suggested, *react-native-push-notifications* could be very useful to the implementation of this feature and also in keeping the platform independence of the application's source code by mitigating the need for native code.

## 6.4 React Primitives

Another possible extension to the project could be to rework the code base already in place to make use of React Primtives. Doing this would allow sections of the code to be reused completely for displaying on web through React.js. This could be useful to the u-can-act project as they move certain elements of their web infrastructure over to React.js as well as potentially providing greater flexibility to other EMA projects in terms of application deployment and code reuse.

# 7 Appendix

## 7.1 Timeline

Note that timeline is created pre-implementation and is thus only a rough guide of the expected time taken to implement certain requirements. The timeline is set to prioritise the top 3 requirements. If implementations are completed earlier than expected, further functionality will be added as per the lower-priority requirements. Unit testing and general documentation should be ongoing.

- **Week 1:** Development environment set up, requirements finalised and time-projected, relevant literature identified and studied.
- **Week 2:** Proof of concept app created that can GET from a REST API and display the results, architecture documentation and planning.
- **Week 3:** Minimal front-end set up, further documentation/planning work.
- **Week 4:** Fetching of question data implemented, initial formatting of different question types.
- **Week 5:** Further work on display of multiple expected question formats.
- **Week 6:** Front-end/UX improved w.r.t questions.
- **Week 7:** Sending of question responses via REST.
- **Week 8:** Further work on sending of question responses.
- **Week 9:** Initial refactoring and debugging.
- **Week 10:** Testing across (both) platforms.
- **Weeks 11 & 12:** Final tweaks and refactoring.

# References


[1] S. Shiffman, A. A. Stone, and M. R. Hufford, "Ecological momentary assessment," *Annual Review of Clinical Psychology*, vol. 4, no. 1, 2008. PMID: 18509902.

[2] M. Csikszentmihalyi and R. LARSON, "Validity and reliability of the experience-sampling method," vol. 175, pp. 526–36, 10 1987.

[3] S. Shiffman and A. A. Stone, "Ecological momentary assessment: A new tool for behavioral medicine research," in *Technology and methods in behavioral medicine* (D. S. Krantz and A. Baum, eds.), ch. 7, p. 117–131, Lawrence Erlbaum Associates, 1st ed., 1998.

[4] L. V. D. Krieke, B. F. Jeronimus, F. J. Blaauw, R. B. Wanders, A. C. Emerencia, H. M. Schenk, S. D. Vos, E. Snippe, M. Wichers, J. T. Wigman, E. H. Bos, K. J. Wardenaar, and P. D. Jonge, "Hownutsarethedutch (hoegekisnl): A crowdsourcing study of mental symptoms and strengths," *International Journal of Methods in Psychiatric Research*, vol. 25, no. 2, pp. 123–144.

[5] LifeData, *LifeData Help Center - Prompts*, Last Accessed 15/06/2018. `https://lifedata.zendesk.com/hc/en-us/articles/208224026-Prompts`.

[6] ilumivu, *mEMA Full Features List*, Last Accessed 15/06/2018. `https://ilumivu.com/solutions/ecological-momentary-assessment-app/full-features-list/`.

[7] PIEL Survey, *Survey Control File*, Last Accessed 15/06/2018. `https://pielsurvey.org/tutorials/control-file/1/`.

[8] LifeData, *LifeData - Pricing*, Last Accessed 15/06/2018. `https://www.lifedatacorp.com/pricing`.

[9] ilumivu, *Pricing -mEMA*, Last Accessed 5/07/2018. `https://ilumivu.com/pricing`.

[10] Facebook, *React Native Documentation - Who's Using React Native*, Last Accessed 31/05/2018. `https://facebook.github.io/react-native/showcase.html`.

[11] Apache Cordova, *Apache Cordova Documentation - Architectural Overview*, Last Accessed 04/06/2018. `https://cordova.apache.org/docs/en/latest/guide/overview/index.html`.

[12] Apache Cordova, *Apache Cordova Documentation - Core Plugin APIs*, Last Accessed 04/06/2018. `https://cordova.apache.org/docs/en/latest/guide/support/index.html#core-plugin-apis`.

[13] Airbnb Engineering and Data Science, *Airbnb react-sketchapp Documentation - Universal Rendering*, Last Accessed 11/06/2018. `http://airbnb.io/react-sketchapp/docs/guides/universal-rendering.html`.

[14] M. Veldhuis, "Quby: A domain-specific language for non-programmers," 2012.

[15] M. Veldhuis, "Quby: A domain-specific language for non-programmers," 2012.

[16] M. P. Papazoglou, P. Traverso, S. Dustdar, F. Leymann, and B. J. Krämer, "Service-Oriented Computing: A Research Roadmap," vol. 05462, pp. 1–29, May 2005.

[17] F. J. Blaauw and A. Emerencia, "A service-oriented architecture for web applications in e-mental health: Two case studies," pp. 131–138, Oct 2015.

[18] J. Trujillo, *Github Repository & Readme - React Native Push Notifications*, Last Accessed 15/06/2018. `https://github.com/zo0r/react-native-push-notification`.

[19] *Github Issues - React Native*, Last Accessed 1/07/2018. `https://github.com/facebook/react-native/issues`.

[20] G. Peal, *Sunsetting React Native*, Last Accessed 27/06/2018. `https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a`.