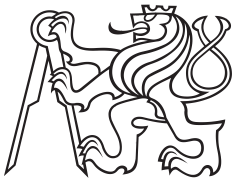
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Mobile Application for Safety Occurrence Reporting

Rail Chamidullin

Supervisor: Ing. Martin Ledvinka
May 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Chamidullin** Jméno: **Rail** Osobní číslo: **435602**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Mobilní aplikace pro hlášení bezpečnostních událostí

Název bakalářské práce anglicky:

Mobile Application for Safety Occurrence Reporting

Pokyny pro vypracování:

1. Analyzujte možnosti tvorby mutliplatformních mobilních aplikací.
2. Navrhněte a vytvořte multiplatformní mobilní aplikaci pro hlášení bezpečnostních událostí.
3. Analyzujte, jakým způsobem lze tuto mobilní aplikaci integrovat s již existujícím serverovým řešením vytvořeným skupinou znalostních a softwarových systémů.
4. Analyzujte a porovnejte různé přístupy k synchronizaci dat vytvořených v offline módu mobilní aplikace a serverové databáze.
5. Integrujte vytvořenou mobilní aplikaci se serverovým řešením. Zvláštní důraz bude kladen na zabezpečení aplikace a na synchronizaci dat vytvořených v offline režimu.
6. Ověřte správnou funkčnost aplikace automatizovanými testy. Proveďte uživatelské testování mezi experty v oblasti letecké dopravy.

Seznam doporučené literatury:

Bonnie Eisenman. 2017. Learning React Native: Building Native Mobile Apps with JavaScript, O'Reilly Media
Zach Mccormick and Douglas C. Schmidt. 2012. Data synchronization patterns in mobile application design. In Proceedings of the 19th Conference on Pattern Languages of Programs (PLoP '12). The Hillside Group
Bogdan Kostov, Jana Ahmad and Petr Křemen. 2017. Towards Ontology-Based Safety Information Management in the Aviation Industry. On the Move to Meaningful Internet Systems: OTM 2016 Workshops. Springer International Publishing
III.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Martin Ledvinka, Skupina znalostních softwarových systémů FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **19.02.2018** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Martin Ledvinka podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

Datum převzetí zadání

Podpis studenta

Acknowledgements

I would like to thank my thesis advisor Ing. Martin Ledvinka for sharing expertise and valuable guidance.

I would also like to thank the experts who were involved in the user testing of the mobile application.

Declaration

I declare that this work is all my own work and I have cited all sources I have used in the bibliography.

Prague, May 24, 2018

Prohlašuji, že jsem předloženou práci vypracoval samostatně, a že jsem uvedl veškerou použitou literaturu.

V Praze, 24. května 2018

.....
Rail Chamidullin

Abstract

Reporting Tool is a web application for safety occurrence reporting. The goal of this thesis is to analyse possibilities of mobile application development, communication with a server of Reporting Tool and synchronisation of data created during offline mode. And then implement a multiplatform mobile application and integrate it with the server.

Keywords: mobile multiplatform application, React, React Native

Supervisor: Ing. Martin Ledvinka

Abstrakt

Reporting Tool je webová aplikace pro hlášení bezpečnostních událostí. Cílem práce je analyzovat možnosti vývoje mobilních applikací, komunikace se Reporting Tool serverem a synchronizace dat při offline režimu. A následně vyvinout multiplatformní mobilní aplikaci a integrovat ji se serverem.

Klíčová slova: mobilní multiplatformní aplikace, React, React Native

Překlad názvu: Mobilní aplikace pro hlášení bezpečnostních událostí

Contents

Figures

Tables

Chapter 1

Introduction

In organisations, safety management is an important part, especially when it comes to a high-risk industry. And in aviation field safety management is considered mandatory. About that is dealing the project called Indicator-Based Safety, shortly INBAS. It is described as research and development of progressive methods for measuring aviation organisation's safety performance. This project is elaborated by Knowledge-based and Software Systems Group, which is part of Department of Cybernetics at FEE CTU.

A part of the INBAS project is a Reporting tool. It is a software designed for reporting safety occurrences. Main functions of the system are:

- creating occurrence reports,
- classification of severity of occurrences,
- analysis of sub-events,
- statistical evaluation of occurrence reports.

The Reporting tool is currently implemented as a web application, i.e. combination of a back-end server and a front-end user interface (UI), which is accessible from a web browser of devices such as desktop or laptop computers, tablets and smart-phones. Though the application provides all necessary data of reports, it also brings inconvenience regarding mobile devices. The application is accessible on mobile devices only when connected to the Internet and UI is adjusted mainly for large screens.

Imagine a situation when a safety occurrence has happened, e.g. an aircraft has entered a runaway without permission, one of the employees did notice that, but he is not around his work computer and his smart-phone has no Internet connection. That means he needs to go to his office to report an occurrence in the Reporting tool. But what if he does not have the time or he cannot leave his workplace. He needs to remember or note details about safety occurrence and report them later after his job is done. But what if then it is too late, or he forgets to report the occurrence at all. That is one of the problems organisations are dealing with, where the management has no real-time review about the situation at the workplace.

This disadvantage of the current web application is the primary motivation for developing an additional application designed for mobile devices. The new mobile application will be working independently on the Internet connection. During the offline mode users will be still able to report safety occurrences, and when connected the application will synchronise with the server of the Reporting tool. It will be a solution, which employees could keep with them all the time and report safety occurrences with fewer obstacles.

Furthermore, in this situation, there is an opportunity to improve Reporting tool with extra features. As all mobile devices have cameras and sensors, there is a possibility to attach sensory data to the occurrence report and provide more valuable information within the application. The support for this kind of attachments will be implemented in the mobile application within this thesis. And the users will be able to attach following sensory data:

- photos and videos from camera or local storage,
- voice recording and
- GPS location.

In case of the whole INBAS project, it would be a logical step to implement the same support for the web application.

For additional clarification, this thesis is focused on developing the mobile application, which will be a part of application Reporting tool. The reader can expect an explanation of technology used, description of used patterns for synchronisation, security of personal data, and best practices regarding implementation. However, this thesis does not contain any further information about INBAS project or safety management.

Chapter 2

Analysis of mobile application development

There are three main approaches to application development for mobile platforms - native, web and hybrid mobile applications. All of them have their benefits and liabilities, and all of them are commonly used. We can decide by provided requirements for the application which trade-off is the best for the particular case.

2.1 Native application

Native applications are developed specifically for one platform and can use all device features (e.g. camera, accelerometer, GPS, notification system, contacts, etc.). Native applications are usually downloaded and installed through application store (e.g. Apple's App Store or Google Play). Different technologies and environments are used for development on each platform. Apple's iOS uses programming languages Swift or Objective-C, whereas Google's Android uses Java or Kotlin (for performance and computation reasons it is also possible to use C++).

Benefits

- Fast performance.
- High degree of reliability, i.e. less unexpected errors.
- Applications are installed directly in the device. Therefore they are usable without the Internet connection.
- Applications can access device features.

Liabilities

- Applications needs to be created for each platform separately.

2.2 Web application

Mobile web applications are basic web applications, i.e. websites, which are adjusted for small screens. Web applications are not installed on the device. Rather they are accessed on device's web browser. Although web applications look and feel like a real native application, they are not available offline, and they can not access all device features (e.g. accelerometer, notification system, running in the background, complex gestures, etc.). Furthermore, applications are highly dependent on server and quality of network connection. The whole user interface needs to be fetched each time the browser is opened, and there is no possibility of long-term caching of the user interface, i.e. storing data to speed up future requests. That can cause performance issues.

Mobile web applications are developed using web technologies such as HTML5, CSS3 and JavaScript.

Benefits

- Accessible from all platforms. Not only mobile devices but also PCs.

Liabilities

- Applications cannot work without the Internet connection.
- Applications can struggle with performance issues.
- Applications cannot access all device features.

2.3 Hybrid application

Hybrid applications are the compromise of previous two approaches - part native applications and part web applications. They have the same behaviour as the native applications, but they are written using web technologies (HTML5, CSS3 and JavaScript). It is typically ensured by frameworks such as Apache Cordova or React Native; these frameworks provide a consistent set of JavaScript APIs to access device capabilities through native APIs. The main advantage is that the applications are available on all platforms.

Benefits

- Accessible from all mobile platforms. Applications are created once and can be compiled for each platform.
- Applications can work without the Internet connection.
- Applications can access device features.

Liabilities

- Possible performance issues. Faster than web applications but slower than native applications.

2.4 Choosing framework

The following requirements were specified for this thesis.

1. Mobile application must be multi-platform, i.e. it must be supported by both iOS and Android operating systems.
2. Application must be available offline.

It is clear that building native or web application would not satisfy both requirements. Thus, it leads to developing a hybrid mobile application.

There are dozens of available frameworks for hybrid applications. To analyse, test, compare and find the most suitable one would be enough work for the whole thesis (most probably it would also be unnecessary as the technologies change rapidly). So, in this case choosing the technology was done by researching the descriptions of a few commonly used frameworks and applying intuition and personal preferences.

2.4.1 Xamarin

Xamarin is a well know framework developed by Microsoft. The applications are written in C#. The disadvantage is that the UI for each platform must be designed separately, this was the main deal-breaker as it would be time-consuming to do one task twice (for iOS and Android) especially when the implementation of frequent changes was assumed. Therefore there was no need to analyse Xamarin furthermore.

2.4.2 Appache Cordova, Ionic Framework and Adobe PhoneGap

Apache Cordova is an open-source mobile development framework. The applications are implemented using standard web technologies such as HTML5, CSS3 and JavaScript. HTML5 provides access to underlying hardware such as the accelerometer, camera, and GPS. Furthermore, the framework can be extended with native plug-ins, which allow access to the device's compass, file system, microphone, and more. The user interface is rendered using an embedded web browser, i.e. web-view. Besides support for iOS and Android, Apache Cordova also supports Windows Phone 8, and computer operating systems: Windows 8.1/10, macOS and Ubuntu.

Ionic Framework is built on Apache Cordova and AngularJS, front-end web application framework mainly maintained by Google. Besides standard web technologies, it supports TypeScript, an extension of JavaScript, and Sass, a CSS extension.

Adobe PhoneGap is an open source distribution of Apache Cordova framework and with additional Adobe services and tools like the PhoneGap Developer App, the PhoneGap Desktop App, PhoneGap Build and PhoneGap Enterprise.

2.4.3 React Native

React Native is a framework developed by Facebook. It is based on React library which is used for developing web applications, and it carries all the features of the React. The React Native applications are developed mainly with JavaScript, and additionally, there is a possibility to use native code such as Java, Swift, etc.

The JavaScript frameworks such as Ionic Framework or Adobe PhoneGap render elements using web-views (wrappers). This approach is the most common, but it comes with performance drawbacks and issues with mimicking native UI elements. React Native, on the other hand, uses standard rendering APIs of its host platform, i.e. it uses real native UI components, which is faster.

Furthermore, following advantages have played their role in choosing React Native.

- React Native is well documented.
- Previous knowledge of React library. React Native uses the same design as React, i.e. component life-cycle (properties and state) and declarative rendering (i.e. application re-renders the minimal amount necessary).
- Build utilities for easier development (e.g. Debug mode and Hot-reloading).
- Wide community of JavaScript developers. There is a lot of discussion regarding development problems and a lot of NPM (Node Package Manager) packages which can be used.

As every great technology React Native has its drawbacks and risks. The primary concern is that this framework is relatively young, as it was released in 2015, and there are still some unfinished aspects.

2.4.4 Conclusion

In the end, React Native was chosen. It has promised good performance and simplicity. Also, the current implementation of the user interface of the web application Reporting Tool was considered, as it is written in React library.

Developing with React Native has spared a lot of time since there was no need for extra additional learning. However, as in every more extensive implementation, there were discovered some best practices, which had enforced

rewriting a lot of code. Best practices are discussed in **Chapter 5**. And the resulting impression of application is described in **Section 7.2**.

2.5 Technology specification

In this chapter are listed technologies used for developing the mobile application with a description of key features of each technology. Besides React Native framework, there were also used Redux, React Navigation, Realm, React Native i18n, React Native Keychain and Jest.

This work does not intend to supplement official resources for learning mobile development technologies. However, a minor knowledge is advised for better understanding future chapters, e.g. explanation of implementation and architectural mistakes and their solutions.

If the reader is familiar with technologies described in the following sections, he can continue to the next chapter.

2.5.1 React native / React

React Native is the main framework, that holds mobile application together and allows functioning on mobile devices with operating systems such as iOS and Android.

The framework is based on the library React, which is intended for developing user interface of web applications. There are slight differences in developing a React web application and React Native mobile application. For example, React Native applications are mainly developed in JavaScript, whereas React applications also use HTML and CSS, i.e.:

- React Native uses XML elements instead of HTML elements. Basically, for developers, it means different names of elements and sometimes different names of attributes, e.g. in React there is `<div>`, and in React Native there is `<View>`.
- Instead of CSS style-sheet stored in separate files, there are JavaScript objects called StyleSheet. This JavaScript StyleSheets contain fewer rules/styles than basic CSS and are designed with a focus on Flexbox, a CSS3 layout model that aims at providing a more efficient way to lay out, align and distribute space among items in a container.

But otherwise, React Native has the same major features as React.

The user interface of an application is assembled from React Components. When the state of application changes only React Components containing modified data re-render. That is a significant performance advantage since the application does not re-render the whole view.

React Component is a JavaScript class or an object, depending on the version of ECMAScript, a specification of scripting language that standardises JavaScript. It contains a render function which returns the view, optional Component Lifecycle related functions (explained below) and optional helper functions.

Each React Component contains Props and State.

- Props (as properties) are arguments that are passed to the component by the parent component. It is an immutable object, i.e. it can not be changed inside a current component, only by the parent.
- State is a set of variables defined inside of the component, and the variables can be changed inside of the component.

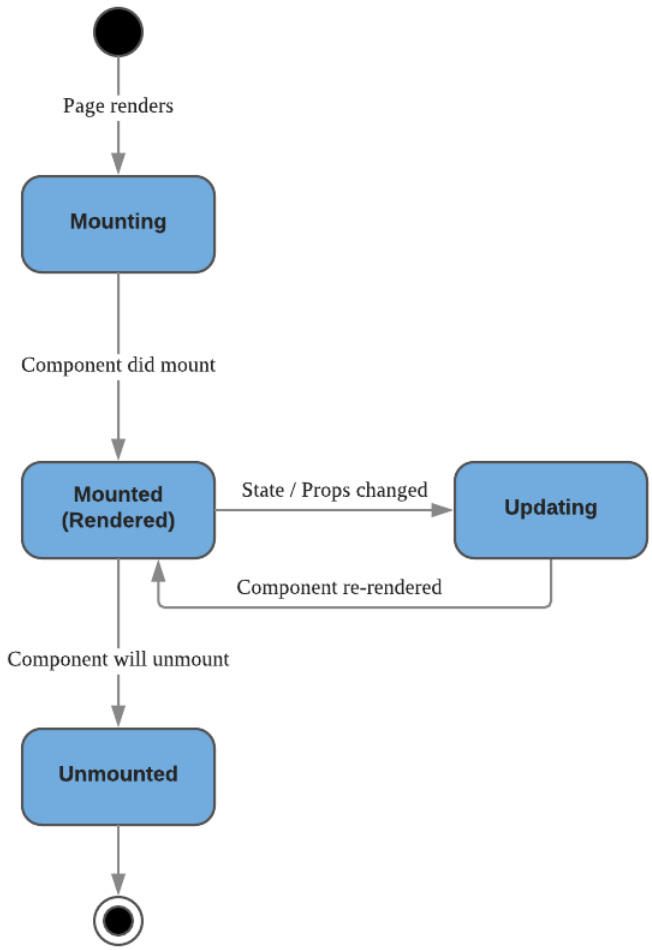


Figure 2.1: State diagram of React Component Lifecycle

Each modification of Props and State invokes re-render, see the **Figure 2.1**. That is a part of a Component Lifecycle which starts with mounting component, where the State variables are established. It continues with

optional updates of the component, where the new Props are passed, or the State is changed. And it ends when the component is unmounted, in this stage, all references should be removed, and the framework completely removes the component from the view.

There is also a specific type of React Component that does not contain State. The component is called Functional Stateless React Component, and it is used for simple use cases, when a component needs to be just rendered and its state will not change, offering potential for performance optimisations by React.

2.5.2 Redux

Redux is an implementation of Flux[9], an architectural pattern defined by Facebook. It allows to store global state and pass it to the React Components via Props. Often Redux (or similar libraries) is referred to as a single source of truth. Typical cases of usages are following:

- Storing global state of application which cannot be different across multiple components, e.g. information whether the user is logged in.
- Storing data received from the server. For example, when multiple parts of view need to render information from one data request, it is better to cache the data than to repeat the same request.

Redux must not be confused with a database since it stores the data in the computational memory and it does not access file system. I.e. when an application is closed all data stored in Redux will be removed.

2.5.3 React Navigation

The library React Navigation allows to implement transitions between screens simply. The main principle of the library is Navigator object, which contains identification of current screen, a list of previous screens and navigation functions, which allow to return to the previous screen (if one is available) or to go to the next screen. There are three types of navigators, see **Figure 2.2**:

- Stack navigator - the Main type of the Navigator, it adds Header element to the top of the screen. The header contains a title of the screen and a go back button in case there is a previous screen.
- Tab navigator - Panel with buttons typically located at the bottom, each button navigates to the different screen.
- Drawer navigator - Hidden drawer on the left (or right) side of the screen. The drawer is typically shown by pressing the button or by the swipe gesture, and it typically contains navigating buttons to the different screens and some additional information depending on the type of application.

(a) : Stack navigator **(b) :** Tab navigator **(c) :** Drawer navigator

Figure 2.2: Examples of navigators

2.5.4 Realm Database

For storing local data is used Realm Database. It is an object database, which is an alternative to SQLite. In comparison with SQLite, Realm has following advantages:

- Easy to use - Database provides an intuitive interface for writing and reading data.
- Performance - Thanks to its zero-copy design, Realm Database is often faster than raw SQLite, as stated by the creators of the database[10], which prevents the unnecessary load of unneeded data.

Realm also provides functionalities typical of a relational database such as Triggers (called Notifications in Realm), relations between objects, etc. But they are not used in this application due to the shape of entities and implementation reasons.

2.5.5 React Native i18n

One of the requirements for the mobile application is support for localisation, i.e. support for multiple languages. This feature is ensured by a library React Native i18n which is developed by a community. Shortcut i18n is common term for localization and internationalization in computing, it stands for the first and last letters in the word **i**nternationalizatio**n** and the 18 characters between them.

The simple description of how it works is following:

- There are dictionary files, and each file is designated for a specific language. The files contain key-value pairs, where the key is a unique universal shortcut and the value is a text in given language. All dictionary files contain the same set of keys, only the values vary.

- Then there is a configuration file that specifies used language of the mobile application, and typically it is adjusted by the language of the device.
- When there is needed to print some text, e.g. name of a button, an application calls a function which returns text in a specific language for given key.

2.5.6 React Native Keychain

For safely storing credentials is used library React Native Keychain which is developed by a community. The library provides access to Android keystore and iOS keychain, the official storage units for saving cryptographic keys. This library is described in detail in the chapter 4.3.

2.5.7 Jest

Jest is a library developed by Facebook, which provides an interface for testing JavaScript applications including applications implemented with React and React Native. The process of automation testing of the mobile application is described in chapter 4.4.

Chapter 3

Specification of application

The specification of the application contains list and description of requirements that the mobile application needs to fulfil, description of user interface structure and main use-cases with screenshots of the application.

3.1 Requirements

In this section are described requirements for the mobile application specified by the project supervisor. The requirements are divided into functional requirements, i.e. description of application behaviour, and non-functional requirements, i.e. criteria for quality of the application.

3.1.1 Functional requirements

The main purpose of the mobile application is to manage reports, and the application will support operations such as viewing, creating, editing and removing reports.

The attributes of report are **headline** - name of the report, **date** - when the event has occurred, **occurrence class** and **occurrence category** - two types of classifications of reports, **summary** - detailed description of report, **list of sub-events** and **sensory data** - photos, videos, audio recordings and GPS location. Besides specified attributes, which are editable by users, the application will store hidden attributes such as date of creation, author and number of revisions.

The mobile application will synchronise reports with the existing server application, and all data will be stored in the server database.

The mobile application will be able to work in offline mode, i.e. when the mobile device has no network connection. During this mode, the application

List of functional requirements	
FR: 1	The mobile application will be synchronized with existing server application and all data will be stored in the server database.
FR: 2	Communication with the server application will be provided via REST API.
FR: 3	In case of no available network connection application will store its data in a local database.
FR: 4	Users will be able to view reports.
FR: 5	Users will be able to create reports.
FR: 6	Users will be able to edit reports.
FR: 7	Users will be able to remove reports.
FR: 8	Each report will contain following attributes: ■ Headline ■ Date ■ Occurrence class (severity) and category ■ List of sub-events ■ Summary ■ Sensory data
FR: 9	Headline, date, occurrence class and category and summary will be required attributes, i.e. each report must contain this information.
FR: 10	Application will remotely load and store predefined values for Sub-event types, Occurrence class and Occurrence Category.
FR: 11	User will be able to attach photos to reports.
FR: 12	User will be able to attach videos to reports.
FR: 13	User will be able to attach audio recordings to reports.
FR: 14	User will be able to attach GPS location to reports. GPS location will be automatically recorded by mobile application.
FR: 15	Users will be able to create a short report (called Initial report) with providing only description of occurrence. The server will parse the Initial report and will return full report.
FR: 16	Application will present reports in a list or a table.
FR: 17	Users will be able to filter reports by occurrence category and class.
FR: 18	Each item in the list of reports will contain headline, date, category and class.

Table 3.1: List of functional requirements

will allow users to create reports and will save the reports to the local database.

Besides main entity - report, the application will support shorter version of a report, which is called Initial report. The initial report contains only one attribute, that is a description of occurrence. When a user creates an Initial report, the mobile application will send it to the server, which will parse text in the description, based on which it will create a regular report and send it

back to the mobile application.

The functional requirements described above are structured in a list, see 3.1.

3.1.2 Non-functional requirements

Along with required functionalities, the mobile application must also fulfil quality criteria. The mobile application must be multi-platform, supporting two widely used mobile operating systems: iOS and Android. It is expected, that in the production, the server will contain hundreds of reports, and the mobile application must ensure flawless processing of that data. The architecture of the mobile application must be designed with a support for localisation in mind, i.e. support for multiple languages may be implemented in the future.

The non-functional requirements described above are structured in a list, see 3.2.

List of non-functional requirements	
NR: 1	Application will be supported by iOS and Android operating systems.
NR: 2	Application must process and present to user hundreds of reports without recognizable flaws.
NR: 3	A part of the application will be automated tests.
NR: 4	Support for localization - (prepared, not necessary implemented).

Table 3.2: List of non-functional requirements

3.2 UI structure

The mobile application contains two top-level screens: **Reports screen** and **Settings screen**. A user can switch between them with a drawer, a UI component that slides in from the side by pressing a button or by swiping. The user needs to be logged in, to access top-level screens and all other features of the application. If the user is not logged in, the application displays **Login screen**, otherwise, if the user has already logged in ones before, the application has saved his credentials and performs an automatic login.

The first screen, a user sees, is the **Reports screen**, which contains a list of reports. In this screen user can perform following actions:

- view detail of a report, this action will navigate to the **Report-detail screen**,
- create a new report, this action will navigate to the **Add-report screen**, where user can switch between forms of two types of report: basic report and initial report,

- edit an existing report, this action will navigate to the **Add-report screen**. In this case, the screen contains a form of a report with filled attributes.
- view a list of saved reports which were not uploaded yet, this action will navigate to the **Saved-reports screen**.

In the second top-level screen, **Settings screen** are located **Logout button** and GPS settings. In the future, if needed, there can be added further settings, for example, user account management.

For a better understanding of application structure refer to the figure below, **Figure 3.1**.

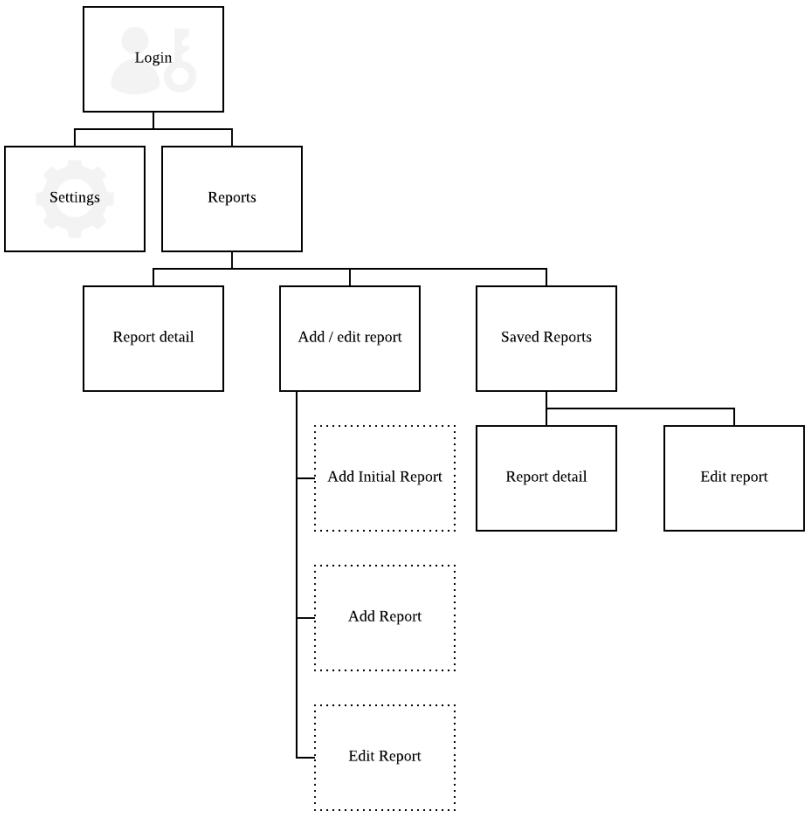


Figure 3.1: Screen diagram of the mobile application

3.3 Use-cases

The application supports one type of user, which needs to be registered and logged in to access the application. In this section are listed main supported operations such as creating, editing, removing and viewing reports.

3.3.1 Create report

When a user needs to create a report, he must decide whether to create a basic report with all attributes or to submit a few notes within simplified report so-called Initial Report.

The application checks network connection after the user submits a form of the report. If a mobile device is online it sends report directly to the server and report is successfully created. Otherwise, the application saves the report to a local database and shows panel with a list of pending reports. When the mobile device is connected again to the network, the user can decide whether to upload the report to the server or to remove it from the mobile application.

Create report scenario

This scenario realizes requirements **FR: 5**, **FR: 8**, **FR: 9**, **FR: 11**, **FR: 12**, **FR: 13**, **FR: 14** and **FR: 15**.

1. User opens **Reports screen** which contains a list of reports and operation buttons such as create new report, edit or remove existing report and filter reports. See **Figure 3.2a**.
2. User presses on **Add report button**.
3. Application navigates to the **Create-report screen**. The screen contains tab bar, which allows to switch forms between two types of report - basic Report (see **Figure 3.2b**) and Initial Report (see **Figure 3.2c**).
4. User chooses the preferred type of report, fills the required attributes, optionally attaches sensory data and clicks on **Save button**.

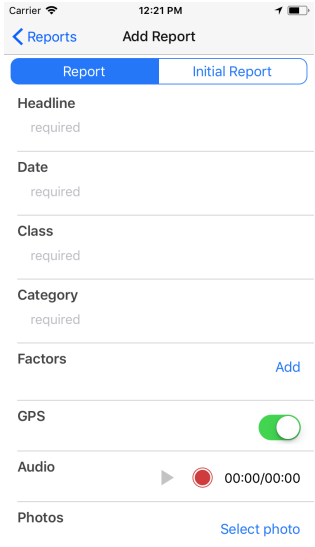
Application is online:

5. Application sends the report directly to the server where the report is saved and refreshes the list of reports.

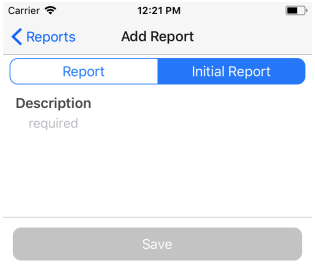
Application is offline:

5. Report is saved to the internal database and application sets event listener for network changes.
6. When the device connects to the Internet, the application notifies the user about pending report.
7. Then user decides whether to upload or delete report and accordingly presses on **Save button** or Remove button in the bottom of the screen (see **Figure 3.2d**). Then the application refreshes the list of reports.

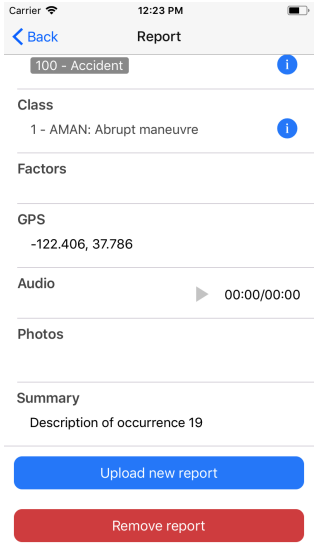
(a) : Reports screen



(b) : Create-report screen - basic report tab



(c) : Create-report screen - initial report tab



(d) : Report-detail screen of saved report

Figure 3.2: Screenshots of the mobile application screens - Reports screen which contains list of reports, Create-report screen which contains form for creating a report and initial report and Report-detail screen of saved report which contains attributes of the report.

3.3.2 View report

A user can preview report with attributes it contains. Besides text attributes such as headline, occurrence category, occurrence class, etc., the application also shows images (if the report contains any) and allows to play voice recording (if the report contains any). The same scenario is applied to both

reports on the server and reports in the local database.

Viewing reports involve finding a report. It can be complicated when the server contains hundreds of reports. For this purpose, the application allows filtering reports by occurrence class and category.

Filter reports scenario

This scenario realizes requirement **FR: 17**.

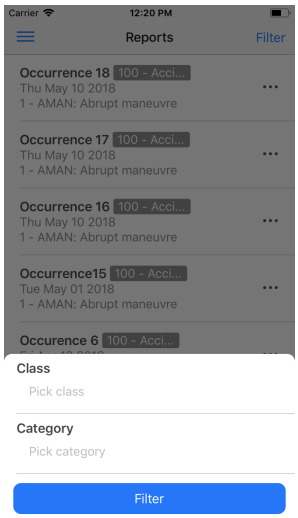


Figure 3.3: Screenshot of the mobile application Filter modal.

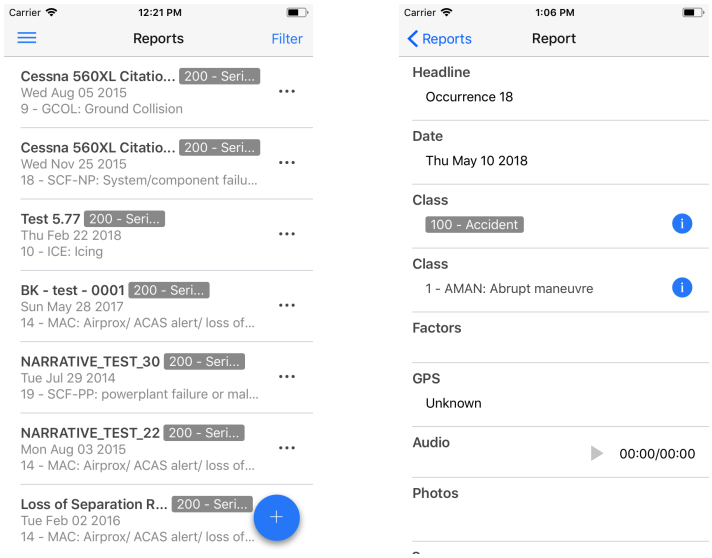
1. User opens **Reports screen** which contains a list of reports and operation buttons such as create a new report, edit or remove existing report and filter reports.
2. User presses on **Filter button**.
3. Application opens modal with form for filtering reports (see **Figure 3.3**). The attributes of the form are occurrence class and category.
4. User picks occurrence class and category and presses on **Filter button**.
5. Application closes the modal and loads a new filtered list of reports from the server.

View report scenario

This scenario realizes requirement **FR: 4**.

1. User opens **Reports screen** which contains a list of reports and operation buttons such as create new report, edit or remove existing report and filter reports. See **Figure 3.4a**.

Application is online:



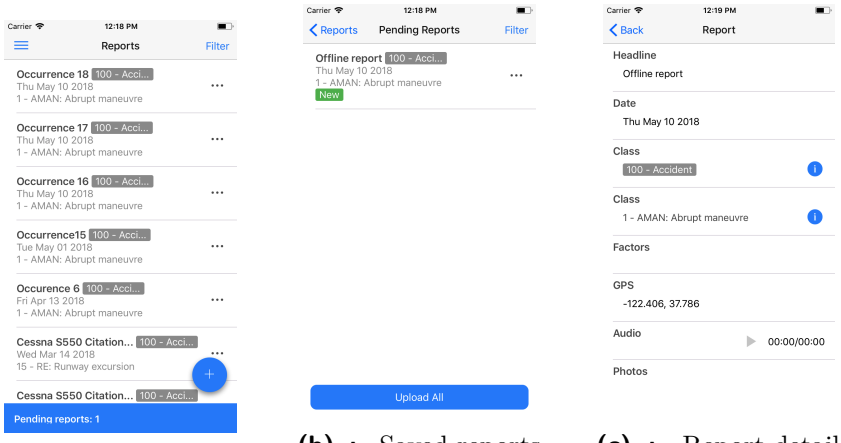
(a) : Reports screen **(b) :** Report-detail screen

Figure 3.4: Screenshots of the mobile application screens - Reports screen which contains list of reports and Report-detail screen which contains attributes of the report.

2. As the application is online, it fetches reports from the server and shows them in the list of reports. User finds desired report and presses on it.
3. Application navigates to the **Report-detail screen**. The screen contains attributes of the report such as headline, date, summary, etc. See **Figure 3.4b**.

Application is offline and local database contains reports:

2. As local database contains reports, in the bottom on the **Reports screen** is located panel, which indicates number of offline reports. See **Figure 3.5a**.
3. User presses on the **Offline reports panel**.
4. Application navigates to the **Offline-reports screen**, where a list of reports is located and operation buttons such as create new report, edit or remove existing report and filter reports. See **Figure 3.5b**.
5. User chooses the preferred report and presses on it.
6. Application navigates to the **Report-detail screen** (see **Figure 3.5c**). The screen contains attributes of the report such as headline, date, summary, etc.



(a) : Reports screen

(b) : Saved-reports screen

(c) : Report-detail screen

Figure 3.5: Screenshots of the mobile application screens - Reports screen which contains list of reports, Saved-reports screen which contains list of locally saved reports and Report-detail screen of saved report which contains attributes of the report.

3.3.3 Edit report

When a user needs to edit a report, it depends whether the report is saved in the server and is accessible by every other user, or the report is located only in a local database.

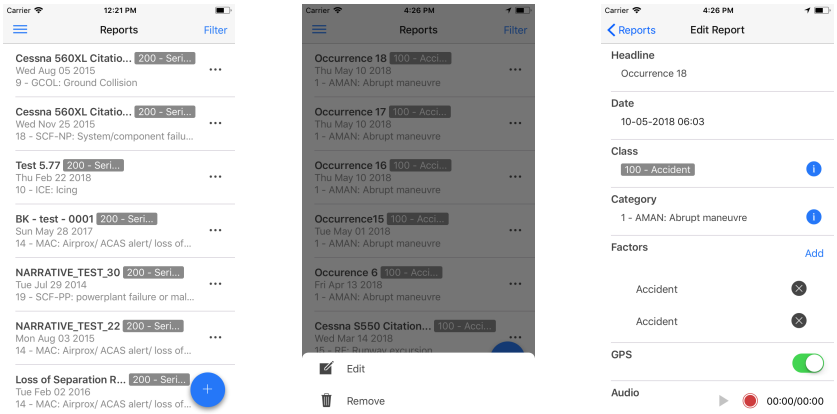
In case the report is saved on the server, the mobile device must be connected to the network or must contain cached reports. If so, a user can find the report in the list of reports, open preview, edit attributes and submit changes.

If the report is saved in the local database, the user always can modify it. The process is the same, the user finds the report in the list of local reports, open preview, edit attributes and submit changes.

Edit report in the server scenario

This scenario realizes requirement **FR: 6**.

1. User opens **Reports screen** which contains a list of reports and operation buttons such as create a new report, edit or remove existing report and filter reports. See **Figure 3.6a**.
2. User selects report and presses on the right side of the row where is located list icon.
3. Application shows modal with two buttons - Edit report and Remove report. See **Figure 3.6b**.
4. User presses on **Edit report button**.



(a) : Reports screen **(b) :** Options menu **(c) :** Edit-report screen

Figure 3.6: Screenshots of the mobile application screens - Reports screen which contains list of reports, Options menu with buttons for editing and removing report and Edit-report screen with form for editing a report.

5. Application navigates to the **Edit-report screen** (see **Figure 3.6c**). The screen contains a form of the report.
6. User modifies attributes of the form and clicks on **Save button**.
7. Application sends the report directly to the server where the report is modified.

Edit report in the local database scenario

This scenario realizes requirement **FR: 6**.

1. User opens **Reports screen** which contains a list of reports and operation buttons such as create a new report, edit or remove existing report and filter reports.
2. As local database contains reports, in the bottom of the **Reports screen** is located panel, which indicates a number of offline reports.
3. User presses on the **Offline reports panel**.
4. Application navigates to the **Offline reports screen**, where a list of reports is located and operation buttons such as create a new report, edit or remove existing report and filter reports.
5. User selects report and presses on the right side of the row where is located list icon.
6. Application shows modal with two buttons - Edit report and Remove report.
7. User presses on **Edit report button**.

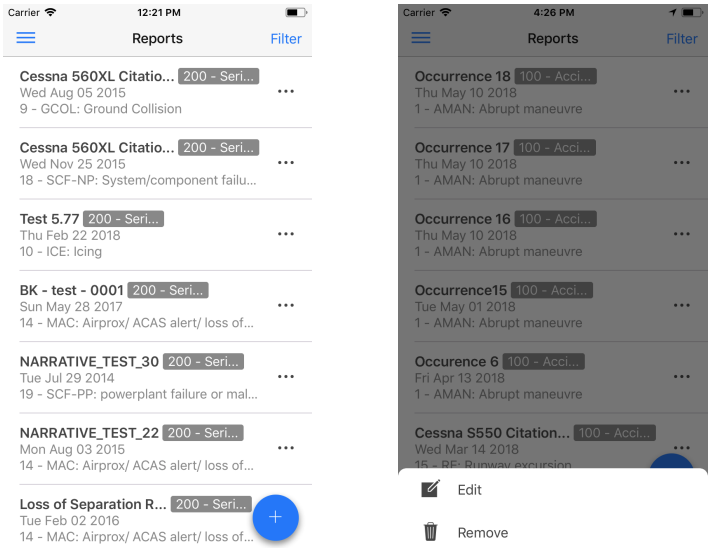
8. Application navigates to the **Edit-report screen**. The screen contains a form of the report.
9. User modifies attributes of the form and clicks on **Save button**.
10. Application saves modified report into the local database.

3.3.4 Remove report

Removing report is very similar to editing, it only requires fewer steps.

Remove report in the server scenario

This scenario realizes requirement **FR: 7**.



(a) : Reports screen **(b) :** Options menu

Figure 3.7: Screenshots of the mobile application screens - Reports screen which contains list of reports and Options menu with buttons for editing and removing report.

1. User opens **Reports screen** which contains a list of reports and operation buttons such as create a new report, edit or remove existing report and filter reports. See **Figure 3.7a**.
2. User selects report and presses on the right side of the row where is located list icon.
3. Application shows modal with two buttons - Edit report and Remove report. See **Figure 3.7b**.
4. User presses on **Remove report button**.
5. Application removes the report from the server and refreshes the list of reports.

Remove report in the local database scenario

This scenario realizes requirement **FR: 7**.

1. User opens **Reports screen** which contains a list of reports and operation buttons such as create a new report, edit or remove existing report and filter reports.
2. As local database contains reports, in the bottom of the **Reports screen** is located panel, which indicates a number of offline reports.
3. User presses on the **Offline reports panel**.
4. Application navigates to the **Offline reports screen**, where a list of reports is located and operation buttons such as create a new report, edit or remove existing report and filter reports.
5. User selects report and presses on the right side of the row where is located list icon.
6. Application shows modal with two buttons - Edit report and Remove report.
7. User presses on **Remove report button**.
8. Application removes the report from the server and refreshes the list of reports.

Chapter 4

Implementation

4.1 Integration with server

Integration with the server is straightforward, as the implementation of the server already exists and mobile application must adapt. The server uses Representational State Transfer (REST) to communicate with a front-end user interface. REST is an architectural style, which defines set of properties and rules for communication over Hypertext Transfer Protocol (HTTP).

The mobile application uses SuperAgent library for interaction with the server. It is a light-weight API that provides a set of methods for retrieving data from the server or sending the data to the server. The methods are based on HTTP methods such as POST, GET, PUT, DELETE, etc.

The most of the data such as reports are transferred (sent or received) within file format called JavaScript Object Notation (JSON), see **Listing 4.1**. The JSON data objects consisting of attribute-value pairs, where the attribute is denoted as a string and value can be any serializable data type such as string, number, boolean, array or nested JSON object.

Besides JSON format, the mobile application also sends data as HTTP form data. Depending on header HTTP standard allows to encode data, suitable for authentication of a user, or to send binary streams, ideal for sending files such as photos or voice recordings.

Listing 4.1: Example of report represented in JSON format.

```
{
  "key":"72643792288465405893",
  "fileNumber":1526655215986,
  "phase":"http://onto.fel.cvut.cz/processed",
  "author": {
    "uri":"http://www.inbas.cz/people#John+Smith",
    "firstName":"John",
    "lastName":"Smith",
```

```
    "username":"smith@fel.cvut.cz",
    "types":["http://onto.fel.cvut.cz/Agent"],
    "enabled":true,
    "locked":false
  },
  "dateCreated":1526655215986,
  "revision":1,
  "identification":"testing1",
  "date":1527087000000,
  "summary":"Safety event"
  "severityAssessment":"http://onto.fel.cvut.cz/v-200",
  "occurrenceCategory":"http://onto.fel.cvut.cz/v-1"
}
```


4.2 Synchronization with server

Communication with the server is one of the main requirements for this mobile application. Besides the technical aspects described in the previous section, it is also necessary to consider synchronisation patterns so the application can provide up-to-date data to the users without noticeable delays and it will prevent creating duplicates.

Before choosing and implementing patterns for data synchronisation, it is essential to understand the purpose of the application, how it should work and what users would expect. In architectural design, there is no universal solution, but rather trade-offs, i.e. different patterns suite different types of applications. So, it is essential to set priorities for various features.

The primary purpose of the application is to provide the users with a possibility to report occurrences everywhere in the workplace. It is doubtful that users will use the application to monitor and inspect existing reports, it would be very impractical, as mobile devices have small screens. Therefore, the application is designed for uploading, rather than downloading. Hence the priority is set for creating new occurrence reports and in the second place is a presentation of existing reports fetched from the server.

4.2.1 Online and offline mode

The mobile application is designed to work in two states, depending on network connection:

- online mode and
- offline mode.

When the application is connected to the Internet, it is considered as the online mode. The application is connected directly to the server, and it

directly downloads and uploads data to the server, i.e. it works the same way as the web application.

During in the offline mode, when the application has no connection to the Internet, it cannot communicate with the server. That makes downloading the data impossible. So, is the application usable in the offline mode? Yes, it is. The application can still create new data, i.e. users can create new reports, which will be stored in the local database, and when the mobile device connects again to the network, it can send saved reports to the server.

Furthermore, during the online mode, the mobile application stores downloaded reports in the temporary memory. These reports are still accessible during offline mode (assuming the mobile application was not closed) and a user can view the reports and edit them (editing during offline mode is explained in the next section). Cached reports along with other temporary data are removed when the application is closed.

4.2.2 Downloading the data

For choosing suitable synchronisation patterns for downloading the data, there is needed to consider following questions:

- when and how often the data needs to be synchronised,
- what amount of data needs to be fetched,
- is it possible to cache the data?

To, minimise unnecessary data synchronisation, most of the transfers are triggered by a user manually. When the user opens the application, it downloads occurrences from the server and then waits for further actions.

The total amount of occurrences in the server can be in the range of thousands and more. Therefore, storing all occurrences in the mobile application would be unpractical and complicated. Instead, it is preferred to load a small number of occurrences and if the user requires, load more. The application is downloading 60 reports in the current implementation.

There is expected that in the production the set of occurrences in the server will vary. Many users will create or modify occurrences. Therefore there is no way to cache them in the mobile application. However, there was indicated that records of different types of occurrence class, occurrence category and factors are mostly static. Hence, there were implemented cache measures: after initialisation, the mobile application will download and store static data in a local database. Then it will monthly check if the static data has changed and if so, it will update its records. Also users are able to download the data manually.

All synchronisations are done as an asynchronous mechanism[2]. That means, a user can use other functions of the application, while the data is loading.

E.g. if the user waits for receiving the list of occurrences, he can still create a new report.

4.2.3 Uploading the data

Uploading the data during online mode works the same way as in a web application. The user can create, edit or remove report and the mobile application will send a relevant request to the server for processing. But in offline mode synchronisation can originate issues regarding collisions.

Imagine a situation when the user has no network connection and creates a report on the mobile application. The report is saved in the local database of the mobile application. Then he comes back to his workplace and creates the same report on the web application that sends it directly to the server (reason can be that the users prefer a desktop over smart-phone and he has forgotten about the already created report in the mobile application). What should the mobile application do with its report in a local database, since there is already another version of the same report on the server? The correct solution is to remove report in the local database, but how mobile application or the server can recognise that report already exists?

It would be complicated, as there is no unique identification. The attributes that come to mind are a time of creation, headline, occurrence category and occurrence class. But, there could be cases when the user creates two very similar reports, but application automatically removes one of them. Hence the final solution is to leave the decision to the user. I.e. mobile application shows locally saved report to the user, who can decide whether to upload or remove it.

The used synchronisation pattern is called Read/Write Data Last Write Wins. This pattern is recommended for mobile applications where it is not likely for multiple end-users to change the same data while the application are offline[3].

The very similar case is with editing report during the offline mode. However, in this case, the user can only edit cached reports, i.e. the reports that were downloaded during online mode and temporarily saved in the mobile application. When user edits report during offline mode, the application stores edited the report in the local database. After mobile device connects back to the network, the user can upload edited report to the server, and that report becomes the newest version of the original report.

Support for removing reports during offline mode seems unnecessary and too complicated. Hence current implementation of the mobile application does not support it.

4.3 Security

For ensuring sufficient security measures of mobile application, one must identify sensitive data, e.g. passwords, personal data, location records, handle passwords on device securely and ensure secure transit of data between device and server.

The only sensitive data in the mobile application are user credentials, i.e. combination of a user-name and a password. The credentials are obtained when the user logs in for the first time and then saved in the mobile application. Each time application is opened, it performs automatic login task, and the user does not need to fill his credentials regularly.

An issue is to choose the right technology for storing credentials. React Native offers AsyncStorage, a simple key-value storage system, and it is an alternative to the LocalStorage which is often used as a storage system in the front-end of web applications. However AsyncStorage is not adequately protected, i.e. it stores data unencrypted. Hence there were evaluated other possibilities and third-party solutions and in the end, was chosen library React Native Keychain developed by a community. The library provides access to Android keystore and iOS keychain (secure storage systems for storing cryptographic keys, passwords, etc.), the credentials are encrypted, and the key is non-exportable, this means that the system will not allow exporting key to a transportable file that can be copied on another device. Therefore it makes an entire process more secure.

Note: The Android keystore is available on API level 23+ and is considered as secure storage. On older levels the data are stored in SharedPreferences, a simple key-value storage system for Android, and encrypted / decrypted with Facebook Conceal, a set of Java APIs to perform cryptography. In this case, authors of React Native Keychain library warns that the key may be obtained when the Android device is rooted or if an attacker can somehow access the filesystem.

For secure exchange of credentials between the mobile application and the server is used protocol HTTPS, which ensures secure transfer of data. HTTPS is an extension of the Hypertext Transfer Protocol (HTTP) for secure communication over a computer network.

4.4 Testing application

Automated testing is a useful addition to applications. Developers or testers design automated tests to validate individual functions of the application. Tests make implementation process faster and less faulty, as the developer does not need to validate some functionalities manually and it reduces human error.

In the mobile application, automation testing is implemented with library Jest. As every testing software, it allows to execute parts of an application and compare actual outcome with predicted outcomes.

In the application are automatically tested user interface, i.e. if the view renders correctly, and controller functionalities, e.g. management of reports or management of application preferences.

For testing user interface was used Snapshot testing. It is a feature of Jest that allows to create a human-readable snapshot of a component and compare it with a previous snapshot. If the component has changed, the test will recognise it and alert developer. In the mobile application is tested each elementary component (i.e. button, label, etc.) and each screen (i.e. login screen, reports screen, etc.).

Testing controller methods require mocking some functionalities, like a local database or a filesystem of a mobile device, to prevent the creation of permanent changes. The Jest library allows mocking either individual functions or whole modules.

Chapter 5

Best practices

At the beginning of developing an application, one has a general picture of application structure. But before seeing the details, it is not clear, whether the main idea contains any flaws and whether the design should be modified.

One can refer to sources on the Internet, where the community describes best ways of implementation and usage of particular technology. However sometimes described possibilities vary and it is not clear what is the best for specific case. Therefore one must expect possible changes.

During the development of this mobile application, there were done many architectural modifications and were discovered some best practices, which can help make a better decision right from the beginning.

5.1 Memory leaks

Before starting developing an application one of the first decisions was whether to use Redux. Due to the simpleness of application, it seemed not necessary. So, in the first design, the global state (i.e. reports) has been stored in the root component of the screen, the application contained a pointer to the component, and it could update the state when needed. It was all done simply and minimalistic to prevent errors and unnecessary code. Besides the state in the component, there were implemented a simple caching system and an observer object[8] - a software design pattern in which an object notifies subscribers when the data changes.

This concept worked to the point when it came to switching between screens. When the screen changes, the screen component is unmounted and the garbage collector cleans all data of the component. The garbage collector is a mechanism for automatic memory management, it reclaims memory occupied by objects that are no longer in use by the program. However if the application still contains a pointer to this unmounted component, it can not be cleaned. That leads to memory leaks, which cause impaired

performance or failure. Also when the component is unmounted, and the pointer still exists, other parts of an application can try to change the state of it. That is a large anti-pattern described by Facebook, which can lead to the impaired functioning of the application. That was the exact case of the first design.

To fix the memory leaks, there needed to be implemented more complex layer over the screen components, including more observer objects, which would track the data changes. But it was already implemented, these mechanisms are part of one global state container called Redux.

Using Redux, memory leaks were solved, and the code became more transparent. In conclusion, this practice has shown that the Redux (or other library implementing global state pattern flux) is a crucial part of almost every React or React Native application.

5.2 Navigation

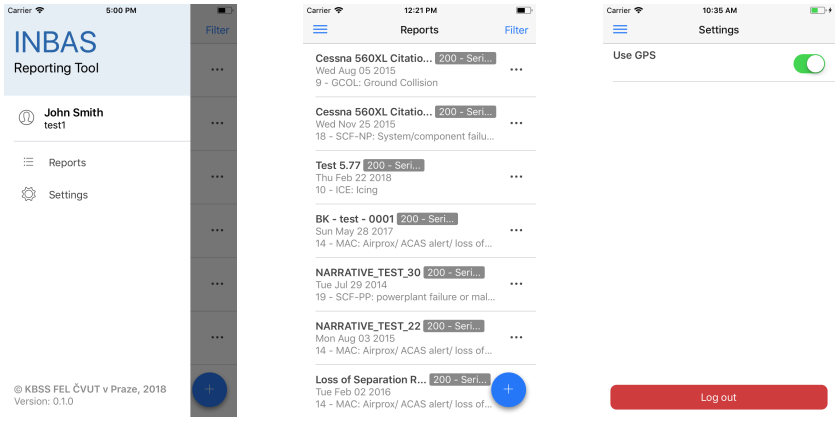
With React Navigation, a library which contains a few already implemented navigators with a navigation system comes to a question, which navigator to use. The first choice is a Stack navigator since it contains a header (a typical part of every mobile application) with already implemented back button and a title of the screen. It allows forward and reverse navigation within one part of the application, e.g. in the screen with a list of reports application can navigate to the screen with full detail of a chosen report.

The user interface structure was designed to contain two top-level screens, for this is recommended lateral navigation[11]. Therefore there should also be used either a Tab navigator or a Drawer navigator.

The decision was to use the Drawer navigator, since the drawer can contain more information, not only the buttons but also information of user or application, etc.

The navigation design of the mobile application was based on an example of multiple navigators usage in the documentation of React Navigation library[12]. The Drawer navigator**Figure 5.1a** was the root component which was switching between two screens - screen with reports**Figure 5.1b** and screen with settings**Figure 5.1c**. Each screen was a Stack navigator which contained further screens. However, this approach was too complicated. The application needed to store pointers of two navigators, one for switching between reports and settings and another one for navigating within this screens.

In the end, it was decided not to use Drawer navigator and use only one Stack navigator with a custom implementation of the drawer. The user interface design was not changed, just the code architecture. This approach has paid off, and the application code became more straightforward and more



(a) : Drawer **(b) :** Reports screen **(c) :** Settings screen

Figure 5.1: Screenshots of two main screens of the mobile application - Reports screen and Settings screen, and a Drawer for navigation between those screens.

transparent. Furthermore, the application creates fewer objects, which means better performance.

In development, especially with JavaScript language, there are many libraries which provide many functionalities. But not always it is a right decision to use every available feature.

5.3 Stateless components

At the beginning of development, there were used only classical React Components. It has seemed like an only way to define components. Even though the current version of React library documentation promotes usage of Functional Stateless React Component[13], a lot of examples in React Native documentation and Learning React Native book uses classical React Components[1][14].

Classical React Component created as JavaScript class contains functionalities which are not always required. A typical example is a State and component Lifecycle function. If an only reason of React Component is to be rendered as a fundamental element, such as a button or a text, which does not contain any logic inside, it should not carry any State. For these reasons, React allows creating a so-called Functional Stateless React Component, which is created as a JavaScript function. The component does not contain constructor nor Lifecycle functions, and it offers potential performance optimisation.

After some time, the decision was made to utilise Functional Stateless React Components. The performance improvements were not visible, but the modification made a code clearer. For the beginner, who is exploring the library, it might seem not that important, but for the developer, who is

creating an application with a potential to be used in the future, every such detail is significant.

That might be an evident approach, but still, there are some React or React Native applications with every React Component implemented as a JavaScript class.

Chapter 6

Usability testing

Usability testing is a technique to evaluate a product by testing it on users. It is one of the requirements for this thesis.

When the core features of the mobile application were finalised, the thesis supervisor had handed over the application to the four potential users (testers).

Testers were required to complete tasks to determine a quality of the user interface and whether all features are intuitive, easily accessible, work correctly and fast enough. The tasks are following:

- Open detail of a report
- Filter list of the reports
- Create a report
- Create a report during offline mode - Turn off wifi and mobile data and then create a report. Then open detail of that report.
- Logout

After testing the application, the testers filled out a provided form which contains 4 question. The results of questions 1 and 2, that are listed in the **Table 6.1** and the **Table 6.2**, are positive. The answers indicate that the user interface is intuitive and the response time is satisfactory.

On the other hand, the testers had an issue with crashing of the application. See the answers to question 3 that are listed in **Table 6.3**. The issue should be traced and solved in the future work.

At last, the testers have suggested recommendations to improve the application. See the answers to question 4 that are listed in **Table 6.4**.

Note: The usability testing was done at the end of the project. Thus there was no time to solve the reported issues and process the recommendations.

Was every task easy to accomplish? If not, which task and what difficulties were there?	
Tester 1	Yes, the application is fairly straightforward, intuitively usable. No tasks were really problematic.
Tester 2	Every task was easy to accomplish (except in the case below - question 3).
Tester 3	Each task was tested and it was easy to accomplish. It was intuitive and responded as expected.
Tester 4	All tasks are OK, except of log out – it was impossible

Table 6.1: Usability testing survey - question 1

Is the application fast enough?	
Tester 1	Yes, no problems were identified regarding the application speed.
Tester 2	Yes, it is.
Tester 3	Application responded without problem. Response time is satisfactory. I tested it few times and noticed a slight differences in response time, however it never freeze or crashed.
Tester 4	Yes, except the issue below (question 3)

Table 6.2: Usability testing survey - question 2

Does the application crash, hang or freeze?	
Tester 1	It does crash occasionally when browsing details of a report, but at all times when I want to record audio in a new report or do so whilst editing an existing report. Sometimes, when I click details of a report, the screen goes blank and the app needs to be restarted.
Tester 2	The application repeatedly crash during the factor addition step. (a report creating, the airport public wifi connection)
Tester 3	During the test application did not crashed, hanged or freeze.
Tester 4	During saving occurrence, after clicking on save, there were still the “working sign” but the application don’t go to other window, I have to do it manually by clicking on “back arrow” During saving, there is word “Loading” it looks strange

Table 6.3: Usability testing survey - question 3

Do you have any recommendation for the application?	
Tester 1	In a new report, the factors added are all organized in parent-child fashion when displayed in web-based RT to edit the chain. The first added factor is parent of the second, the second is parent of the third etc. All factors should be equal, none should be parent of any other. Photos can be added in the application, but not viewed (edited) afterwards. Maybe the photos are not saved, but the latter feature would definitely help. Interestingly, for offline reports in preview of a pending report, attached photos are visible and editable, even though full-size picture cannot be displayed to check individual pictures.
Tester 2	—
Tester 3	It is possible to set a date in future as a date of event. The classic INBAS reporting tool controls whether the date is set appropriately and it prevents setting a date in future.
Tester 4	For users in field, it could be good, if there would be possible to fill in other basic information like aircraft registration, type of aircraft etc.

Table 6.4: Usability testing survey - question 4

Chapter 7

Conclusion

7.1 Conclusion

The purpose of this thesis was to implement Mobile Application for Safety Occurrence Reporting which is intended to be a part of a web application called Reporting Tool.

Before the beginning of development, there were analysed and evaluated approaches and technologies for mobile application development. In the end, was chosen the hybrid mobile application implementation approach with React Native framework designed by Facebook.

As the mobile application has to cooperate with the server, to store and retrieve occurrences and user data, there were needed to analyse possibilities of integration with the server. At last, the integration was adapted to the server and was used an architectural style REST.

One of the requirements for the application is a possibility to work in offline mode, which can cause a corruption of the data (e.g. the creation of duplicates). To prevent that there were analysed and utilised synchronisation patterns.

During the implementation was several times needed to modify the structure of the application. The whole process of implementation has led to a discovery of useful practices, for example, utilisation of a Flux architectural pattern.

In the end, the description of the mobile application, applied approaches and discovered outcomes were written into this document. And the mobile application was passed on to potential users for a usability testing.

In the future future work there would be implemented requirements such as support for attaching video records to reports and support for creating Initial reports during offline mode.

7.2 Final impression

When trying new technology (new framework) developer typically creates 'hello world' demo, which contains a simple implementation of major features without any meaningful purpose of the application. In this case, everything seems to work, and if it does not, it is probably the mistake of the developer, who uses the technology in a wrong way, because of a lack of knowledge.

However, when the developer approaches the end of implementation, and the application contains functionalities which real users might use, the impression is different. First of all, the developer already possesses more profound knowledge about the technology, he has read literature, browsed through community forums and has solved numerous bugs in his program. Thus when there is a strange behaviour with no possible cause of application, it might be an error of the framework.

During the implementation of this mobile application, there were several cases of bugs in the React Native framework. The most of them were already solved in newer versions. Hence an update was a solution.

But there are still unresolved issues. When using an iOS simulator, the application behaves flawlessly. On the other hand, when running the application on an android device, there are slowdowns in animations, e.g. while opening modal, and occasional freezes of a view. These minor issues do not prevent using the mobile application, as all functionalities work, but sometimes it might be irritating for the users.

During this project, which has lasted approximately for a year (including the period of learning the framework), there were released several new versions of React Native and React. In this updates were introduced better UI components and implementation approaches and there were solved bugs. In the future, hopefully, the creators of React Native framework will resolve issues with animations.

Also, there was reported an issue of application crashing by the testers. As the crashes were observed only on some devices it may be caused by different versions of operating system and it does not have to be an error of the React Native. This issue should be traced and resolved in the future work.

In the end, I do not regret my decision of choosing React Native. The negatives described above are little, and other frameworks may have even more significant flaws. Furthermore, I am convinced, that the performance optimisation may be applicable and animation issues can be solved (probably not in the JavaScript code, but in the native code, i.e. Swift and Java).

Bibliography

[1] Bonnie Eisenman. *Learning React Native: Building Native Mobile Apps with JavaScript*. O'Reilly Media, 2017.

[2] Zach Mccormick and Douglas C. Schmidt. *Data synchronization patterns in mobile application design*. [*In Proceedings of the 19th Conference on Pattern Languages of Programs (PLoP '12)*]. The Hillside Group, 2012.

[3] OutSystems (2018, May 21) OutSystems 10 documentation. Retrieved from `https://success.outsystems.com/Documentation/10/Developing_an_Application/Use_Data/Offline/Offline_Data_Sync_Patterns`

[4] Bogdan Kostov, Jana Ahmad and Petr Křemen. *Towards Ontology-Based Safety Information Management in the Aviation Industry*. [*On the Move to Meaningful Internet Systems: OTM 2016 Workshops*]. Springer International Publishing III, 2017.

[5] European Network and Information Security Agency. *Smartphone Secure Development Guidelines for App Developers*. [*On the Move to Meaningful Internet Systems: OTM 2016 Workshops*]. Springer International Publishing III, 2011.

[6] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, 2017.

[7] Alex Banks, Eve Porcello. *Learning React*. [*Functional Web Development with React and Redux*] O'Reilly Media, 2017.

[8] John Vlissides, Ralph Johnson, Richard Helm, Erich Gamma *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[9] Facebook Inc. (2018, May 1) Flux - Application architecture for building user interfaces. Retrieved from `https://facebook.github.io/flux/`

[10] Realm (2018, May 1) Documentation of Realm Database. Retrieved from https://realm.io/docs

[11] Google LLC (2018, May 14) Material design. Retrieved from https://material.io/design/navigation/understanding-navigation.html

[12] A. Tevosyan, B. Vatne, D. Pack, K. Magiera, M. Hamil, A. Miskiewicz, S. Carli (2018, May 14) Documentation of React Navigation. Retrieved from https://reactnavigation.org/docs/en/tab-based-navigation.html

[13] Facebook Inc. (2018, May 14) Documentation of React. Retrieved from https://reactjs.org/docs/components-and-props.html

[14] Facebook Inc. (2018, May 14) Documentation of React Native. Retrieved from https://facebook.github.io/react-native/docs/getting-started.html

Appendices

Appendix A

Building mobile application

Below are described steps for building and running the mobile application on iOS simulator or device and Android emulator and device.

A.1 Android

For building a React Native application on Android emulator or device are required following dependencies:

- Node
- the React Native command line interface
- Python2 (for Windows)
- Watchman (for macOS)
- JDK
- Android Studio

For installing Node, Python2, Watchman, a JDK and Android Studio, please refer to React Native documentation.

React Native CLI is installed by running following command in a Comand Prompt or shell:

```
npm install -g react-native-cli
```

Then in a Command Prompt or shell change directory to the mobile application source code directory and run following command:

```
react-native link
```

At last, to run the mobile application connect the mobile Android device to the computer via USB cable or start an Android emulator in Android Studio and run following command:

```
react-native run-android
```

A.2 iOS

Building application for iOS is possible only in macOS operating system, which needs to have installed following dependencies:

- Node
- the React Native command line interface

For installing Node, please refer to React Native documentation.

React Native CLI is installed by running following command in a shell:

```
npm install -g react-native-cli
```

Then in a shell change directory to the mobile application source code directory and run following command:

```
react-native link
```

At last, to run the mobile application connect the mobile iOS device to the computer via USB cable (in case of running the application on iOS Simulator there are no other steps needed) and run following command:

```
react-native run-ios
```