Politechnika Wrocławska

Faculty of Computer Science and Management

Field of Study: Computer Science

Bachelor's Thesis

Mobile application with gamification elements

Michał Karbownik

Keywords:

Gamification, Mobile Application, Flutter

Short Summary :

The goal of this paper is to minimize possible harms and maximize benefits of using smartphones by children by providing a mobile application that utilizes gamification elements. The proposed solution is a result of broad market analysis, followed by a thorough planning process. It is implemented using modern techniques and tested in real-life conditions.

Supervisor	Ph.D. Krzysztof Waśko		
	Title/ degree/ name and surname		
The final evaluation of the thesis			
Chairman of the Diploma Examination Committee	.....	.....	.....
	Title/ degree/ name and surname	*grade*	*signature*

*For the purposes of archival thesis qualified to:**

a) Category A (perpetual files)

b) Category BE 50 (subject to expertise after 50 years)

** Delete as appropriate*

stamp of the faculty

Wrocław, 2020

Acknowledgments

This work could not have been accomplished if not for the support I received from many great people.

First and foremost, I would like to express my deepest gratitude to my girlfriend, Lucie Vitáková, whose bright soul and mind shine on my days, even in the moments of greatest despair.

I am sincerely thankful to my parents for their love and sacrifice to give me the best in life. Special thanks belong to my beloved sister for the smile she bestows upon the world, as well as other members of my family, whose support I feel despite the distance.

I would like to thank PhD Krzysztof Waśko for his enthusiastic encouragement and advice. I would also like to mention all of the teachers and mentors that I had a privilege to meet over the course of my education, in particular from *I Społeczne Liceum Ogólnokształcące im. Zbigniewa Herberta* and *Społeczne Gimnazjum nr 2 im. Zbigniewa Herberta* in Częstochowa, for their passion and wisdom, which they always eagerly shared.

Acknowledgments go to all my friends from the university and high school. Their humour and kindness has been an invaluable asset.

Finally, I would like to thank all the people of good will that I crossed paths with.

Abstract

Recognising “Gaming disorder” as an addictive behaviour by the World Health Organisation in 2018 has sprung conversations about the use of technology by children, its threats and opportunities. The former is usually associated with loss of productivity and significant impairment in family, social and educational areas of functioning. One of the most frequent keywords in the context of the latter is *gamification*.

The goal of this paper is to minimize possible harms and maximize benefits of using smartphones by children by providing a mobile application that utilizes gamification elements. The proposed solution is a result of broad market analysis, followed by a thorough planning process. It is implemented using modern techniques and tested in real-life conditions.

The created application allows parents and children to engage in a gamification process of establishing tasks and rewards to increase productivity. It has a research-based functionality and user interface. The final product has been distributed among and assessed by the end-users.

Streszczenie

Uznanie “zaburzeń growych” jako zachowania uzależniającego przez Światową Organizację Zdrowia w 2018 roku wywołała dyskusje dotyczące użycia technologii przez dzieci, jego zagrożeń oraz możliwości. Pierwsze są zazwyczaj związane z utratą produktywności oraz znaczące osłabienie funkcjonowania w rodzinie, w społeczeństwie oraz edukacji. Jedno z najczęstszych słów kluczowych w kontekście tych drugich to *grywalizacja*.

Celem tej pracy jest zminimalizowanie możliwych szkód i zmaksymalizowanie korzyści korzystania z telefonów przez dzieci poprzez dostarczenie aplikacji mobilnej, która zawiera elementy grywalizacji. Proponowane rozwiązanie jest rezultatem szerokiej analizy rynku, wraz z gruntownym procesem planowania. Implementacja wykorzystuje nowoczesne techniki i jest przetestowana w warunkach rzeczywistych.

Wytworzona aplikacja pozwala rodzicom i dziecom na zaangażowanie w proces ustalania zadań i nagród w celu zwiększenia produktywności. Zarówno funkcjonalność, jak i interfejs użytkownika są poparte badaniami. Końcowy produkt został rozdystrubowany do użytkowników końcowych oraz przez nich oceniony.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

1.1 Background

Whenever I talk to someone about the topic of my thesis, I receive a confused look followed by a question:

> I understand the *mobile application*, but what is *gamification*?

As an answer, I proudly quote the most popular and widely accepted definition, which I know by heart already:

> It is the use of game design elements in non-game contexts [14].

The look of confusion on my counterpart's face usually grows even bigger, so I quickly follow up with a simpler explanation:

> You certainly like playing games. There are mechanisms that cause it to be fun and make you want to play it. You can play with your friends, collect points and claim rewards - those are *game design mechanisms*. And *non-game contexts*? It is essentially everything but a game and using these mechanisms in every place outside of a game - may it be sports, productivity, or even health.

A focused frown is immediately replaced with a self-satisfied smile. I hope it also came to The Reader's face.

One of the earliest applications of gamification reaches back to the early 1910s when the Scouts movement was found. Since then, their flagship programme was to reward young adepts with badges for various activities. Moreover, it is a flagship example of gamification itself, long before the definition was formed [9, 13]. A few decades later, in 1973, Charles A. Coonradt noticed that in sports, unlike in the United States' workforce, productivity and teamwork was blooming. His attempts to transfer the same strategies to the business environment resulted in a methodology called *The Game of Work* (which he later described in a book with the same title [12]). This was ground-breaking, as proved that gamification is not only a way to entertain children, but can also be profitable in the commercial world.

At present, there are countless examples of effective gamification implementations, not only in business [28] or crowdsourcing [40], but also in healthcare [47] and, most importantly in the context of this thesis, child development [37, 42].

1.2 Motivation

"Gaming disorder". The name was given to an addictive behaviour related to gaming by the World Health Organisation in 2018[59]. It is "characterized by impaired control over gaming, increasing priority given to gaming over other activities to the extent that gaming takes precedence over other interests and daily activities [...]". It results in "significant impairment in personal, family, social, educational, occupational or other important areas of functioning [...]". In 2019 Society for Research in Child Development reported that in United Kingdom 66% of children aged 5-7 regularly play games for around 7.5 hours a week and around 80% of those aged 8-15 play for at least 10 hours a week [31].

Newzoo, one of the most reliable sources of the game market analytics, estimates that in 2020 the game industry revenue will reach $160 billion, almost half of which will be gaming on mobile devices. In consequence of the 10% year-to-year growth, by 2023 annual revenue will have exceeded $200 billion worldwide [58]. And COVID-19 pandemic is only increasing these numbers.

Such statistics make one appreciate the scale of the problem and its impact on the young generation. It also provokes thoughts about how the situation could be improved. If only there was a way to copy the mechanisms that cause the games to be so appealing and use them to motivate children to be more productive...

I hope the smile comes back to The Reader's face now as they remember the solution described in the previous paragraph - *gamification*. And probably the best way to implement it is by using the most popular medium among youth - mobile devices. Therefore, the topic of this thesis - "Mobile application with gamification elements".

1.3 Goal

The goal of this thesis is to develop a project of a mobile application with gamification mechanisms, to implement a prototype of a working product and verify it.

1.4 Scope of Work

Chapter 1 introduces to the subject of this work (1.1, describes motivation behind it (1.2, defines its goal (1.3) and scope (1.4).

Project assumptions are presented in Chapter 2. It states the problem that the work will be trying to solve (2.1), its target group (2.2), core features (2.3) and platform (2.4). It also defines a glossary of terms used in the project ((2.5).

A market analysis is performed in Chapter 3. Firstly, it provides background information (3.1) and moves on to an overview of existing solutions (3.2). In the end, conclusions are drawn (3.3).

Chapter 4 forms the project requirements. The subject of the project is defined (4.1),

along with the users (4.2), functional (4.3) and non-functional requirements (4.4).

Chapter (5) focuses on creating a project of the future solution. An introduction (5.1) is followed by use case modelling (5.2), user interface design (5.3) and planning out a database solution (5.4). The project is concluded in the Summary section (5.5).

Tools, technologies and techniques are chosen in Chapter 6. After a short introduction (6.1) a cross-platform framework is chosen (6.2). The system's architecture is depicted in Section 6.3 , followed by services description in Section (6.4). A project management software (6.5), version control system (6.6) and static code analysis tools are chosen in the following sections.

Chapter 7 aims to explain implementation aspects. After an introduction (7.1), development environments and tools are described (7.2). Section 7.3 overviews a high-level architecture. After implementation challenges (7.4) and installation (7.5) are described, the chapter is summarized (7.6).

Ways of testing the system are presented in Chapter 8. An introduction (8.1) provides with entry information needed to understand further sections. Unit (8.2), widget (8.3) and integration (8.4) tests are supplied with example code. Cross-device tests (8.5) include execution screenshots from different devices. User Acceptance tests are conducted in Section 8.6 and the chapter is summarized in Section 8.7.

Chapter 9 contains the final results (9.1), further work suggestion (9.2) and conclusions (9.3).

Chapter 2

Project Assumptions

2.1 Problem Statement

As children's productivity declines nowadays and current methods of boosting it are not sufficient any longer, parents need a simple tool to manage their tasks and reward them.

2.2 Target Group

2.2.1 Children

The primary target group are children. Their age range is between 5 and 12. Younger children might not be self-aware and independent enough to complete tasks. The older ones might find the concept boring and childish.

2.2.2 Parents

As a natural consequence, the other target group are parents. The age range restriction is unnecessary; however, they should be comfortable using mobile devices.

2.3 Core Features

Main features of the application should be:

- Parents being able to set up an account.
- Parents being able to manage multiple children.
- Parents being able to manage children's tasks and rewards.
- Children being able to view their tasks and rewards.

2.4 Platform

The application should be targeted for mobile devices; however, with consideration of a future expansion to other platforms (web in particular). Target devices are smartphones

with Android operating system, nevertheless, it would be desirable that it could be easily adapted to iOS operating system and tablets.

2.5 Project Glossary

- *Parent* - The person responsible for children as well as managing their tasks and rewards.
- *Child* - The person designated by the parent to be a recipient of tasks and rewards.
- *Task* - A piece of work defined by a parent to be undertaken by a child.
- *Reward* - Something given in exchange for completed tasks, not necessarily a material thing.
- *Point* - A numeric score assigned to every task and every reward. It is defined as an asset in case of the former and as a cost in case of the latter.
- *Artifact* (also *Gamification Artifact*) - Game design element that attempts to invoke certain motivations, e.g. *Task*, *Reward*, *Point*.
- *User's (Child's / Parent's) perspective* - layout and functionality corresponding with the user's role (a parent or a child).
- *Device* - any device that runs the application.
- *Device Belonging* - device assignment corresponding with the current user's role (a parent or a child). It defines the user's perspective.
- *Account / Parent's Account* - an account identified by email and password, one per family, usually belonging to a parent; it is an administrative unit and can manage multiple profiles.
- *Profile / Child's Profile* - a separate unit with its own points, tasks and rewards, usually belonging to a child; it belongs to exactly one account.

Chapter 3

Market analysis

3.1 Introduction

The global market for educational application for children is extremely challenging. With nearly $400 million of revenue in the year 2017 and estimated $1 billion in 2022 [8] (which equals to around 22% Compound Annual Growth Rate) it is one of the fastest-growing categories. Hence, it is almost impossible to analyse every available application. Section 3.2, however, will sum up advantages and disadvantages of a few examples of possible direct competition. Section 3.3 will summarize and conclude the analysis.

3.2 Overview of Existing Solutions

All the gathered information is based on official Google Play (Apple App Store if not available) documentation and, if possible, physical installation of an app. Devices used to test applications were Samsung Galaxy A50 (Android, version 10) and Apple iPad Pro 2020 (iOS, version 14.2). Screenshots are taken on an emulated LG Nexus 5X (Android, version 10) or mentioned iPad.

3.2.1 iRewardChart

Application *iRewardChart*[1,2], originally released in 2009, was one of the first on the market. It is available both on Android and iOS platform. Its target group are children aged three and older. It has more than 10,000 downloads on Google Play, but its average rating is 2.7 (on a scale from 1 to 5). Application's size is 7MB. It can be run on a device with Android 4.1 and higher. Its last update was in October 2017, and the project website[3] is not reachable, which suggests it not being supported anymore.

iRewardChard is rather simple to use, yet its design is outdated. Its main screen is a week table view with particular tasks represented as stars. Tasks can be marked with multiple yellow (positive) and red (negative) stars, which can be redeemed for

[1]https://play.google.com/store/apps/details?id=com.gotclues.irewardchart.full (accessed Nov. 30, 2020)

[2]https://apps.apple.com/app/irewardchart-chore-tracker/id341306389 (accessed Nov. 30, 2020)

[3]'iRewardChart project page'. www.irewardchart.com, (accessed Nov. 30, 2020)

rewards. The software has a feeble user experience (including occasional errors presented to the user) and gives an impression of an expense tracker more than a family application. It allows for tasks and rewards customisation, as well as multiple children management, though only in the paid version. It lacks a child's perspective as well.

Example screenshots of the *iRewardChart* application are presented in Figure 3.1. A summary of its positives and negatives was presented in Table 3.1.

Advantages	Disadvantages
Available on both platforms	Low Google Play rating
Small application size	Not updated for a long time
Older Android versions support	Cumbersome design
	Poor user experience
	Not many options in the free version
	No child's perspective

Table 3.1: *iRewardChart* application advantages and disadvantages

3.2.2 ChorePad

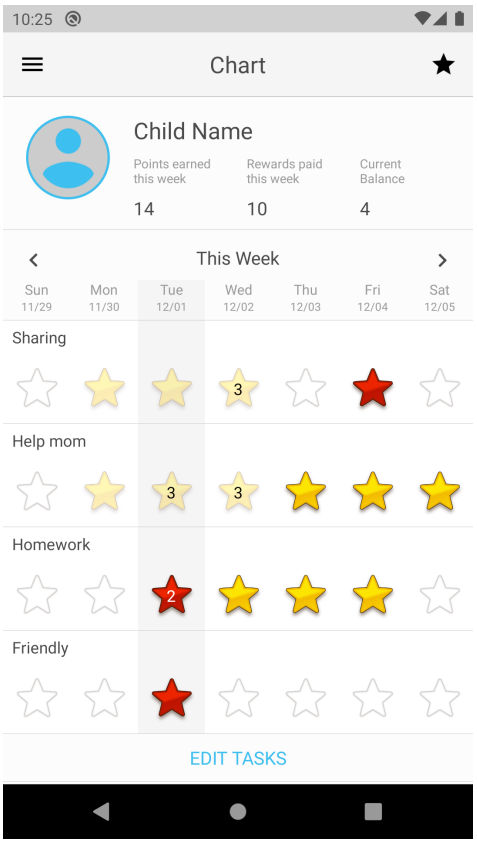
ChorePad[4,5] has two different versions, paid and free, distributed as separate applications. It targets children aged four and older. The application supports only Apple devices with iOS version 9.0 or later. Apple App Store does not share the downloads number or the last update date, therefore, it is difficult to estimate it. The free version has a rating of 2.9 and the paid one - 4.4. Both versions have a size in range 60-70MB.

The application has a child-friendly, though occasionally confusing design. Despite provided onboarding mechanisms, it is not always straightforward to use. Its main screen represents a board with children profiles, with only one available in the free version. *Chores*, because this is the name of tasks, are the main focal point in the application (however, only four are available in the free version). Rewards functionality, even if available, is marginal. Application has a child's perspective; however, it does not change elements visibility, but only editing functionality.

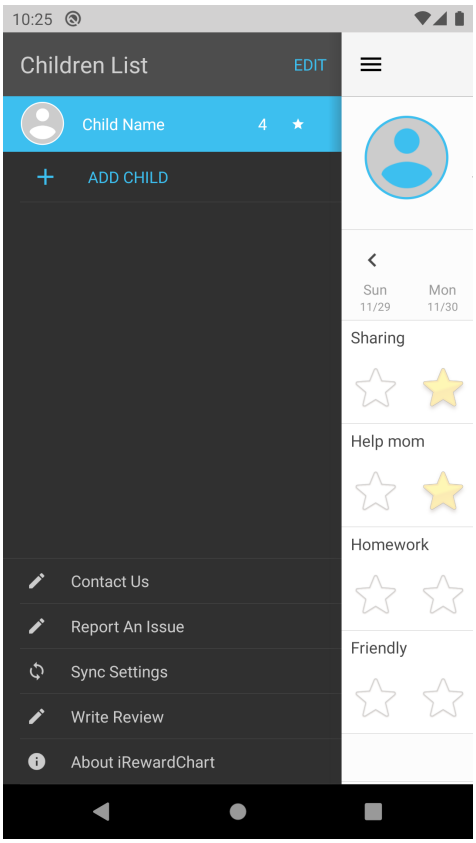
Example screenshots of the *ChorePad* application are presented in Figure 3.2. A summary of its strengths and weaknesses was presented in Table 3.2.

[4]https://apps.apple.com/app/chore-pad-lite-chores-rewards-with-themes/id408265556 (accessed Dec. 2, 2020)

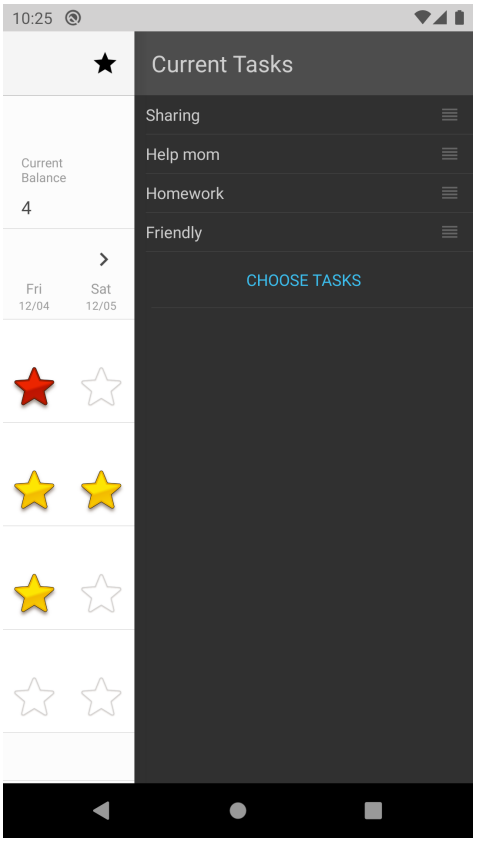
[5]https://apps.apple.com/app/chore-pad-chores-rewards-beautifully-themed/id384854237 (accessed Dec. 2, 2020)



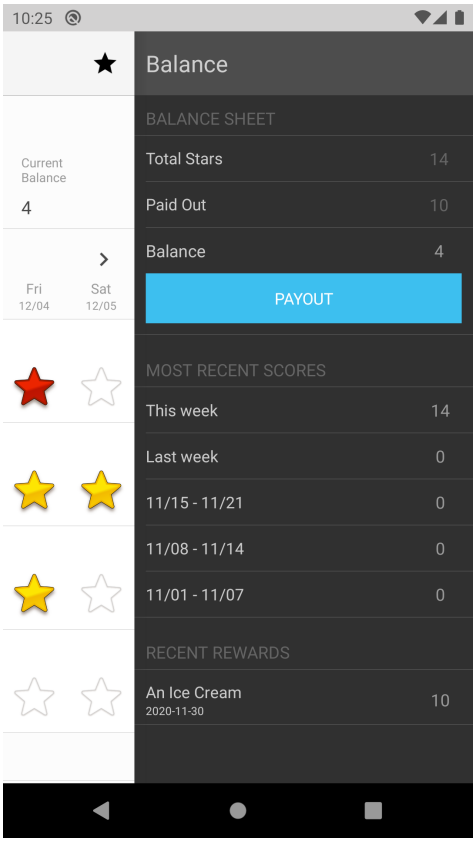
(1) Calendar view (main screen)



(2) Children list

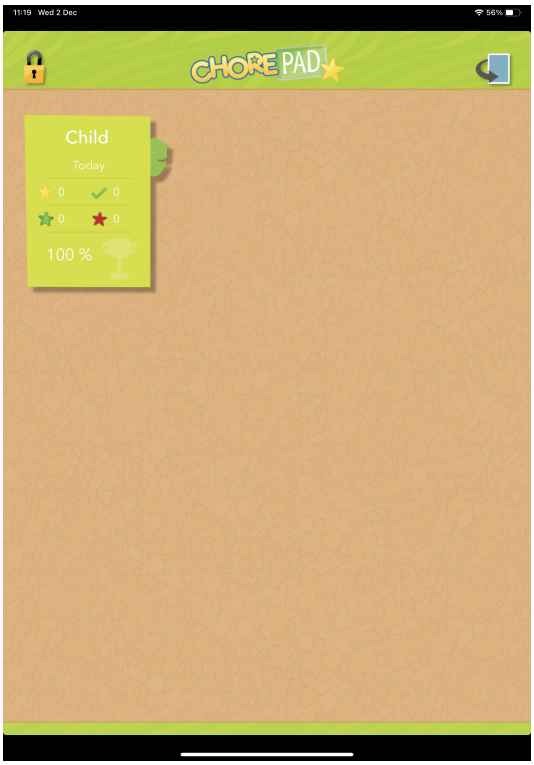


(3) Tasks list

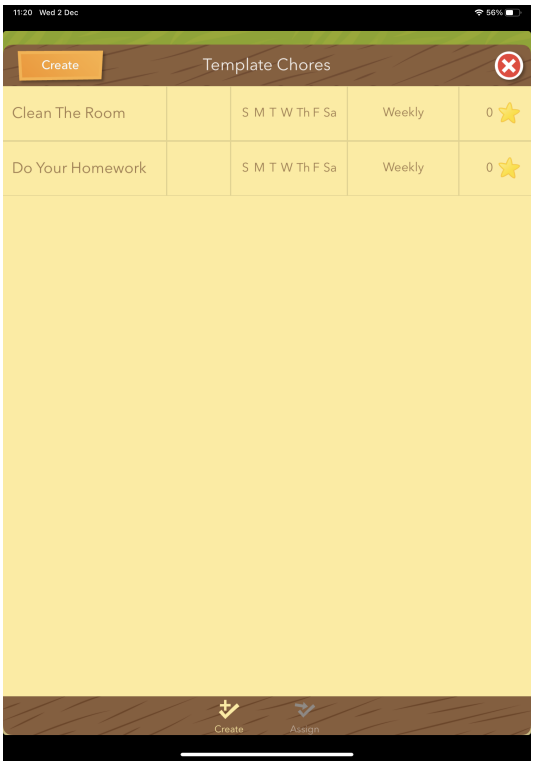


(4) Rewards list

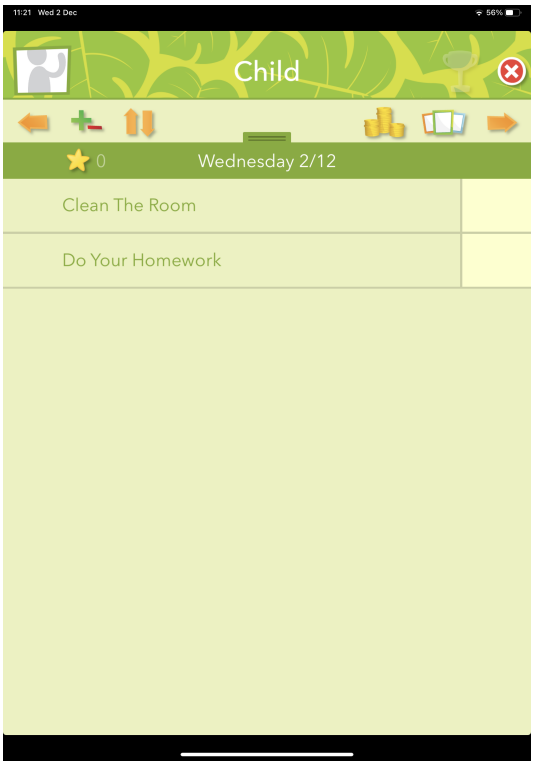
Figure 3.1: *iRewardChart* application screenshots



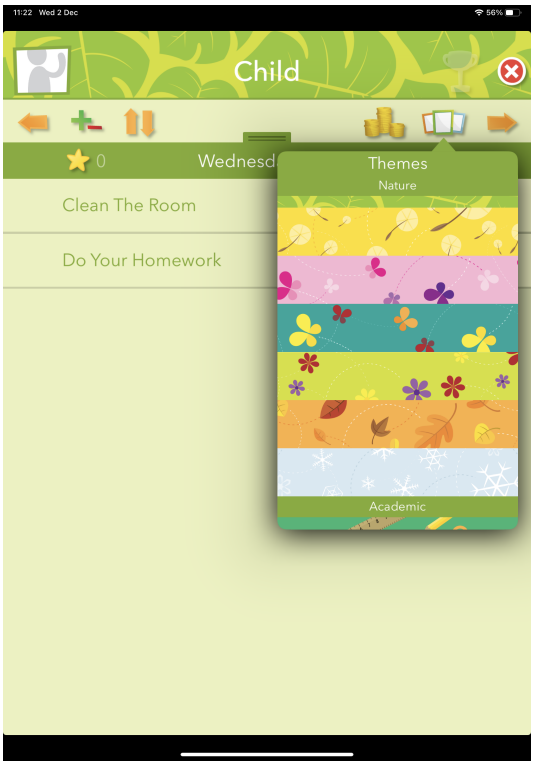
(1) Children list (main screen)



(2) Chores list



(3) Child view



(4) Theme customization

Figure 3.2: *ChorePad* application screenshots

Advantages	Disadvantages
Available on both platforms	Separate free and paid versions
Child-friendly, customisable design	Low Apple App store rating
	Available just on iOS platform
	Quite big application size
	Occasionally confusing design
	Not many options in the free version

Table 3.2: *ChorePad* application advantages and disadvantages

3.2.3 Homey - Chores and Allowance

Homey - Chores and Allowance[6,7] (later referred to as *Homey*) is an application available on both Android and iOS operating systems. The application does not have an age limit and is targeted to everyone. It rates 2.8 on Google Play, has more than 50,000 downloads, and its size is 27MB. It needs Android version 4.1 or higher to run correctly. The last update was released in February 2020.

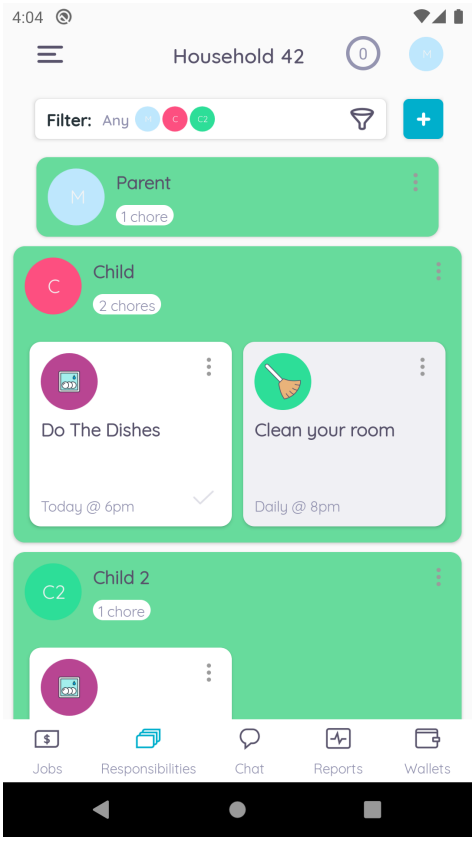
Application is much more complex than the previous two. It has many options, including two different types of tasks (*jobs* and *responsibilities*), pre-defined lists of tasks, in-family chat or even custom reports. Instead of rewards, one can set a weekly allowance that is paid to children once they meet their goals. It also supports switching between parent's and child's perspective. The design is tidy but tends to be over-engineered. *Homey* crashes frequently, which is also pointed out by the community in the ratings.

Example screenshots of the *Homey* application are presented in Figure 3.3. A summary of its merits and demerits was presented in Table 3.3.

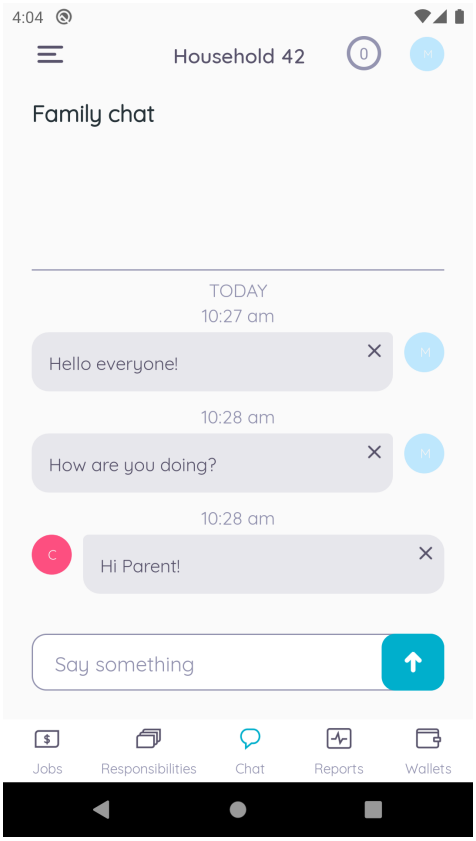
Advantages	Disadvantages
Available on both platforms	Low Google Play rating
Older Android versions support	Frequent crashes
Rich functionality	Occasionally confusing design
	Not many options in the free version

Table 3.3: *Homey* application advantages and disadvantages

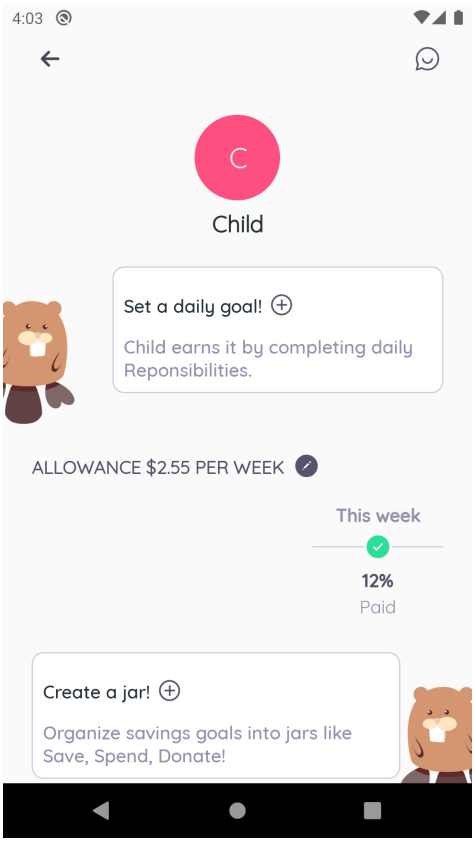
[6]https://play.google.com/store/apps/details?id=com.homey.app (accessed Dec. 2, 2020)
[7]https://apps.apple.com/app/homey-chores-and-allowance/id1033286805 (accessed Dec. 2, 2020)



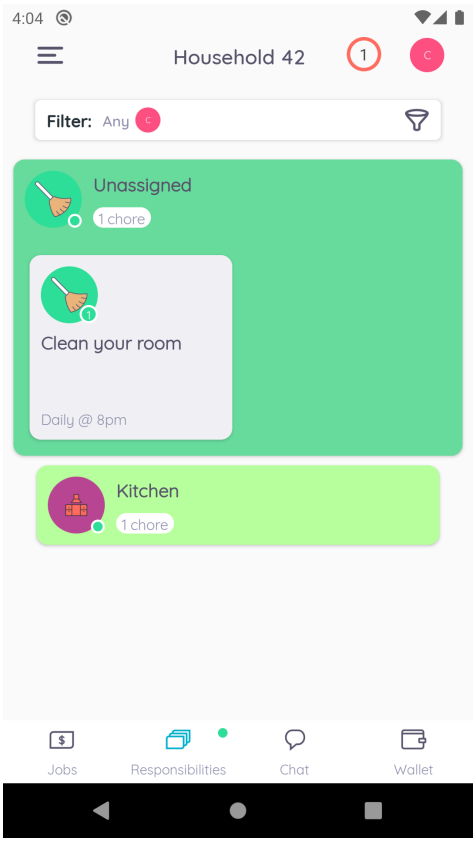
(1) Responsibilities list



(2) In-family chat



(3) Allowance tracker



(4) Child's perspective

Figure 3.3: *Homey* application screenshots

3.2.4 S'moresUp

S'moresUp[8,9] is an application that goes much beyond requirements. It is available on both Android and iOS platforms and is targeted to all age groups. It has more than 100,000 downloads from Google Play and a rating of 4.0. The application requires at least version 4.4 of Android operating system and has a size of 19MB. Its last update was in October 2020.

The application is intricate. It has many screens and options that tend to be overwhelming and illegible. Extra features like in-family social networking might lead to counterproductiveness. *S'moresUp* tends to load data on every screen change, which causes severe delays interrupting the flow and slight issues with saving data. It has both parent's and child's perspective, however, requires a separate email account for every child. Despite free version providing more functionality than predecessors, only paid account allows usage without hindrance.

Example screenshots of the *S'moresUp* application are presented in Figure 3.4. A summary of its strengths and shortcomings was presented in Table 3.4.

Advantages	Disadvantages
Available on both platforms	Frequently confusing design
High Google Play rating	Counterproductive features
Small application size	Long data load
Remarkably rich functionality	Requiring email account for a child
Abundant design	Frequent purchase incentives

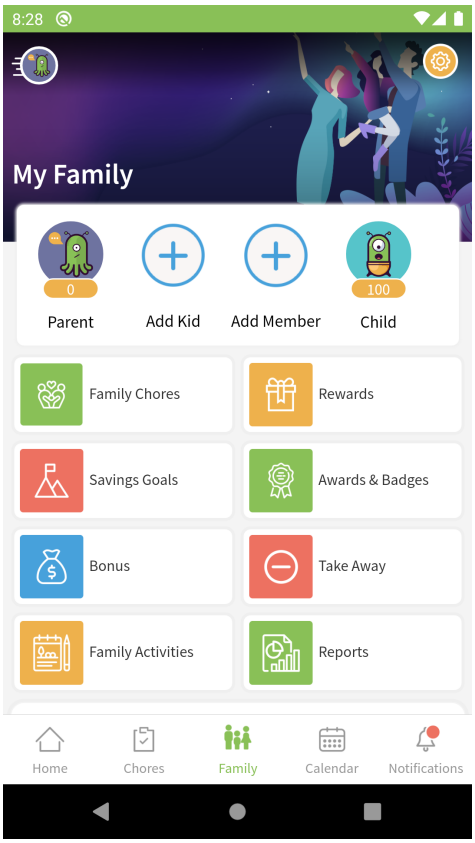
Table 3.4: *S'moresUp* application advantages and disadvantages

3.2.5 OurHome

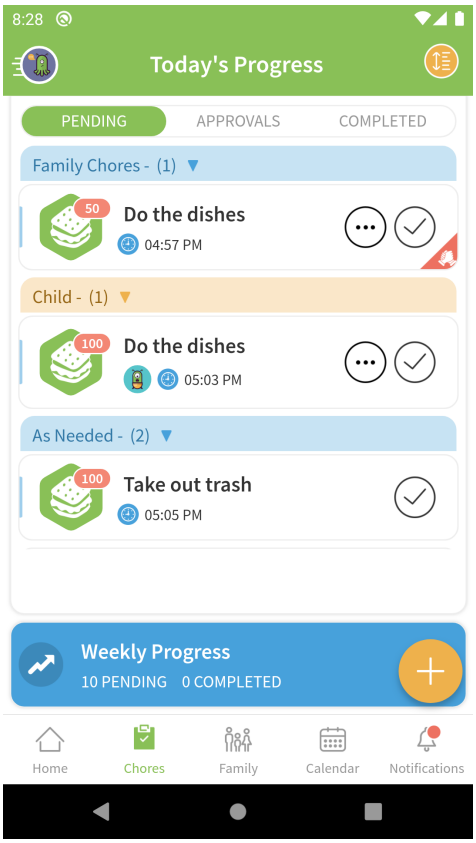
OurHome[10,11] serves the purpose of not only a productivity application but also a family organiser. It is available on both operating systems (Android and iOS) and targeted to everyone. With more than 500,000 downloads it has a rating of 3.6. It requires Android in version 5.0 or higher. Its size is 6.7MB, and the last update was in October 2020.

OurHome's design is very tidy, but one might find it slightly confusing, nonetheless. Instead of separate child's and parent's perspective, it offers an *admin* account, but it does not seem to function as intended. It has features going beyond the core features defined in the Project Assumptions chapter (2.3), such as a calendar or grocery list,

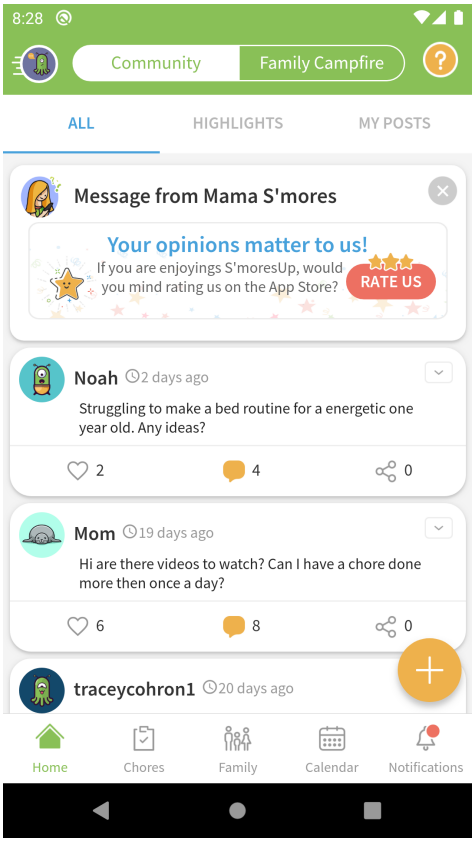
[8]https://play.google.com/store/apps/details?id=com.rotation5.smoresup (accessed Dec. 2, 2020)
[9]https://apps.apple.com/app/smoresup-best-chores-app/id1287367596 (accessed Dec. 2, 2020)
[10]https://play.google.com/store/apps/details?id=com.getfairshare.ourhome (accessed Dec. 2, 2020)
[11]https://apps.apple.com/app/ourhome-chores-and-rewards/id879717020 (accessed Dec. 2, 2020)



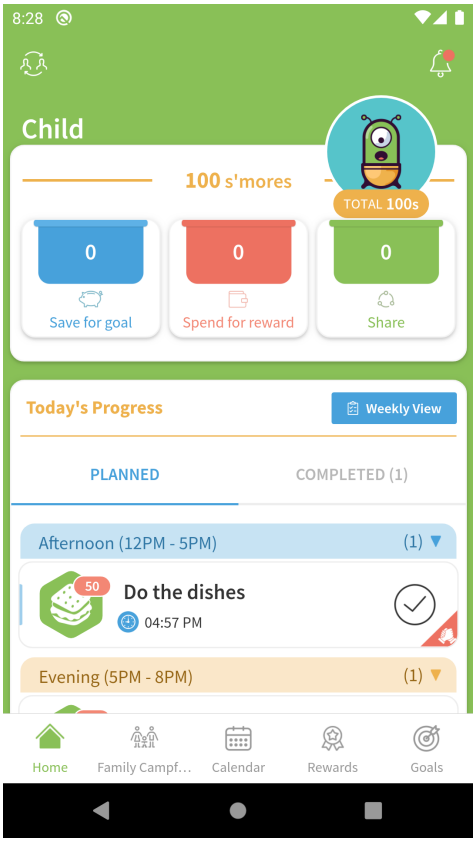
(1) Family dashboard



(2) Chores list



(3) Social networking view



(4) Child's perspective

Figure 3.4: *S'moresUp* application screenshots

as well as many pre-defined tasks. All the features are free. Data load times are occasionally longer than expected.

Example screenshots of the *OurHome* application are presented in Figure 3.5. Rewards list screen was captured during excessive load time. A summary of its benefits and drawbacks was presented in Table 3.5.

Advantages	Disadvantages
Available on both platforms	Occasionally confusing design
Small application size	Non-functional child's perspective
Rich, free functionality	Infrequent long data loads
Tidy design	

Table 3.5: *OurHome* application advantages and disadvantages

3.3 Conclusions

It is crucial to draw conclusions from the competition analysis. There are several categories of opportunities and obstacles that are especially important:

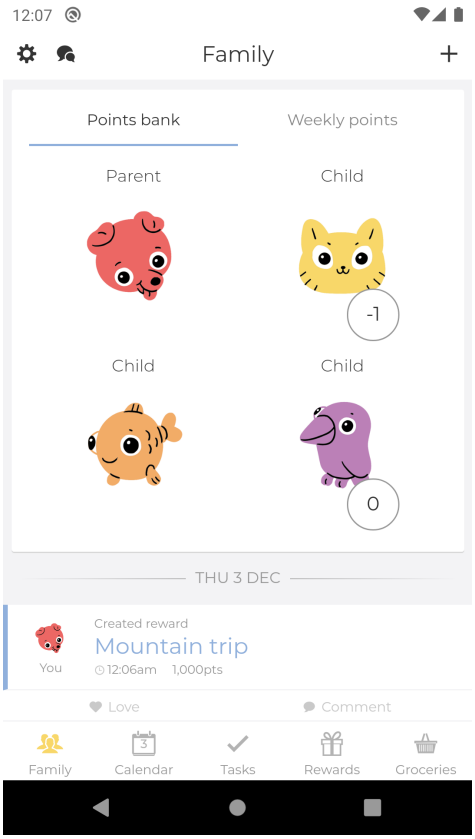
- Design and user experience
- Functionality
- Performance

3.3.1 Design and user experience

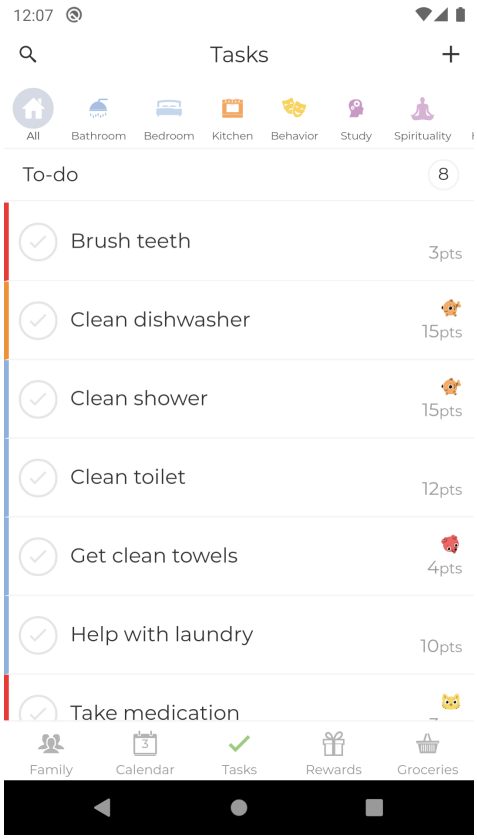
One of the most distinctive issues all the applications face is design and user experience. The layout is either not engaging enough or perplexing. It ought to be tidy, child-friendly and simple in order to minimise the risk of user loss. Several analysed applications rely on *Material Design* [26] design system, which helps to provide clear interface but, if not used correctly or abused, might lead to confusion or a raw, unappealing design.

3.3.2 Functionality

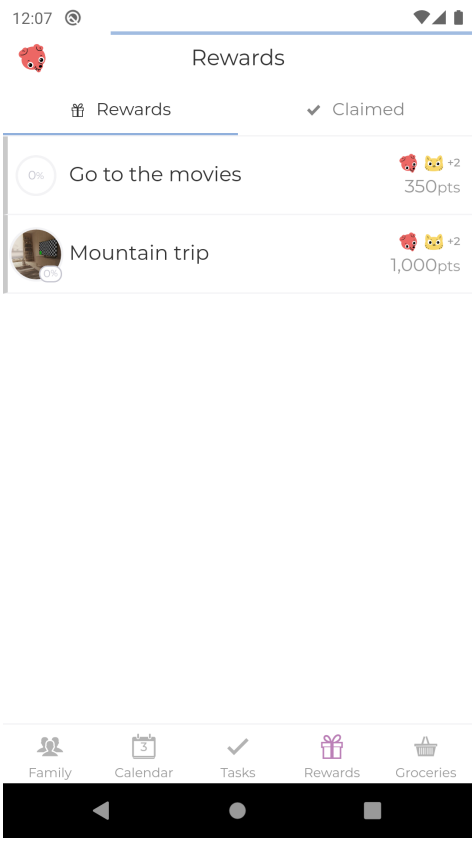
Functionality, even if extensive, should always serve the primary purpose of the application. Derogations like social networking in *S'moresUp*, while attractive business-wise, could not be implemented in the project. Thoroughly polished features are of at least the same importance. *Quality over quantity* should apply to the final product's functionality.



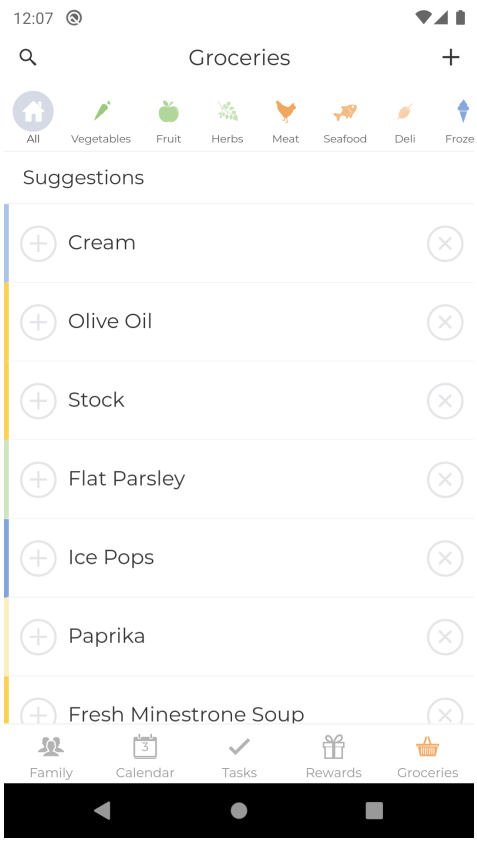
(1) Family view



(2) Tasks list



(3) Rewards list



(4) Groceries list

Figure 3.5: *OurHome* application screenshots

3.3.3 Performance

Even intermittent application crashes or lags might cause a tremendous user loss. It is a case of *Homey - Chores and Allowance* and *OurHome* and is reflected in user ratings and reviews. The implemented application should be stable and working smoothly, without long waiting times.

Chapter 4

Project requirements

4.1 Subject of the Project

The subject matter of the project is to develop a working prototype of a mobile application with gamification mechanisms and its verification.

4.2 Users

The application will have three types of users - a *New User*, a *Parent* and a *Child*.

4.2.1 New User

A *New User* is a person, who does not have an account in the system yet. They should be able to create an account.

4.2.2 Parent

A *Parent* user is a kind of administrator user. They have access to their account, where they can add children, set, modify, and delete tasks and rewards for them, as well as manage the application. It is also them who has access to children's devices and sets up their profiles. They have access to all settings, where they can modify the application's behaviour.

4.2.3 Child

A *Child* user should be restricted from any administrative action within the application. The implemented mechanisms should not encourage spending more time using the phone, but to be more productive. They should have access to their profile information, tasks, and rewards.

4.3 Functional requirements

The IEEE Standard 830 [33] is one of the most renowned documents specifying a traditional approach to defining requirements, which should start with "The system shall...". Bob Lightsey, in *Systems Engineering Fundamentals* from 2001, states that

functional requirements "Define what the system must accomplish or must be able to do." [39]. This kind of approach is referred to as *system-centred*.

Capturing functional requirements as *user stories* is *user-centred*. "User stories are the right size for planning, comprehensible by everyone, work for iterative development and support opportunistic design", which is "just right for very early planning" [11].

User stories, for each user separately, will be used to define functional requirements for the project. Numeration of every requirement will be prefixed with *FR* for a future reference.

4.3.1 New User user stories

FR 1. As a New User, I want to be able to choose parent device belonging so that after registration, I am not confused about my role within the application.

FR 2. As a New User, I want to be able to register an account so that I can access the application's functionality.

FR 3. As a New User, if there is an error during the registration process, I want to get the error message displayed so that I know the reason for the failure.

4.3.2 Parent user stories

4. As a Parent, I want to be able to log in to my account so that I can have access to the saved data and operate on it.

5. As a Parent, if there is an error during the login process, I want to get the error message displayed so that I know the reason for the failure.

6. As a Parent, I want to be able to log out of my account on my device so that I do not have access to the application's functionality.

7. As a Parent, I want to be able to log out of my account on my child's device so that they do not have access to the application's functionality.

8. As a Parent, I want to be able to change device belonging so that a child can access their account on my device.

9. As a Parent, I want to be able to change my child's device belonging so that I can access my account on their device.

10. As a Parent, I want to be able to display a list of my children so that I can choose whose profile I want to manage.

11. As a Parent, I want to be able to display a chosen child's profile information so that I can recollect their number of points, tasks and rewards.

12. As a Parent, I want to be able to add a child's profile so that they can access it.

13. As a Parent, I want to be able to edit my child's profile so that I can reflect any changes in reality or correct my mistakes.

14. As a Parent, I want to be able to delete my child's profile so that I do not see it on the children's list.

15. As a Parent, I want to be able to add new tasks to a child's profile so that I can set their goals.

16. As a Parent, I want to be able to add new rewards to a child's profile so that I can reward them for achieving the goals.

17. As a Parent, I want to be able to define tasks' title and description so that I can specify my expectations.

18. As a Parent, I want to be able to define rewards' title and description so that I can portray it better.

19. As a Parent, when a child performs their task, I want to be able to mark the task as done so that I can distinguish them from tasks to be done.

20. As a Parent, when a child receives their reward, I want to be able to mark the reward as claimed so that I can distinguish them from the still available rewards.

21. As a Parent, I want to be able to display rewards that my child has too few points to claim so that I know which rewards are not yet available for them.

22. As a Parent, I want to be able to define tasks' and or rewards' points so that I can motivate my child further.

23. As a Parent, when I change the status of a child's task or reward, I want their points to change so that I know what their current balance is.

24. As a Parent, I want to be able to display a list of the chosen child's tasks or rewards so that I can have a clear representation of their state.

25. As a Parent, I want to be able to edit tasks or rewards so that I can correct my mistakes or adapt its content to new requirements.

26. As a Parent, I want to be able to delete tasks or rewards so that the list of tasks does not contain unnecessary elements.

4.3.3 Child user stories

27. As a Child, I want to be able to display my profile information so that I can recollect my number of points, tasks and rewards.

28. As a Child, I want to be able to display a list of my tasks or rewards so that I can have a clear representation of their state.

29. As a Child, I want to be able to display done and to-do tasks separately so that I can distinguish between them.

30. As a Child, I want to be able to display available and claimed rewards separately so that I can distinguish between them.

31. As a Child, I want to be able to display rewards that I have too few points to claim so that I know which rewards are not yet available for me.

4.4 Non-functional requirements

While functional requirements concern users and the functionality available to them, non-functional requirements are system related and cannot be expressed as user stories [11], therefore, they will be defined in a standard way. Numeration of every requirement will be prefixed with *NFR* for a future reference.

4.4.1 Requirements definition

NFR 1. Application will be written in a technology that supports compilation for both Android and iOS operating systems.

NFR 2. Application will run on devices with at least 4.1 version of Android operating system.

NFR 3. Application interface will be responsive [34] on a screen with a diagonal length of up to 6.4 inches.

NFR 4. User authentication service will handle at least 10 requests per second.

NFR 5. Database service will handle at least 10000 read requests and at least 5000 write and delete requests per day.

4.4.2 Requirements reasoning

As per project platform assumptions (Section 2.4), the application should be targeted to devices with Android operating system, but should be compilable to iOS in the future. According to the information displayed in the Android Studio's Create New Project wizard[12], version 4.1 covers about 99.8% devices. It is also the lowest version that the project can be created for at the time of writing.

The application, at least in the first iterations, will be designed for smartphones, at the same time not many smartphones have screen size larger than 6.4 inches. Application responsiveness requires additional effort during development and testing, therefore considering tablets could unnecessarily increase the delivery time.

It is not required that authentication and database services handle high traffic from the early stages. Defined requirements guarantee that the architecture withstands temporary traffic spikes as well as initial stages of user acquisition.

[12]'Distribution dashboard', Android Developers. https://developer.android.com/about/dashboards (accessed Dec. 05, 2020).

Chapter 5

Project

5.1 Introduction

"A goal without a plan is just a wish." - this sentence is often ascribed to Antoine de Saint Exupéry, and while some question this attribution, no one disputes the essence of the quote. The requirements defined in the previous chapter allow to move on to the project phase, which allows for an early product discovery.

In Section 5.2, the use case modelling will be performed. It will help to analyse the system from the user's point of view. The interface, and visual identity of the application will be designed in Section 5.3. Section 5.4 will introduce and model a database needed for the project.

5.2 Use Case modelling

"The basic idea behind use-case modelling is quite simple: To get to the heart of what a system must do, you must first focus on who (or what) will use it, or be used by it. After you do this, look at what the system must do for those users in order to do something useful." [3] - this quote impeccably describes the need for use case modelling. A thorough modelling process will lay foundations for further planning and the implementation phase.

5.2.1 Use Case diagram

A use case diagram is presented in Figure 5.1. It has 5 actors - three of them are users (primary actors) of the applications and the remaining two (secondary actors) are database and authentication services. There is a generalization relation between the Child and the Parent - it indicates that the Parent, beyond their own use cases, has access to all of the behaviours of the Child (*Note:* In the use case notation, the actor, which inherits the structures, behaviours and relationships is called a *child* and its generalization is called a *parent* - it is not to be confused with the actor names in the diagram; in this case the Parent actor is a descendant of the Child actor).

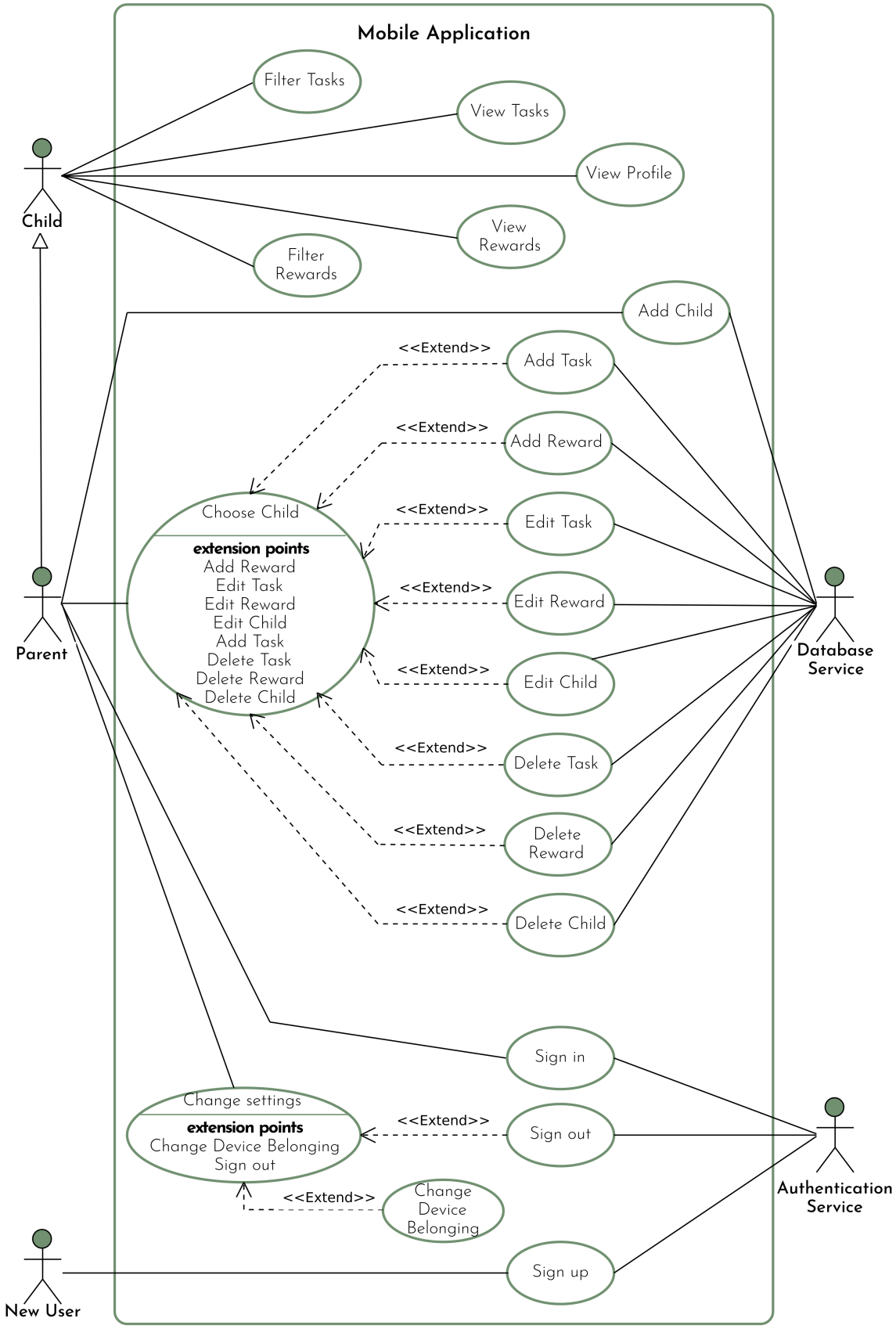


Figure 5.1: Use case diagram of the application system

5.2.2 Use Case descriptions

The use case description provides necessary information that specify the use case flow. Descriptions' structure is inspired, yet slightly modified, by the template proposed by the Alistar Cockburn in "Writing Effective Use Cases" [10]. Due to the visual nature of the application, the elements of the user interface are mentioned (discouraged by the author), yet reduced to minimum. The descriptions are not exhaustive, however, give enough information to continue designing the interface. Below, an explanation of the descriptions' elements is provided.

- *Use Case* - use case unique code.
- *Name* - use case name referencing the diagram.
- *Primary actor* - a primary actor name.
- *Trigger* - an intention of the primary actor, which initiates the use case.
- *Preconditions* - a list of expected states of the system, required for the use case to start.
- *Postconditions* - a list of expected results of the system after if the use case is finished successfully.
- *Basic flow* - a *base success scenario*; a list of steps leading to a successful finish of the use case.
- *Alternative flows* - alternative steps that might be taken to reach a successful ending.
- *Exceptions* - any list of issues or steps that do not lead to achieving the use case's goal.
- *Extensions* - a list use cases that extend the described use case.

The use case descriptions are presented in Tables 5.1 - 5.20.

Use Case	UC1
Name	View Tasks
Primary actor	Child
Trigger	A Child wants to see their tasks.
Preconditions	1. A child's profile has been chosen.
Postconditions	1. The Child's tasks are displayed.
Basic flow	1. The Child navigates to a tasks screen. 2. If there are tasks assigned to the child, they are displayed.
Alternative flows	2.a. If there are no tasks assigned to the child, an information about it is displayed instead.
Exceptions	-
Extensions	-

Table 5.1: *View Tasks* use case description

Use Case	UC2
Name	View Profile
Primary actor	Child
Trigger	A Child wants to see their profile information.
Preconditions	1. A child's profile has been chosen.
Postconditions	1. The Child's profile information is displayed.
Basic flow	1. The Child navigates to a profile screen. 2. The profile information is displayed.
Alternative flows	-
Exceptions	-
Extensions	-

Table 5.2: *View Profile* use case description

Use Case	UC3
Name	View Rewards
Primary actor	Child
Trigger	A Child wants to see their rewards.
Preconditions	1. A child's profile has been chosen.
Postconditions	1. The Child's rewards are displayed.
Basic flow	1. The Child enters a rewards screen. 2. If there are rewards assigned to the child, they are displayed.
Alternative flows	2.a. If there are no rewards assigned to the child, an information about it is displayed instead.
Exceptions	-
Extensions	-

Table 5.3: *View Rewards* use case description

Use Case	UC4
Name	Filter Tasks
Primary actor	Child
Trigger	A Child wants to filter the displayed tasks.
Preconditions	1. A child's profile has been chosen. 2. A tasks screen has been entered.
Postconditions	1. The displayed Child's tasks are filtered according to the filter choice.
Basic flow	1. The Child chooses a filter from a list of available filters. 2. The displayed tasks are filtered accordingly.
Alternative flows	2.a. If there are no tasks with the assigned filter, an information about it is displayed instead.
Exceptions	-
Extensions	-

Table 5.4: *Filter Tasks* use case description

Use Case	UC5
Name	Filter Rewards
Primary actor	Child
Trigger	A Child wants to filter the displayed rewards.
Preconditions	1. A child's profile has been chosen. 2. A rewards screen has been entered.
Postconditions	1. The displayed Child's rewards are filtered according to the filter choice.
Basic flow	1. The Child chooses a filter from a list of available filters. 2. The displayed rewards are filtered accordingly.
Alternative flows	2.a. If there are no rewards with the assigned filter, an information about it is displayed instead.
Exceptions	-
Extensions	-

Table 5.5: *Filter Rewards* use case description

Use Case	UC6
Name	Add Child
Primary actor	Parent
Trigger	A Parent wants to add a new child's profile.
Preconditions	1. A parent's perspective has been enabled.
Postconditions	1. A new child's profile is added and visible on the children's profiles list.
Basic flow	1. The Parent clicks an "add child" button. 2. The Mobile Application displays an "add child" form. 3. The Parent fills in the form with the new child's profile data and confirms it. 4. The Mobile Application sends the data to the Database Service and updates its children list. 5. The new child's profile is displayed on the children's profiles list.
Alternative flows	-
Exceptions	3.a. If the input data is not compliant with the validation rules, the system displays an error message.
Extensions	-

Table 5.6: *Add Child* use case description

Use Case	UC7
Name	Add Task
Primary actor	Parent
Trigger	A Parent wants to add a task for their child.
Preconditions	1. A parent's perspective has been enabled. 2. A child's profile has been chosen.
Postconditions	1. A new child's task is added and visible on the child's task screen.
Basic flow	1. The Parent clicks an "add task" button. 2. The Mobile Application displays an "add task" form. 3. The Parent fills in the form with the new task's data and confirms it. 4. The Mobile Application sends the data to the Database Service and updates its tasks list. 5. The new task is displayed on the tasks list.
Alternative flows	-
Exceptions	3.a. If the input data is not compliant with the validation rules, the system displays an error message.
Extensions	-

Table 5.7: *Add Task* use case description

Use Case	UC8
Name	Add Reward
Primary actor	Parent
Trigger	A Parent wants to add a reward for their child.
Preconditions	1. A parent's perspective is enabled. 2. A child's profile is chosen.
Postconditions	1. A new child's reward is added and visible on the child's task screen.
Basic flow	1. The Parent clicks an "add reward" button. 2. The Mobile Application displays an "add reward" form. 3. The Parent fills in the form with the new reward's data and confirms it. 4. The Mobile Application sends the data to the Database Service and updates its rewards list. 5. The new reward is displayed on the rewards list.
Alternative flows	-
Exceptions	3.a. If the input data is not compliant with the validation rules, the system displays an error message.
Extensions	-

Table 5.8: *Add Reward* use case description

Use Case	UC9
Name	Edit Task
Primary actor	Parent
Trigger	A Parent wants to edit their child's task.
Preconditions	1. A parent's perspective has been enabled. 2. A child's profile has been chosen.
Postconditions	1. An edited task has its details or state changed.
Basic flow	1. The Parent long-presses a chosen task from the tasks list. 2. The Mobile Application displays an "edit task" form with the task data pre-filled. 3. The Parent amends the data and confirms the changes. 4. The Mobile Application sends the data to the Database Service and updates its tasks list. 5. The task with edited data is displayed on the tasks list.
Alternative flows	1.a. The Parent taps a chosen task from the tasks list. 2.a. The Parent confirms changing the task state.
Exceptions	3.b. If the input data is not compliant with the validation rules, the system displays an error message.
Extensions	-

Table 5.9: *Edit Task* use case description

Use Case	UC10
Name	Edit Reward
Primary actor	Parent
Trigger	A Parent wants to edit their child's reward.
Preconditions	1. A parent's perspective has been enabled. 2. A child's profile has been chosen.
Postconditions	1. An edited reward has its details or state changed.
Basic flow	1. The Parent long-presses a chosen reward from the rewards list. 2. The Mobile Application displays an "edit reward" form with the reward data pre-filled. 3. The Parent amends the data and confirms the changes. 4. The Mobile Application sends the data to the Database Service and updates its rewards list. 5. The reward with edited data is displayed on the rewards list.
Alternative flows	1.a. The Parent taps a chosen reward from the rewards list. 2.a. The Parent confirms changing the reward state.
Exceptions	3.b. If the input data is not compliant with the validation rules, the system displays an error message.
Extensions	-

Table 5.10: *Edit Reward* use case description

Use Case	UC11
Name	Edit Child
Primary actor	Parent
Trigger	A Parent wants to edit their child's profile.
Preconditions	1. A parent's perspective has been enabled. 2. A child's profile has been chosen.
Postconditions	1. An edited child's profile has its details changed.
Basic flow	1. The Parent navigates to the child's profile screen and clicks an "edit child" button. 2. The Mobile Application displays an "edit child" form with the child's profile data pre-filled. 3. The Parent amends the data and confirms the changes. 4. The Mobile Application sends the data to the Database Service and updates its child's profile data. 5. The child's profile with edited data is displayed on the children's profiles list.
Alternative flows	-
Exceptions	3.a. If the input data is not compliant with the validation rules, the system displays an error message.
Extensions	-

Table 5.11: *Edit Child* use case description

Use Case	UC12
Name	Delete Task
Primary actor	Parent
Trigger	A Parent wants to remove their child's task.
Preconditions	1. A parent's perspective has been enabled. 2. A child's task edit form has been entered.
Postconditions	1. A deleted task is available neither for the child, nor the parent.
Basic flow	1. The Parent clicks a "delete task" button and confirms the action. 2. The Mobile Application sends the delete request to the Database Service and updates its child's tasks list. 3. The deleted task is not displayed on the tasks list.
Alternative flows	-
Exceptions	-
Extensions	-

Table 5.12: *Delete Task* use case description

Use Case	UC13
Name	Delete Reward
Primary actor	Parent
Trigger	A Parent wants to remove their child's reward.
Preconditions	1. A parent's perspective has been enabled. 2. A child's reward edit form has been entered.
Postconditions	1. A deleted reward is available neither for the child, nor the parent.
Basic flow	1. The Parent clicks a "delete task" button and confirms the action. 2. The Mobile Application sends the delete request to the Database Service and updates its child's tasks list. 3. The deleted reward is not displayed on the rewards list.
Alternative flows	-
Exceptions	-
Extensions	-

Table 5.13: *Delete Reward* use case description

Use Case	UC14
Name	Delete Child
Primary actor	Parent
Trigger	A Parent wants to remove their child's profile.
Preconditions	1. A parent's perspective has been enabled. 2. A child's profile edit form has been entered.
Postconditions	1. A deleted child's profile is available neither for the child, nor the parent.
Basic flow	1. The Parent clicks a "delete child" button and confirms the action. 2. The Mobile Application sends the delete request to the Database Service for all the child's tasks, rewards along with the child's profile and updates its children's profiles list. 3. The deleted child's profile is not displayed on the children's profiles list.
Alternative flows	-
Exceptions	-
Extensions	-

Table 5.14: *Delete Child* use case description

Use Case	UC15
Name	Choose Child
Primary actor	Parent
Trigger	A Parent wants to choose the child's profile to start managing it.
Preconditions	1. A parent's perspective has been enabled.
Postconditions	1. A child's profile is selected and can be managed.
Basic flow	1. The Parent clicks a "child profile" element on the children's profiles list. 2. The Mobile Application navigates to the child's profile screen.
Alternative flows	-
Exceptions	-
Extensions	1. *Add Task* (Table 5.7) 2. *Add Reward* (Table 5.8) 3. *Edit Task* (Table 5.9) 4. *Edit Reward* (Table 5.10) 5. *Edit Child* (Table 5.11) 6. *Delete Task* (Table 5.12) 7. *Delete Reward* (Table 5.13) 8. *Delete Child* (Table 5.14)

Table 5.15: *Choose Child* use case description

Use Case	UC16
Name	Sign in
Primary actor	Parent
Trigger	A Parent wants to access their account.
Preconditions	1. A Parent is not logged in. 2. A device belonging choice screen has been entered.
Postconditions	1. A parent is logged in and can access their account.
Basic flow	1. The Parent chooses the *parent* device belonging. 2. The Mobile Application navigates to the register screen. 3. The Parent clicks a *Sign in instead* button. 4. The Mobile Application navigates to the login screen. 5. The Parent fills in the login data and confirms it. 6. The Mobile Application sends the data to the Authentication Service. 7. The Authentication Service confirms the data is correct. 8. The Mobile Application saves user's authentication token and displays the parent's perspective.
Alternative flows	1.a. The Parent chooses the *child* device belonging. 2.a. The flow continues from point no. 4 of the base flow.
Exceptions	3.b. The Authentication Service denies the data is correct. 4.b. The Mobile Application displays an error message.
Extensions	-

Table 5.16: *Sign in* use case description

Use Case	UC17
Name	Sign out
Primary actor	Parent
Trigger	A Parent wants to stop accessing their account.
Preconditions	1. A Parent is logged in. 2. A settings screen has been entered.
Postconditions	1. A parent is logged out and cannot access their account.
Basic flow	1. The Parent clicks *Sign out* button. 2. The Mobile Application sends the *sign out* request to the Authentication Service 3. The Mobile Application navigates to the
Alternative flows	-
Exceptions	-
Extensions	-

Table 5.17: *Sign out* use case description

Use Case	UC18
Name	Change Device Belonging
Primary actor	Parent
Trigger	A Parent wants to toggle between the user's perspectives.
Preconditions	1. A Parent is logged in. 2. A settings screen has been entered.
Postconditions	1. A device belonging is changed.
Basic flow	1. If the *parent's* device belonging is chosen, the Parent selects the *child's* device belonging. 2. The Mobile Application displays an additional child's profile select field. 3. The Parent chooses a specific child's profile and confirms their choice. 4. The Mobile Application saves the selected device belonging.
Alternative flows	1.a. If the *child's* device belonging is chosen, the Parent selects the *parent's* device belonging. 2.a. The Mobile Application hides the additional child's profile select field. 3.a. The Parent confirms their choice. --- 1.b. If the *child's* device belonging is chosen, the Parent selects another child in the child's profile select field. 2.b. - 3.b. The Parent confirms their choice.
Exceptions	-
Extensions	-

Table 5.18: *Change Device Belonging* use case description

Use Case	UC19
Name	Change settings
Primary actor	Parent
Trigger	A Parent wants to display available settings.
Preconditions	1. A Parent is logged in. 2. A children's profiles list screen (in the parent's perspective) or a child's profile screen (in the child's perspective) is entered.
Postconditions	1. A settings screen is displayed.
Basic flow	1. The Parent clicks a settings button. 2. The Parent enters their password to authorize settings edition. 3. The Mobile Application displays screen setting.
Alternative flows	-
Exceptions	-
Extensions	1. *Sign out* (Table 5.17) 2. *Change Device Belonging* (Table 5.18)

Table 5.19: *Change settings* use case description

Use Case	UC20
Name	Sign up
Primary actor	New User
Trigger	A New User wants to create an account.
Preconditions	1. The Mobile Application does not have device belonging or login data saved.
Postconditions	1. A new account is registered, and the New User becomes a Parent.
Basic flow	1. The New User chooses the parent's device belonging. 2. The Mobile Application saves the device belonging and navigates to the register screen. 3. The New User enters the new account data and confirms it with a button. 4. The Mobile Application sends the data to the Authentication Service. 5. The Authentication Service confirms the data is correct. 6. The Mobile Application saves user's authentication token and displays the parent's perspective.
Alternative flows	-
Exceptions	5.a. The Authentication Service denies the data is correct. 6.a. The Mobile Application displays an error message.
Extensions	-

Table 5.20: *Sign up* use case description

5.3 Interface design

Interface, as one of the most important part of the application, needs to be well thought-through. It not only needs to be appealing to the eye, but simple to use as well. It needs to evoke positive emotions both in parents and children. Designing the interface is also the stage of a project when a name and visual identity should be planned.

5.3.1 Name

The name of an application should relate to the application's purpose, while being simple and memorable at the same time. It is expected to be unique in order to perform well among competition in the Google Play.

The application is going to help parents raise their children. To "raise up" means to elevate something to a position or status of higher regard or value[13]. It correlates with the application's goal to help parents raise (and raise up) their children. Additionally, the word "up" has the same pronunciation as "app", which creates an elegant word play that can be used as an application name - *Raise App*.

5.3.2 Colours

Another important aspect of the visual identity are colours. According to many studies, bright colours evoke positive emotions, especially in children [5]. Two of the best performing colours are green and yellow - not only will they be appealing to children, but they also harmonize very well (they follow an "Analogous Scheme" rule [57].

The chosen colours are *#709070* - green (Figure 5.2) and *#EEDD60* - yellow (Figure 5.3). The green resembles the colour of the flora and the yellow - the colour of the sun.

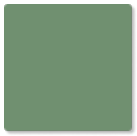


Figure 5.2: The green (*#709070*) colour used in the project

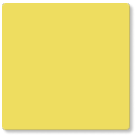


Figure 5.3: The yellow (*#EEDD60*) colour used in the project

5.3.3 Typography

Fonts are often one of the first things that are noticed by a user. It is important that the used fonts are not only legible and compatible with mobile devices, but also appealing to the eye.

[13] https://idioms.thefreedictionary.com/raise+up (accessed Dec. 10, 2020)

Main font

The application is targeted to children; therefore, the main font should maintain a balance between legibility on mobile devices (*sans-serif* fonts) and decorativeness. A font that meets both criteria best is *Patrick Hand*. It is described as "Patrick Hand is a font based on the designer's own handwriting. It is developed to bring an impressive and useful handwriting effect to your texts."[14] and licensed under the *Open Font License*[15], which means it can be freely used in the project.

Secondary font

A good font combination (also referred to as "font pairing") can provide a better perception of the content in the application. While the main font will be used for headers and highlighting important information, a secondary font will allow for displaying supplementary content. The font should be contrasting, but complementary. One of the fonts widely recommended to pair with *Patrick Hand* is *Josefin* font family. *Josefin Sans* is a *sans-serif* font that belongs to the family. It offers multiple weights and is licensed under the Open Font License. Its elegant style will contrast the decorativeness of the main font, whilst the letterform will complement it. This ensures a sufficient balance between them.

The chosen fonts have been presented in Figure 5.4 (Patrick Hand) and Figure 5.5 (Josefin Slab).

Raise app

Figure 5.4: Patrick Hand font example

Tasks & Rewards

Figure 5.5: Josefin Sans font example

5.3.4 Logo and Theme

The perception of the application is also shaped by its symoblics. It needs to be anchored in its design. The applications name and colours suggest a close connection to the nature. The colours symbolize strong and stable trees (green), and a sun and its energy (yellow). It is also reflected in the application name. Based on this information, a logomark and a logotype of the application was designed and is presented in Figure 5.6.

5.3.5 Design

A prototype of the application is created in a *Figma*[16] prototyping tool. Based on the elements and guidelines from the *Material Design* [26], a bespoke style guide has been defined. The prototype can be accessed under this link[17].

[14] https://fonts.google.com/specimen/Patrick+Hand (accessed Dec. 10, 2020)
[15] https://scripts.sil.org/cms/scripts/page.php?site_id=nrsi&id=OFL (accessed Dec. 10, 2020)
[16] https://www.figma.com (accessed Dec. 10, 2020)
[17] https://www.figma.com/proto/fgJWEECDgvKUO7FPYkv4QA/RaiseApp?node-id=788%3A0scaling=scale-down (accessed Dec. 13, 2020)

Figure 5.6: Application's logomark and logotype

User interface flow diagram

Due to complexity of the user interface logic, a user interface flow diagram [1] has been created in order to clarify the screen flow within the application. It is presented in Figure 5.7. In the diagram, boxes illustrate screens and arrows mark a path between the screens (arrowheads imply navigation direction). A black, densely dashed box indicates a screen that can be accessed from any user's perspective. A yellow, loosely dashed box indicates a screen that can only be accessed from the child's perspective, but not from the parent's perspective (there are no screens of this kind in the application). A green, solid-line box a screen that can only be accessed form the parent's perspective, but not from the child's perspective. Analogically, a black, densely dashed imply the flow from both users' perspectives, a yellow, loosely dashed - only from the child's perspective and a green, solid, line - only from the parent's perspective.

Application screens

In Figure 5.8, most of the designed application's screens are presented. Some exceptions are caused by both prototyping tool and this work's text form limitations (i.e. loading animations). Some of the screens from the child's perspective were omitted, due to their analogous look and behaviour. The screens descriptions are provided below.

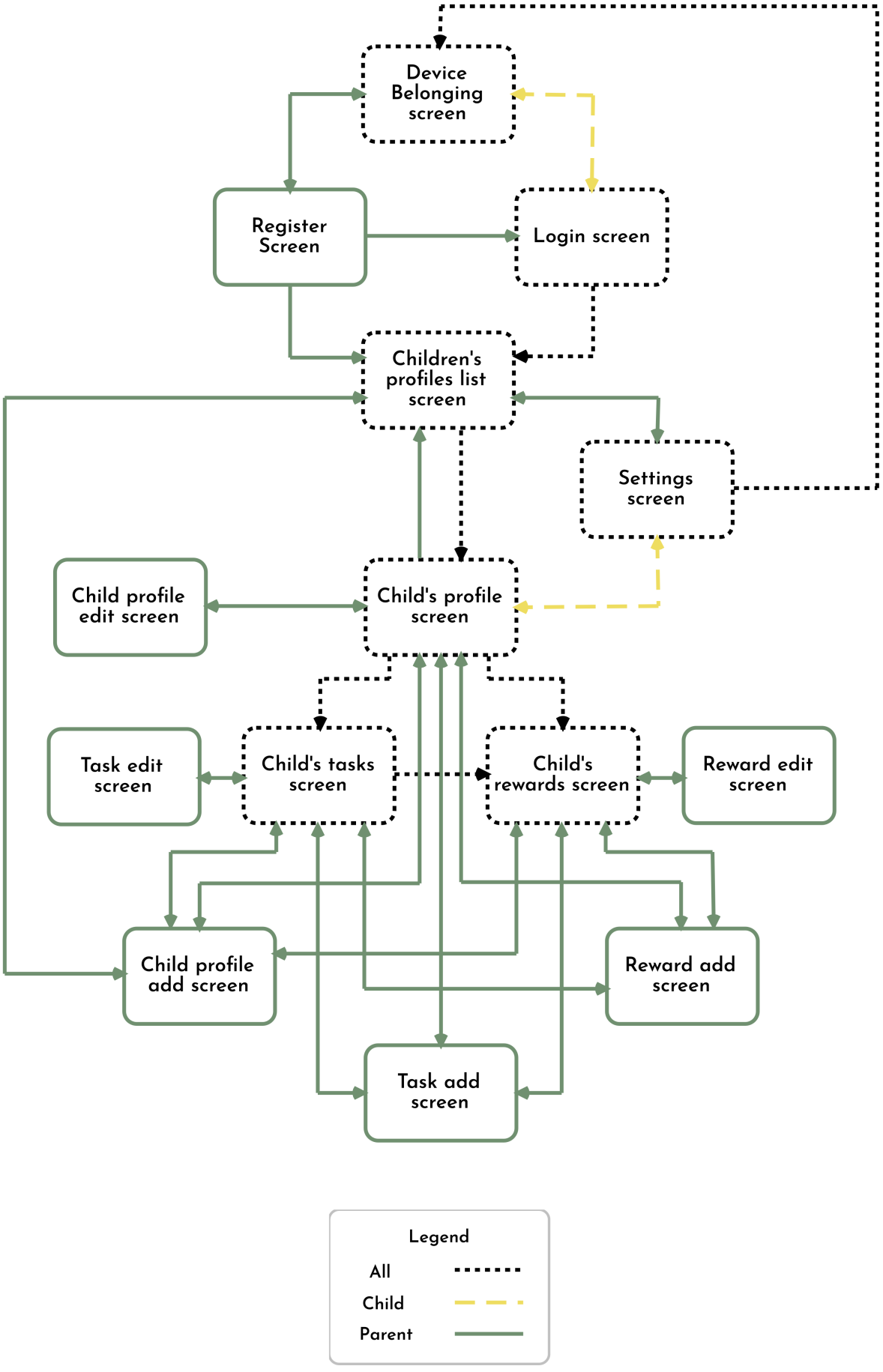
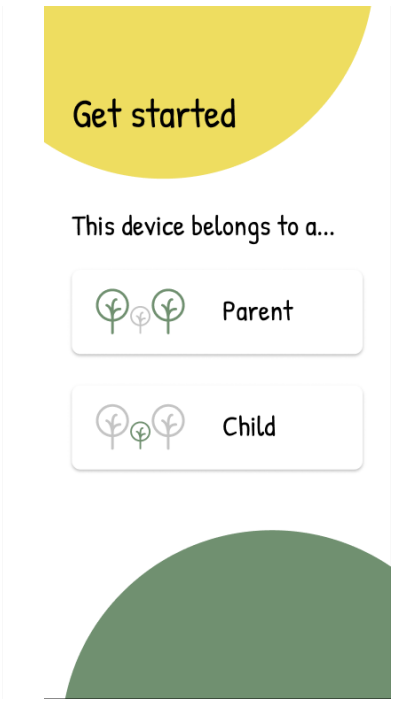
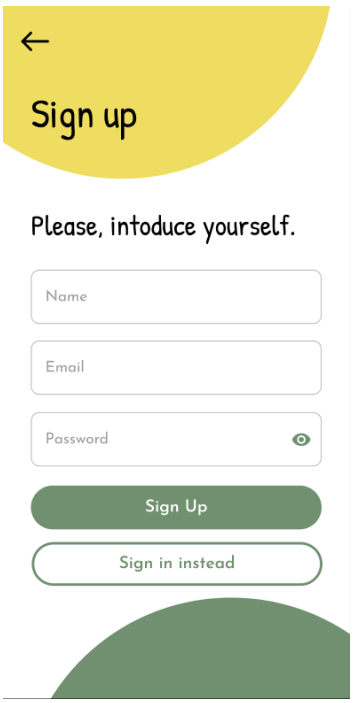


Figure 5.7: User interface flow diagram

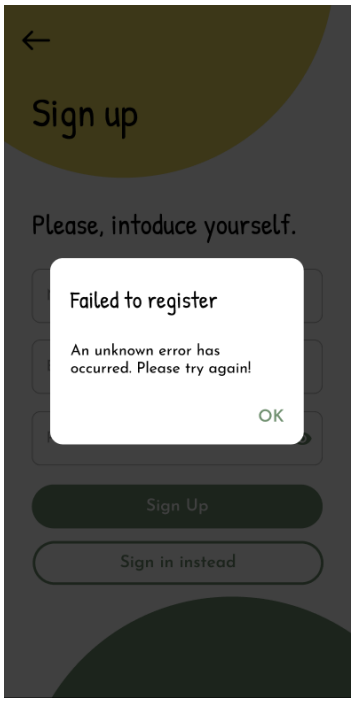


(1) Splash screen

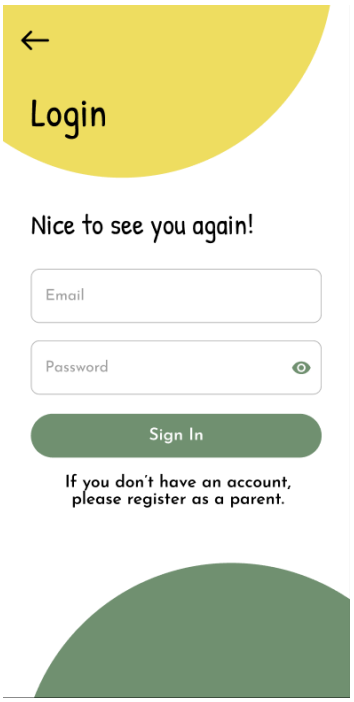
(2) Belonging screen



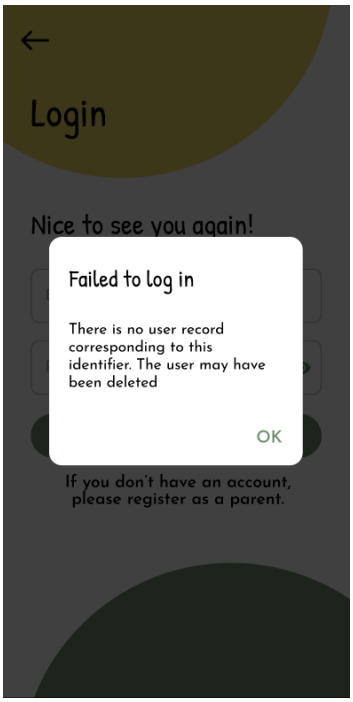
(3) Register screen



(4) Register error

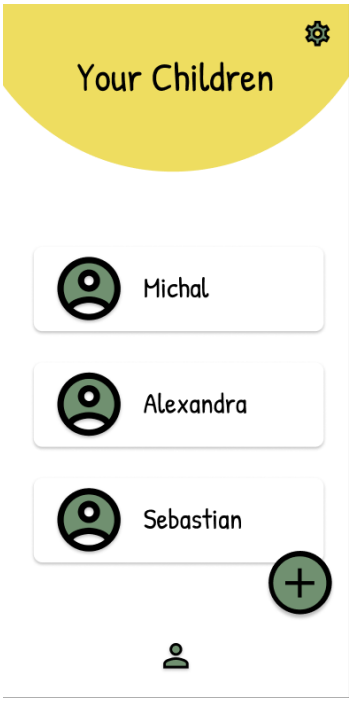


(5) Login screen

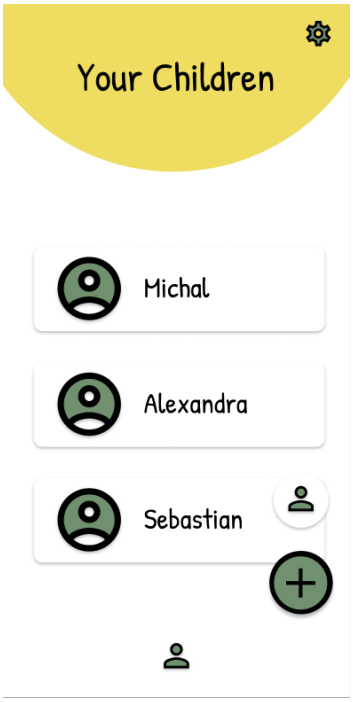


(6) Login error

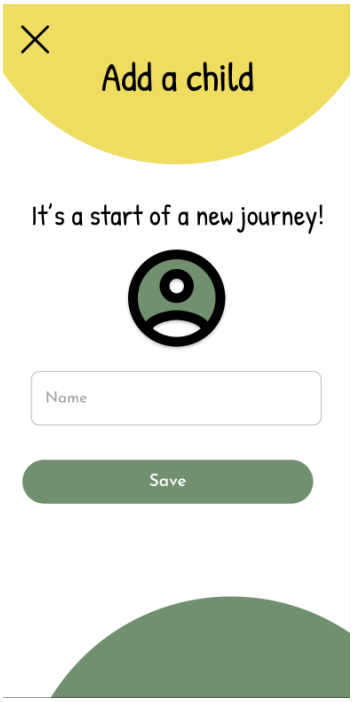
Figure 5.8: *Raise App* design screens



(7) Children's profiles list screen

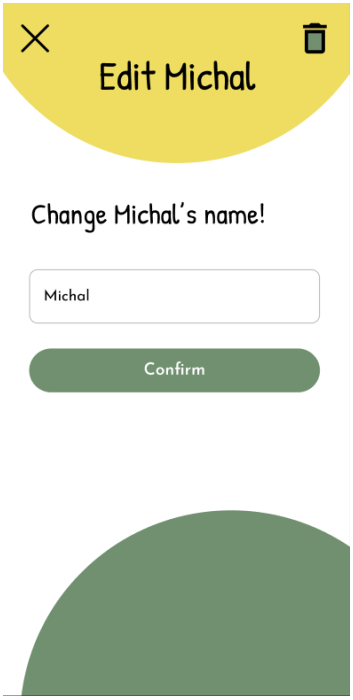


(8) Children's profiles list screen (add button)

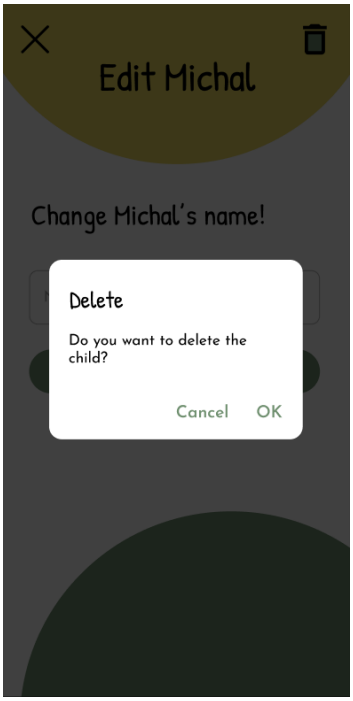


(9) Child's profile add screen

(10) Child's profile screen

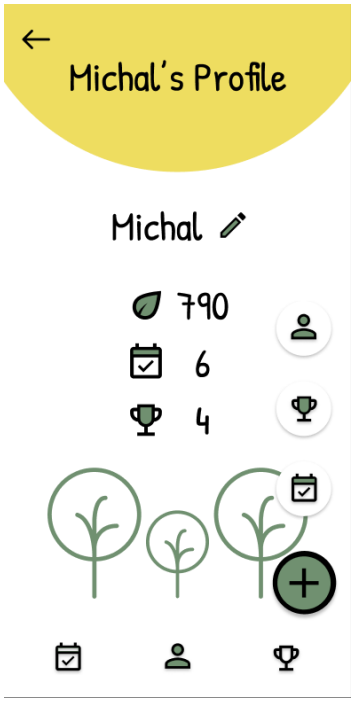


(11) Child's profile edit screen

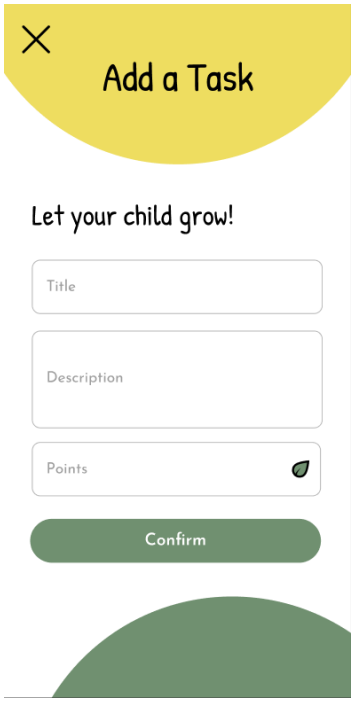


(12) Delete profile confirmation

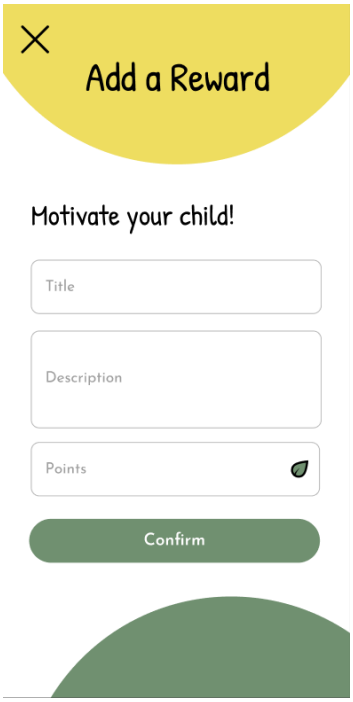
Figure 5.8: *Raise App* design screens (cont.)



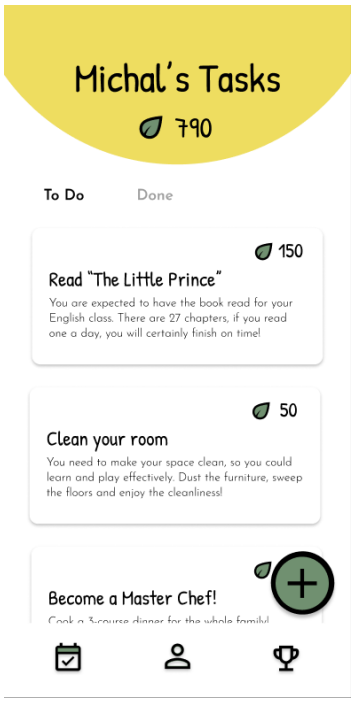
(13) Child's profile screen (add button)



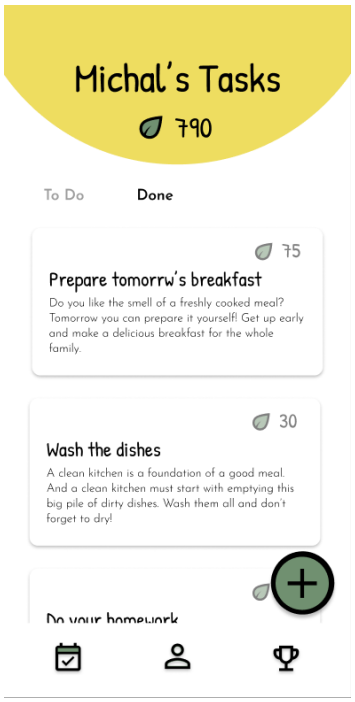
(14) Task add screen



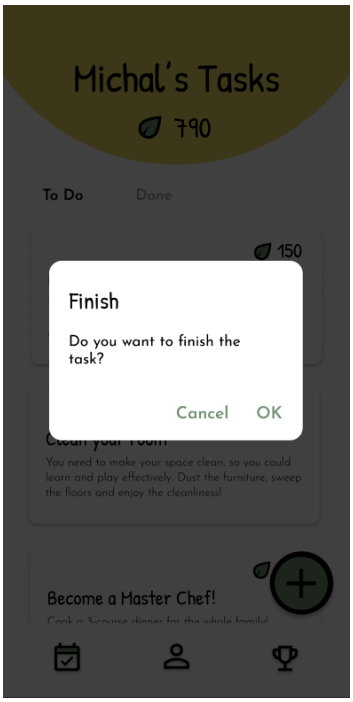
(15) Reward add screen



(16) Child's tasks screen (*To Do* filter)

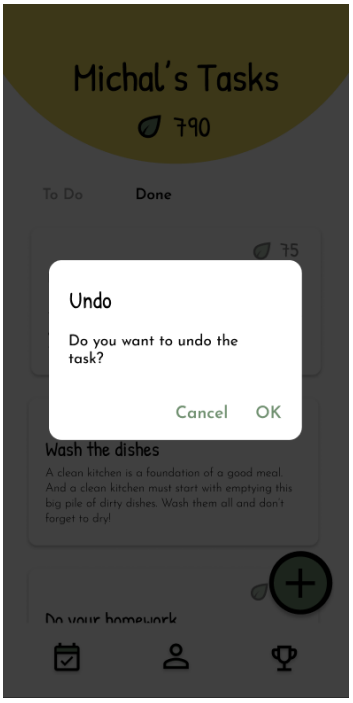


(17) Child's tasks screen (*Done* filter)

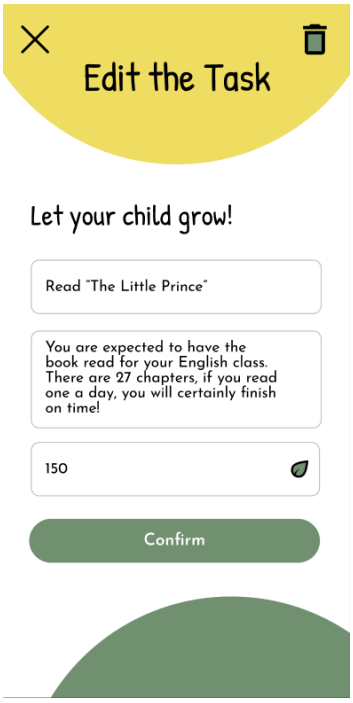


(18) Finish task confirmation

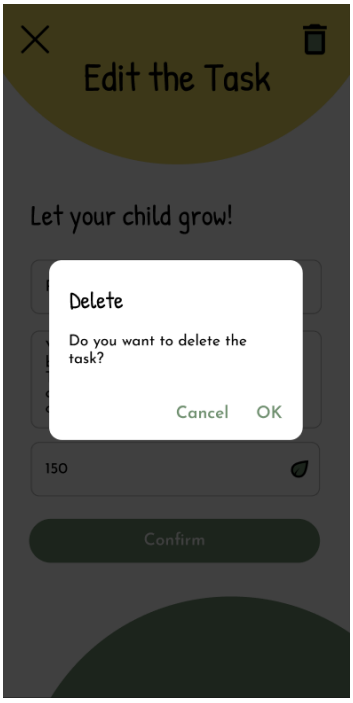
Figure 5.8: *Raise App* design screens (cont.)



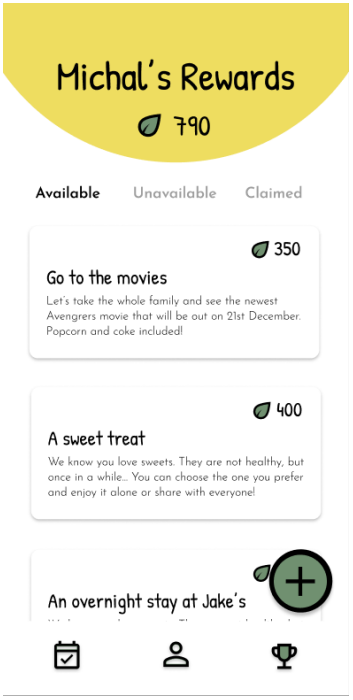
(19) Undo task confirmation



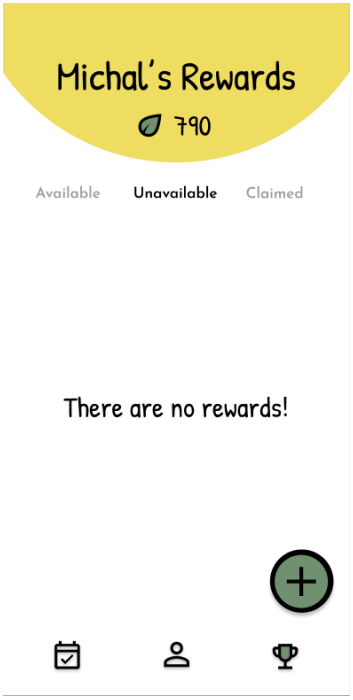
(20) Task edit screen



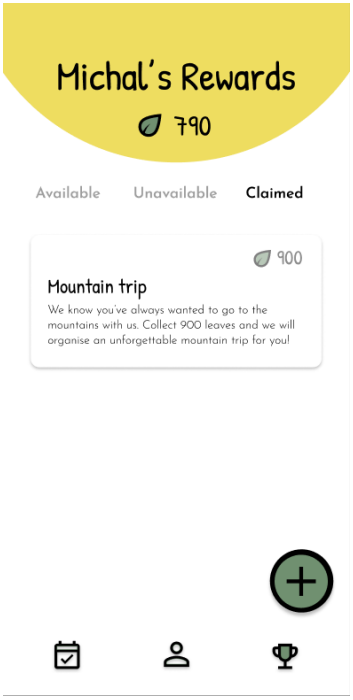
(21) Delete task confirmation



(22) Child's rewards screen (*Available* filter)

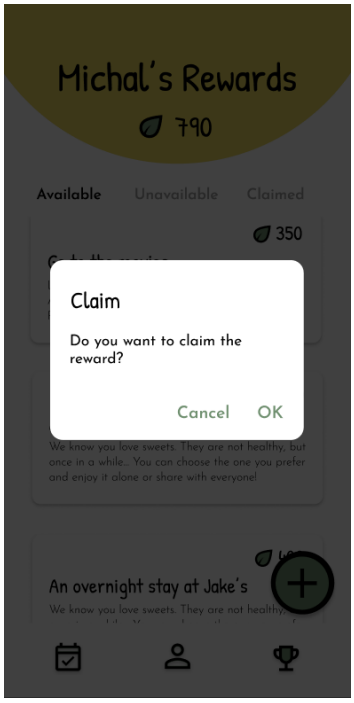


(23) Child's rewards screen (*Unavailable* filter)

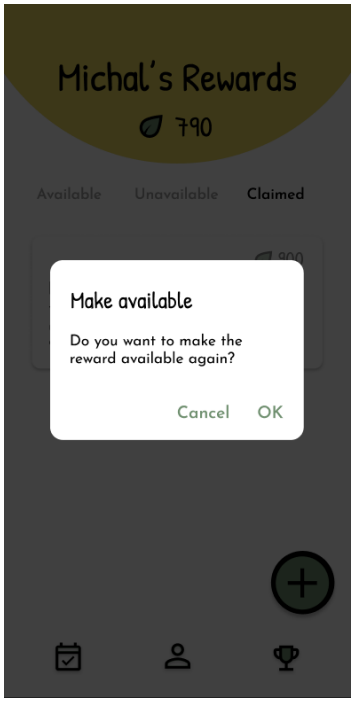


(24) Child's rewards screen (*Claimed* filter)

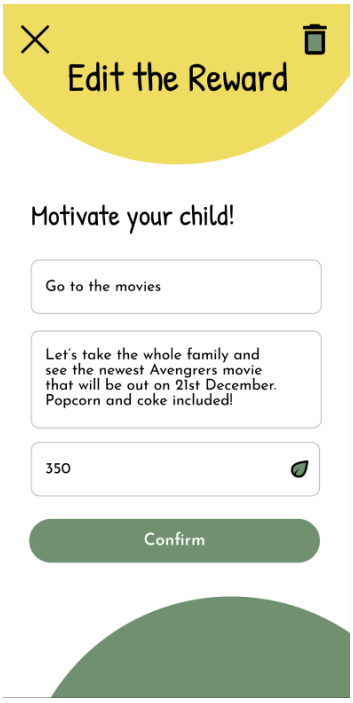
Figure 5.8: *Raise App* design screens (cont.)



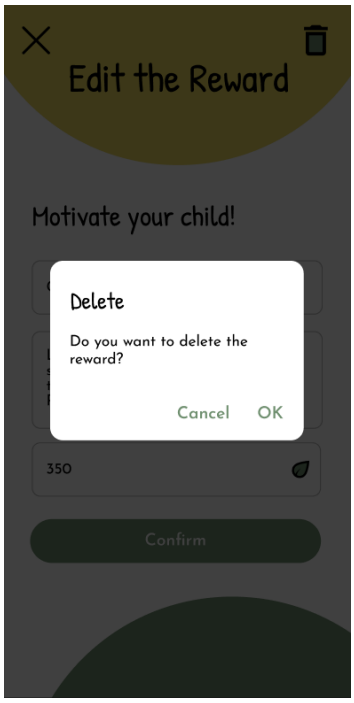
(25) Claim reward confirmation



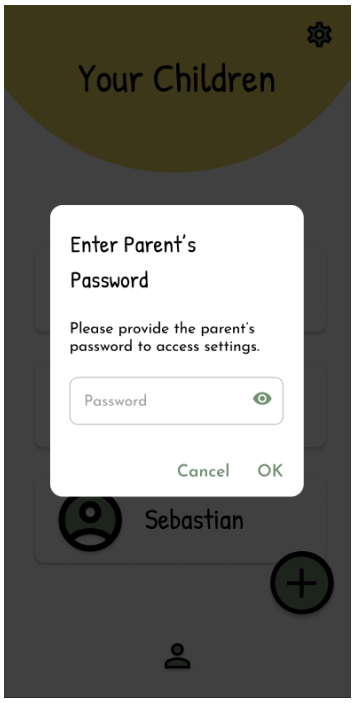
(26) Make reward available confirmation



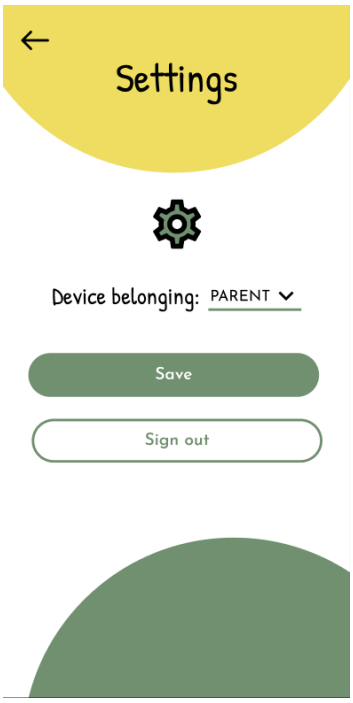
(27) Reward edit screen



(28) Delete reward confirmation

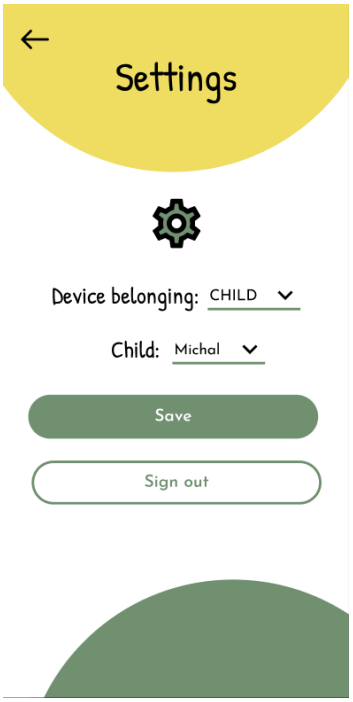


(29) Settings password prompt



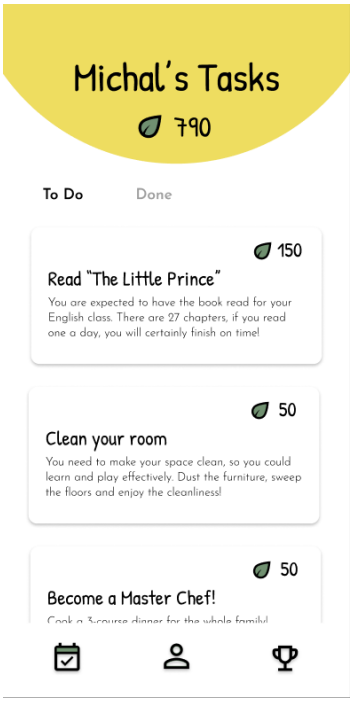
(30) Settings screen (parent's belonging)

Figure 5.8: *Raise App* design screens (cont.)

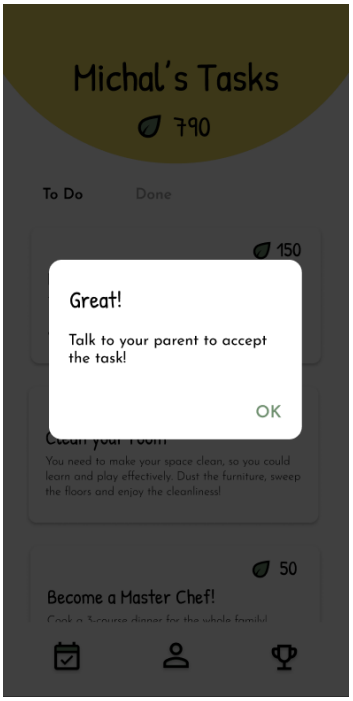


(31) Settings screen (child's belonging)

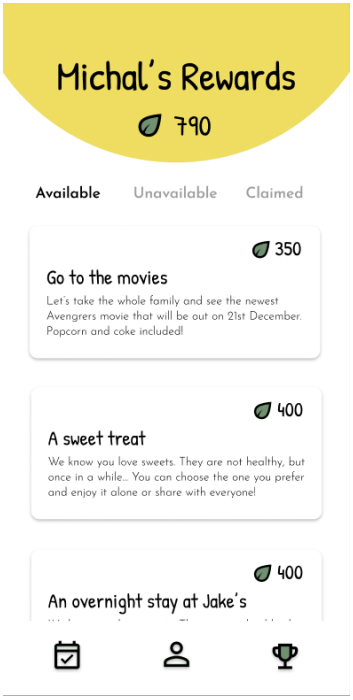
(32) Child's profile screen (child's perspective)



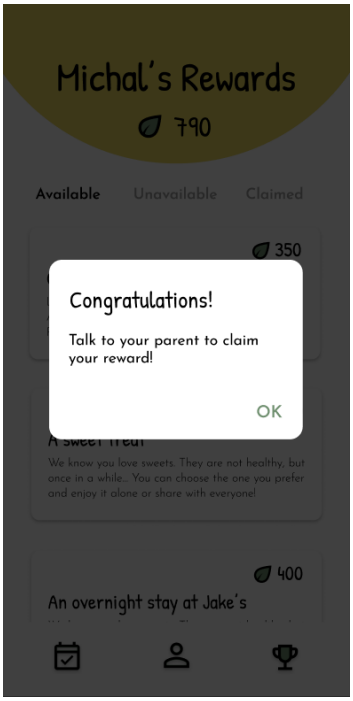
(33) Child's tasks screen (child's perspective)



(34) Finish task alert (child's perspective)



(35) Child's rewards screen (child's perspective)



(36) Claim reward alert (child's perspective)

Figure 5.8: *Raise App* design screens (cont.)

1. Splash screen appears during the application load Splash screen.

2. Belonging screen is displayed if no user and no device belonging is saved on the device. Here, the user can choose their perspective. Clicking the "Parent" card will navigate to the Register screen, the "Child" card - to the Login screen.

3. Register screen is displayed if the "Parent" device belonging is saved on the device. It is used to register an account. If the user inputs data and clicks the "Sign up" button, a register attempt is made. If it is successful, the application navigates to the Children's profiles list screen, if not - a Register error is displayed. Clicking the "Sign in instead" button will navigate to the Login screen (while maintaining the parent's device belonging).

4. Register error is displayed on unsuccessful register attempt.

5. Login screen is displayed if the "Child" device belonging is saved on the device, or the user navigates to it from the Register screen. If the user inputs data and clicks the "Sign in" button, a login attempt is made. If it is successful, the application navigates to the Children's profiles list screen, if not - a Login error is displayed.

6. Login error is displayed on unsuccessful login attempt.

7. Children's profiles list screen is a default screen if the "Parent" device belonging is saved and the user is logged in. It fetches children's profiles assigned to the parent and displays them as a list. After clicking a child card, the application navigates to the Child's profile screen. Clicking the settings button triggers settings access flow, which starts with the Settings password prompt.

8. Children's profiles list screen (add button); clicking the Floating Action Button reveals a profile icon (it is the only place in the application that only one icon is revealed). If the icon is clicked, the application navigates to the Child's profile add screen.

9. Child's profile add screen is displayed after the user clicks "Add new child's profile" button and allows for adding a new child's profile. If the user fills in the form and clicks the "Save" button, the child will be added to their account.

10. Child's profile screen is displayed after the parent chooses a child's profile from the Children's profiles list screen. It displays the chosen profile's name, number of points, tasks and rewards. After clicking the pencil icon, next to the name, the profile can be edited through the Child's profile edit screen. Clicking the back arrow will result in returning to the Children's profiles list screen. This is also the first screen that allows for using the bottom navigation. Clicking the task icon navigates to the Child's tasks screen (*To Do* filter), the reward icon - to the Child's rewards screen (*Available* filter) and the profile icon (while being on a different screen) - to the Child's profile screen.

11. Child's profile edit screen is displayed after the parent clicks a pencil icon on the Child's profile screen. The displayed form has a data pre-filled. After changing it and clicking the "Confirm" button, the changes are saved and displayed on the Child's profile screen. Clicking the bin icon in the top right corner, will show the

Delete profile confirmation. Clicking the "x" icon in the top left corner discards the edit and returns to the previous screen.

12. Delete profile confirmation is shown after the parent clicks a bin icon on the Child's profile edit screen. If the "Cancel" button is clicked (or the click occurs outside of the confirmation box), the deletion will be cancelled. If the "OK" button is clicked, the child will be deleted from the parent's account.

13. Child's profile screen (add button); clicking the Floating Action Button from any screen except Children's profiles list screen will reveal three icons - profile, reward and task. Clicking the child icon will work similarly to the one on the Children's profiles list screen. Clicking the reward icon will navigate to the Reward add screen, and the task icon to the Task add screen.

14. Task add screen allows for adding a new task to the currently managed child's profile. The task will be added after filling in all of the form's fields - title, description and number of points - and clicking the "Confirm" button (its default state is "To do"). Clicking the "x" icon in the top left corner discards the operation and returns to the previous screen.

15. Reward add screen allows for adding a new reward to the currently managed child's profile. It behaves analogically to the Task add screen (its default state, however, is either "Available" or "Unavailable", depending on the child's points).

16. Child's tasks screen (*To Do* filter) displays the tasks with a "To Do" status assigned to the currently managed child's profile. After tapping a task card (shortly), the Finish task confirmation will appear. If the card is pressed and held, the Task edit screen will be displayed. If there are no tasks with such criteria, a "There are no tasks" message is displayed.

17. Child's tasks screen (*Done* filter) works analogically to the Child's tasks screen (*To Do* filter), however, will display tasks with a "Done" status and after a tap, it will display the Undo task confirmation.

18. Finish task confirmation allows for the tasks status change from "To do" to "Done". It behaves analogically to other confirmations.

19. Undo task confirmation allows for the tasks status change from "Done" to "To do". It works analogically to other confirmations.

20. Task edit screen allows for a chosen task's details edit. It behaves analogically to the Child's profile edit screen, but with a Delete task confirmation.

21. Delete task confirmation works analogically to other delete confirmations, but relates to the task.

22. Child's rewards screen (*Available* filter) behaves analogically to the Child's tasks screen (*To Do* filter), however, displays list of rewards with an "Available" state (a reward is not claimed and the child has enough points to claim it). Short-press results in navigating to the Claim reward confirmation and press and hold to the Reward edit screen.

23. Child's rewards screen (*Unavailable* filter) works analogically to the Child's rewards screen (*Available* filter), but displays a list of rewards with an "Unavailable" state (a reward is not claimed and the child does not have enough points to claim it). It also does not provide a short-press action.

24. Child's rewards screen (*Claimed* filter) behaves analogically to the Child's rewards screen (*Available* filter), but displays a list of rewards with a "Claimed" state (a reward that has been claimed already). The press and hold results in presenting the Make reward available confirmation.

25. Claim reward confirmation allows for changing the reward status from "Available" to "Claimed". It works analogically to other confirmations.

26. Make reward available confirmation allows for changing the reward status from "Claimed" to "Available" (or, if the currently managed child's profile does not have enough points, to "Unavailable"). It works analogically to other confirmations.

27. Task edit screen allows for a chosen reward's details edit. It behaves analogically to the Child's profile edit screen, but with a Delete reward confirmation.

28. Delete reward confirmation works analogically to other delete confirmations, but relates to the reward.

29. Settings password prompt is displayed when the settings icon is clicked (either on Children's profiles list screen or Child's profile screen (child's perspective)). It requires re-authentication in order to display the Settings screen (parent's belonging) or Settings screen (child's belonging).

30. Settings screen (parent's belonging); displays settings in relation to the parent's perspective. It allows for changing the device belonging to a child's. After changing the settings, clicking the "Save" button is required in order to save them. Clicking the "Sign out" will clear the saved account from the device and navigate to the Belonging screen. The arrow icon in the top left corner of the screen return to the previous screen.

31. Settings screen (child's belonging) behaves analogically to the Settings screen (parent's belonging).

32. Child's profile screen (child's perspective) is a default screen if the device belongs to a child. It lacks (the same as all of the screens from this perspective) the "add" and "back" buttons, however, in order for the settings to be accessible, the icon was moved to this screen.

33. Child's tasks screen (child's perspective) works analogically to the parent's perspective, however, with no possibility to add or edit tasks. On short task card tap, it displays the Finish task alert (child's perspective).

34. Finish task alert (child's perspective) displays a message informing the child that they should consult their parents in order to have their task accepted.

35. Child's tasks screen (child's perspective) behaves analogically to the parent's perspective, however, with no possibility to add or edit rewards. On short reward card tap, it displays the Claim reward alert (child's perspective).

36. Claim reward alert (child's perspective) displays a message informing the child that they should consult their parents in order to have their reward claimed.

5.4 Database

5.4.1 Database type

When designing a prototype of an application, particularly one that does not depend on data in a great degree, but rather needs a scalable and flexible database solution, it is worth considering non-relational databases.

NoSQL is an umbrella term for non-relational, schema-less, highly scalable, modern approach to databases. Despite its name, it does not exclude SQL-based technologies (the "No" in the name was supposed to be a memorable Twitter hashtag) [50]. There are 4 primary NoSQL database types:

- document-oriented databases
- key-value databases
- wide-column databases
- graph databases

While wide-column databases are more like traditional relational model (the difference is in approach to performance), and graph databases are useful when relationships between data are important, the difference between a key-value and document-oriented database is not obvious. In brief, the document-oriented databases are more structured and provide more meta-data, which allows for more clarity along with flexibility [15]. Because of this, it will be the best choice for the project.

5.4.2 Database modelling

Due to NoSQL databases being a relatively fresh idea, there are no established standards in terms of database modelling. Nonetheless, articles proposing a new modelling standards exist [46, 35, 27] and will be used as a guide (but not a definite specification) in modelling the document-oriented database for the project.

Conceptual model

Conceptual model of the database gathers information from the business requirements and models them. In the project, four entities can be distinguished:

- Parent
- Child
- Task

- Reward

According to the requirements, a parent can have multiple children, who can have multiple rewards and tasks. Such relationship is presented on a conceptual model, in Figure 5.9.

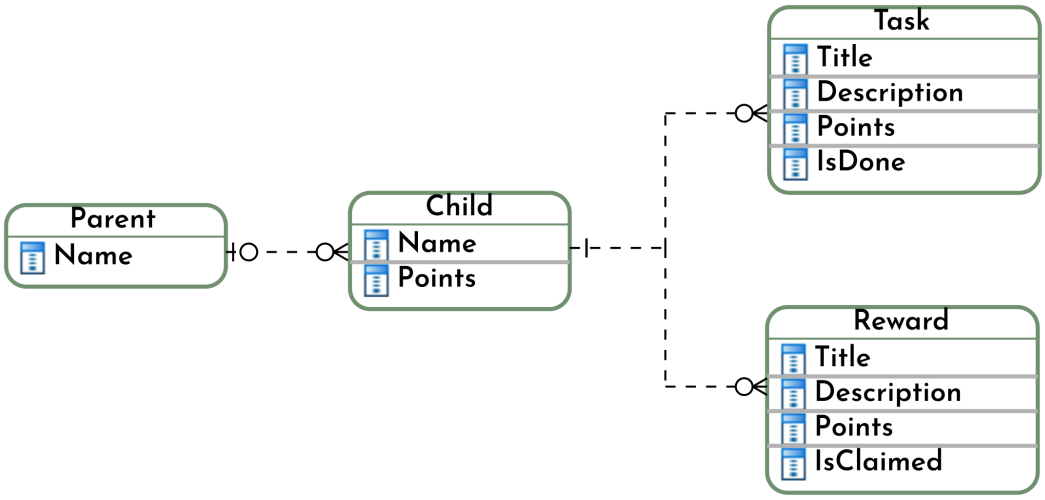


Figure 5.9: Conceptual model of the database

The model proves that, because of its nested structure, a document-oriented database type is a good choice. In this case, the model should be changed to represent a document structure. The conceptual model was amended following the guidelines proposed in the "Data Modelling for NoSQL Document-Oriented Databases" article [46] and is presented in Figure 5.10.

Other models

The aforementioned article does cover only the conceptual data and does not provide a way to represent logical, nor physical models. Authors of the "Modelling and Querying Data in NoSQL Databases" go even further and claim that storing in NoSQL databases eliminates the need for data modelling [35]. Because NoSQL databases are schema-less and focus, it is not necessary to model data with any other intention than to understand the data one needs to operate on. In the case of this project, it is covered by the conceptual model.

5.5 Summary

Project elements created in this chapter provided a broader perspective on the final product. Use case modelling allowed to understand the users' goals and decide which way can the application help accomplish them. Designing visual identity and prototyping the user interface simulated the look and behaviour of the real application. Finally, the database modelling resulted in a plan for its effective

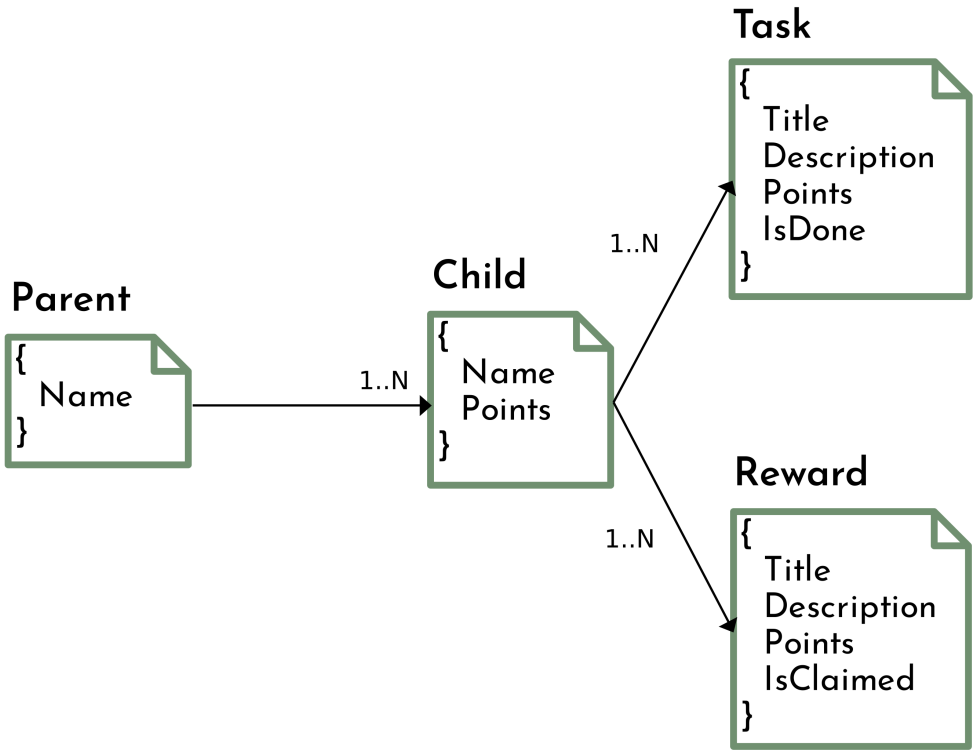


Figure 5.10: Conceptual model of the document-oriented database

implementation. Based on the results from this chapter, a suitable set of tools, technologies and techniques shall be chosen, in order to finish preparations for the implementation phase.

Chapter 6

Tools, technologies and techniques

6.1 Introduction

Implementation phase requires various tools, technologies and techniques in order to meet the requirements, be executed efficiently and accomplished. This chapter will introduce the most important ones and plan how to use them to improve the process and deliver the product.

6.2 Cross-platform framework

In the duopoly world of Google and Apple, to create a mobile application for both platforms, one needs two different projects with two different technologies, and will usually pay twice the price. However, if the performance and security is not required to match the native applications, it is worth considering a cross-platform solution.

Cross-platform in the context of mobile applications is a relatively recent term that describes a set of approaches that assume maintaining a single code base (most often in the same technology) targeting multiple platforms. Not only can they compete with native solutions in some areas, but also conquer - principally in prototyping or simple applications. In recent years numerous technologies emerged, but especially two of them have outgrown the rest.

React Native[18] is an open-source, *interpreted* application, meaning that despite relying on web technologies in terms of development, they do not run in a browser, a web view or a container, but rather emulate a native experience. It currently has 92,000 stars on Github[19] and is maintained under the patronage of Facebook.

Flutter[20] is open-source as well, however, it is a compilation-based approach to being cross-platform. Instead emulating native components, it maps a code written in Dart language[21] to the desired native code. At the time of writing, it has 109,000 stars on Github[22] and is developed by Google.

[18]https://reactnative.dev (accessed Dec. 06, 2020).
[19]https://github.com/facebook/react-native (accessed Dec. 06, 2020).
[20]https://flutter.dev (accessed Dec. 06, 2020).
[21]https://dart.dev (accessed Dec. 06, 2020).
[22]https://github.com/flutter/flutter (accessed Dec. 06, 2020).

Due to Flutter's first version release only in May 2017, there is not many academic sources that compare the two in terms of performance. However, it can be assessed based on the technologies' specification. Comparison of the two from the development point of view, slightly subjective and based on personal experience, is presented in Table 6.1.

Aspect	Technology	
	Flutter	React Native
UI components	Default package provides numerous components with consistent look across platforms (able to imitate platform-specific look).	Uses native components, gives more flexibility in terms of customisation, but takes longer to achieve some results.
Package library	Separation of verified *publishers* and unverified *uploaders* on *pub.dev*[23,24], resulting in a better packages quality.	Anyone being able to publish to the *npm registry*[25], which sometimes results in lower quality and information overload.
Documentation	Extensive, detailed, centralized.	Not as extensive, detailed, decentralized (responsibilities shifted onto particular packages).
Development	Custom, fresh ecosystem, but providing just enough tools.	Utilizing extensive web ecosystem improved for years.
Maturity	Quite young, yet mature; growing rapidly.	Mature and growing.

Table 6.1: Comparison of *React Native* and *Flutter*

All things considered; Flutter appears to be the preferred technology for the mobile application. It will not only meet the requirements (described in Chapter 4, but also bring benefits, such as accelerating the development process.

[23]Flutter package registry publishing guide, https://pub.dev/help/publishing (accessed Dec. 06, 2020).

[24]Flutter package registry verification process, https://dart.dev/tools/pub/verified-publishers (accessed Dec. 06, 2020).

[25]Node Package Manager registry publishing guide, https://docs.npmjs.com/about-the-public-npm-registry (accessed Dec. 06, 2020).

6.3 System architecture

6.3.1 Basic system architecture

According to the requirements (NFR 1.), the system will consist of a mobile application run on a mobile device and two services - authentication and database. A basic representation of such architecture is depicted in Figure 6.1.

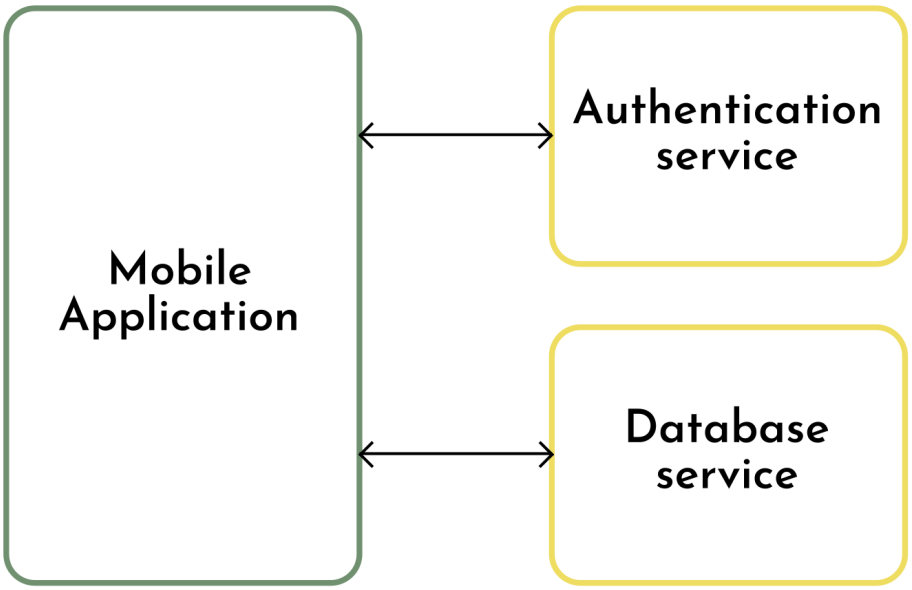


Figure 6.1: Diagram of the basic system architecture

The mobile application part of the system will be further described in this section. Services will be introduced in Section 6.4.

6.3.2 State management

Due to Flutter being a declarative framework (in contrast with imperativeness of Android or iOS programming)[26], the interface is an immediate result of its state. This approach has many benefits, one of which is one codebase for the UI and UI-logic.

While the widgets (a base component of the Flutter framework) frequently rely on internal, "ephemeral" state alone, dynamic applications (relying on an external source of data) induce a need for a global, "app" state[27]. The most elementary approach to managing the app state is passing it down the widget tree. It works, however, only within descendants line of the parent widget. Therefore, it is not enough for more complex applications.

There are various state management approaches. From an *InheritedWidget*[28] (a

[26]https://flutter.dev/docs/get-started/flutter-for/declarative (accessed Dec. 11, 2020)
[27]https://flutter.dev/docs/development/data-and-backend/state-mgmt/ephemeral-vs-app (accessed Dec. 11, 2020)
[28]https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html (accessed Dec. 11, 2020)

slightly more advanced version of the pass-down strategy) a stream-based *BLoC*[29], to the framework-agnostic *Redux*[30] and *MobX*[31]. Nonetheless, the most modern, and recommended approach is a *Provider*[32].

The Provider is a state management helper that distributes values produced by custom classes extending it. In order for it to work, the whole application (or the most top-level widget that needs it) is supposed to be wrapped in a Provider (or a *MultiProvider* in case of multiple providers) widget. Receiving the value is performed by a *Consumer* widget that listens for the value modifications and rebuilds the child widget in case of change. A simplified diagram of this behaviour is presented in Figure 6.2.

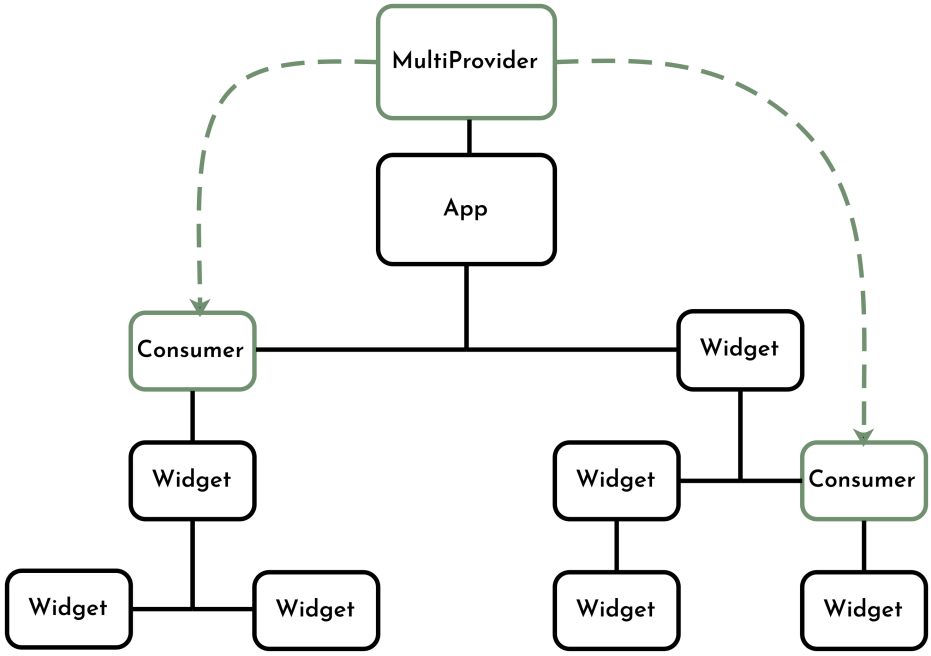


Figure 6.2: Diagram visualizing state management with Provider

6.3.3 Design patterns

As a result of adopting multiple frameworks and libraries, various design patterns might be used underneath (i.e. an *Iterator* design pattern used by `forEach()` and `map()` methods or a *Builder* design pattern used for *context* or *provider* creation). In this section, however, only the patterns used explicitly and important from the implementation point of view will be mentioned.

Model–view–viewmodel

Model-view-viewmodel (abbreviated *MVVM*) is a structural design pattern. It is a modification of Martin Fowler's *Presentation Model* [20] and originally introduced by Microsoft [52]. MVVM divides the application into three layers:

[29]https://bloclibrary.dev (accessed Dec. 11, 2020)
[30]https://pub.dev/packages/redux (accessed Dec. 11, 2020)
[31]https://pub.dev/packages/mobx (accessed Dec. 11, 2020)
[32]https://pub.dev/packages/provider (accessed Dec. 11, 2020)

- *Model* - data access layer; represents content.
- *View* - presentation layer; analogical to the *view* in the *Model-View-Controller* or *Model-View-Presenter* patterns; connected to view model via data binding.
- View model - business logic layer; it holds the state of the data.

The structural design pattern that will be used in the Mobile Application is confusingly similar to the MVVM (with hardly any exceptions) and will be treated as such.

Observer

As already mentioned in Section 6.3.2, in order to update after the value change, the *Customer* implements an *Observer* and the widget extending a *ChangeProvider* implements an *Observable*.

6.4 Services

Both the services will utilize *Backend as a Service*[33] model. The choice is motivated by the performance, scalability and availability of cloud computing. It allows for shifting the resources from implementing and maintaining a repetitive functionality, such as handling database or authentication requests, to developing the primary functionality.

Despite the absence of the *Function as a Service*[34] services, such architecture should be considered a *serverless*[35] architecture [49]. Even more they could be easily incorporated if a need arises.

6.4.1 Database service

Out of a numerous group of BaaS NoSQL service providers (some of them being Amazon, MongoDB, Alibaba, Microsoft or IBM), one of the most user-friendly is *Cloud Firestore*[36] from the Google's *Firebase*[37] platform. It promises over 99.99% uptime[38] and has an advantage of being exquisitely well-integrated with the Flutter language, which is also developed by Google.

6.4.2 Authentication service

While there are many providers offering BaaS authentication solutions, the Firebase suite includes it as well. Considering the aforementioned advantages and an added benefit of adopting solutions from the same provider (most importantly quicker implementation due to knowledge and code reuse), it will be used in the project.

[33] "Backend as a Service", https://www.techopedia.com/definition/29428/backend-as-a-service-baas (accessed Dec. 06, 2020).
[34] "Function as a Service", https://www.techopedia.com/definition/32478/function-as-a-service-faas (accessed Dec. 06, 2020).
[35] https://www.techopedia.com/definition/32475/serverless-architecture (accessed Dec. 06, 2020).
[36] https://firebase.google.com/docs/firestore (accessed Dec. 07, 2020)
[37] https://firebase.google.com (accessed Dec. 07, 2020)
[38] https://cloud.google.com/firestore/sla (accessed Dec. 07, 2020)

6.5 Project management software

Project management software will help to plan and organise work and resources throughout the implementation phase. As the project is conducted using *Agile* methodology, the tool needs to support it. One of industry leading management tools is an issue tracker called *Jira Software*[39].

Jira provides an extensive set of features for supporting a sprint-based, *Scrum*[40] methodology. As a consequence of a one-person team being involved in the process, only several will be used. During the implementation phase following rules will be adopted:

1. A *task* is the smallest workable unit.
2. An *epic* consists of multiple tasks (at least one) and concerns a bigger feature or set or features (i.e. a specific screen functionality).
3. A *sprint* consists of multiple epics (at least one) and concerns a set of features that bring a new functionality to the application, so that after the sprint finishes, the system is working, and it has at least one new feature or fix.
4. A task must be estimated before the sprint start, and the time must be logged against it before the sprint ends.
5. There are three task statuses - *To Do*, *In Progress* and *Done*. The statuses must be updated in the system immediately after it changes.

6.6 Version control system

6.6.1 Introduction

"Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later" [6]. It not only allows backups of every stage of the implementation process, but also supports nonlinear work on the project, which happens frequently in agile methodologies. It will accelerate the development process of the project.

Git, a distributed version control system, has already become an industry standard. According to the 2018 Developer Survey conducted by StackOverflow[41], almost 90% of the 70000 developers that answered the question use Git as a primary version control tool. It is renowned for its very efficient *branching*[42].

6.6.2 Branching model

Adopted *branching model* needs to reflect the management processes on one hand, however, ensure a secure, stable and effective development on the other. As a result

[39] https://www.atlassian.com/software/jira (accessed Dec. 11, 2020)
[40] https://www.scrum.org/resources/what-is-scrum (accessed Dec. 11, 2020)
[41] https://insights.stackoverflow.com/survey/2018 (accessed Dec. 07, 2020)
[42] https://git-scm.com/book/en/v2/Git-Branching-Branches-in-a-Nutshell (accessed Dec. 08, 2020)

of applying advice from the Martin Fowler's "Patterns for Managing Source Code Branches" [19] and the Vincent Driessen's "A successful Git branching model" [16] articles, a branching model was created:

1. The `master` branch is always up-to-date with the production code.
2. The `dev` branch is an "integration" branch allowing for joining the latest development changes for a release. It must not be merged into the `master` branch, however, kept up-to-date with it.
3. The `epic/` branches, followed by the Jira epic code (i.e. `epic/RA-1`), must be sourced out of the `master` branch and, if possible, keep only the code related to the epic's work. If other code is needed for the development, it should be merged into it directly from other `epic/` branches.
4. The `feature/` branches, followed by the Jira task code (i.e. `feature/RA-10`), must be sourced out of an `epic/` branch corresponding to the epic it belongs to in the Jira Software. The `feature/` branches shall not be merged into one another, but rather through `epic/` branches. After the work is finished, they must be merged into a corresponding `epic/` branch.
5. The `release/` branches, followed by a version number (i.e. `release/1.0.0`), must be created independently, from the existing `epic/` branches that are included in the release. After they are created and tested, they shall be merged into the `master` branch.
6. The remote repository branches, named `origin/*`, where `*` is a corresponding name of a local branch, should be:
   (a) in case of `feature/` and `epic/` branches, be updated with the work from local branches as often as possible,
   (b) in case of `dev` and `master` branches, update the local branches as often as possible.
7. All the branch merges must be done through the remote repository *pull request* feature with a branch name, followed by a corresponding Jira ticket title.

6.6.3 Remote repository

Remote repository will be hosted in a *Bitbucket*[43] remote repository. The primary reason for it (as feature-wise it is similar to its competition) is its integration with the *Jira Software* (both tools are created by *Atlassian*[44]) that allows for live status updates between the two.

Remote repository will make the code independent of the local machines (serving as a backup source), bring additional safety to the branch merges through the pull-request feature, allow for continuous deployment and provide a more clear visual representation of the version control processes.

[43] https://bitbucket.org/ (accessed Dec. 11, 2020)
[44] https://www.atlassian.com (accessed Dec. 10, 2020)

6.7 Static code analysis

Static code analysis allows discovering code-issues before the code is even executed. If configured properly, it unifies the code's making improving its maintainability and prevents various issues from happening on a syntax level, which directly translates to better development efficiency and code quality [55].

A set of rules used in the project will be adopted from the Google's recommended[45] package `pedantic`[46]. It will ensure the code is up to standards and help deliver the product without potential delays caused by code syntax issues (i.e. merge conflicts in source version control).

[45] https://dart.dev/guides/language/analysis-options (accessed Dec. 11, 2020)
[46] https://pub.dev/packages/pedantic (accessed Dec. 11, 2020)

Chapter 7

Implementation

7.1 Introduction

Having the requirements specified, a project created, and tools chosen, the implementation phase may begin. Section 7.2 will introduce the remaining tools used during the development. A package diagram presented in Section 7.3 will help depict the final high-level architecture of the system. Section 7.4 will describe challenges encountered during the implementation. The application installation and usage will be described in Section 7.5 In the Summary (Section 7.6), the implementation phase and its steps will be summarized.

7.2 Development Environments and Tools

In addition to the tools mentioned in Chapter 6, various development environments and tools were used. They will be briefly described in this section.

7.2.1 Android Studio

Android Studio[47] is the official IDE (*Integrated Development Environment*) for Android development. Since the Flutter release, it also supports this framework (only compilation to Android). It provides an extensive functionality - from standard code editor, through debugging tools, to device emulators. Because it is a project derived from the IntelliJ IDEA Community Edition[48], it is distributed under an Apache 2 license[49]), which is an open-source license and allows for development within the needs of the project.

7.2.2 Flutter console

Flutter console allows for the Flutter application development management. The most used commands are:

- `flutter analyze` - performs a static code analysis.

[47]https://developer.android.com/studio (accessed Dec. 12, 2020)
[48]https://www.jetbrains.com/idea (accessed Dec. 12, 2020)
[49]https://www.apache.org/licenses/LICENSE-2.0 (accessed Dec. 12, 2020

- `flutter build` - builds the application.
- `flutter doctor` - shows information about the installed tooling and allows to detect issues.
- `flutter pub` - allows for Flutter packages management.
- `flutter run` - runs the Flutter application on an attached device (allows for running, controlling and debugging multiple devices at a time).

Whilst it is possible to run most of them through the IDE, it is more convenient to do it from the terminal (especially if performed often).

7.3 High-Level Architecture Overview

Flutter's projects structure (high level of modularity and numerous classes related to the view layer of the application) causes the class diagram to be overcomplicated. The deployment diagram would be relatively similar to the basic architecture diagram presented in Section 6.3.1. However, it is not detailed enough to provide a comprehensive overview of the implemented system. In order to illustrate the high-level architecture of the application, a simplified package diagram was created and is presented in Figure 7.1.

The mobile application, following the MVVM design pattern, was divided into 3 main packages.

UI (*User Interface*) consists of three packages. *Screens* package contains application screen classes (i.e. `LoginScreen` or `RewardsScreen`). *Components* are classes that are reused (at least twice) in within the *Screens* package. *Utils* package contains functions responsible for displaying modal windows.

Business logic package consists of the *Utils* (which plays a similar role as in the *UI* package) and *View Models* packages. View models depend on two packages that allow for communication with services. If the services were implemented in a traditional manner, they would be a part of the *Business logic*, however, to emphasize their independence, they were depicted separately. Both service packages depend on the service subsystems.

Models package contains models (such as *task*, *child*, *parent* or *reward*) and enumerations (application login state or device belonging).

7.4 Implementation challenges

Every project has its challenges. Ideally, they should be detected and prevented (or at least minimize their impact). Reality, however, is never ideal. It is important to identify those challenges and draw conclusions from them. The latter will be done in the Chapter 9, the former will be described in this section.

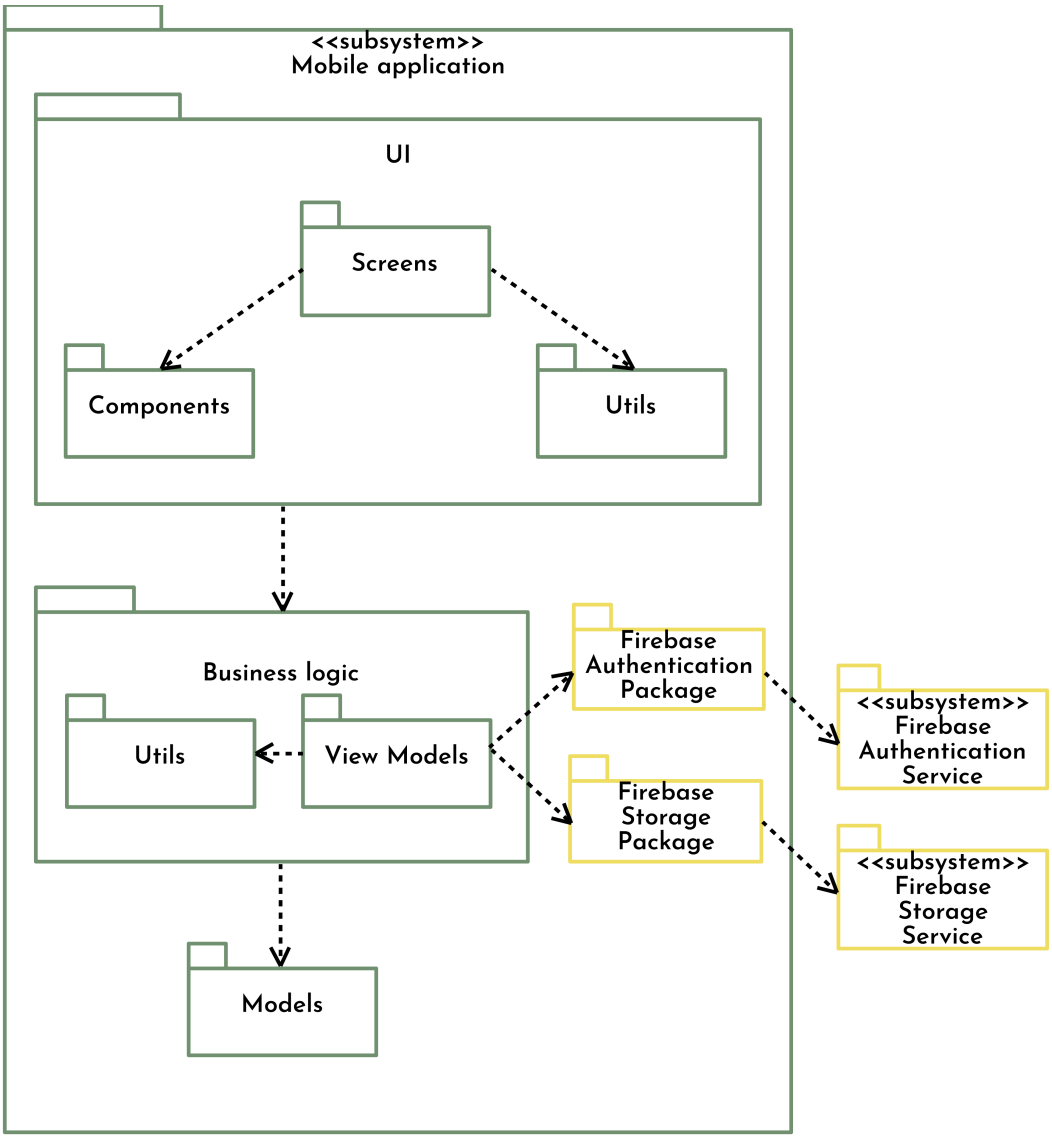


Figure 7.1: A simplified package diagram of the implemented system

7.4.1 User Interface

Many custom, atypical user interface elements, especially the header, could not be implemented using standard techniques. These assume the customisation of the `AppBar`[50]widget, or, in more complicated cases, `SliverAppBar`[51]. Due to the complexity of the user interface visuals, a `Canvas`[52] class was used.

The `Canvas` class is an interface for recording graphical operations. While being much more complex than other widgets, provides a higher flexibility and control over its visual output.

It was used to implement the header, along with the footer, but not without difficulties. During the development, issues with interoperability with other widgets

[50]https://api.flutter.dev/flutter/material/AppBar-class.html (accessed Dec. 13, 2020)
[51]https://api.flutter.dev/flutter/material/SliverAppBar-class.html (accessed Dec. 13, 2020)
[52]https://api.flutter.dev/flutter/dart-ui/Canvas-class.html (accessed Dec. 13, 2020)

emerged. Its initially infinite clip region, repeatedly caused problems with overlapping or being overlapped by other components. Another vital problem emerged during the header title implementation. Text on Canvas, instead of being an embedded `Text`[53] widget, needs to be rendered by a `TextPainter`[54] class. It caused less control over how the text is displayed.

7.4.2 Provider

State management using `Provider` and the reasons for its adoption have been described in Section 6.3.2. During the implementation, with increasing number of Provider classes, their maintenance had started to become slightly cumbersome and ineffective. In the final implementation there are 11 different Provider classes implemented and would increase, in case of the application growth.

Additionally, sometimes Providers depend on one another. This enforces a usage of `ProxyProviders`, which take other Providers as a constructor argument. A dependency created this way, causes even more complication and boilerplate code.

7.5 Installation

The development project has been built into a `raise-app.apk` android package file and is attached with this document (in order to install the application from outside of the Google Play platform, special permissions might be required).

The newest version, however, can be found on the Google Play platform under this link[55].

7.6 Summary

Over the course of the implementation phase, a full application planned during the previous phases has been completed. Thanks to the project phase, where the application was planned, and despite occasional challenges that occurred, the process has been finished with a product that is ready for the end-user testing and release.

[53] https://api.flutter.dev/flutter/dart-html/Text-class.html (accessed Dec. 13, 2020)
[54] https://api.flutter.dev/flutter/painting/TextPainter-class.html (accessed Dec. 13, 2020)
[55] https://play.google.com/store/apps/details?id=dev.karbownik.raise_app (accessed Dec. 14, 2020)

Chapter 8

Tests

8.1 Introduction

Don Norman, in his book "The Design of Everyday Things" [44] said that "a lack of feedback creates a feeling of lack of control, which can be unsettling". If the word *feedback* is replaced with *testing*, a result is a sentence that maintains its message and perfectly describes any project. Testing not only validates the finished implementation, but also creates a closed feedback loop that allows for an early recognition of discrepancies between the initial plans and the implementation at any given time.

Flutter testing documentation[56] describes three levels of testing - *unit*, *widget* and *integration*. While it is enough for the aforementioned closed feedback loop during the implementation phase, it lacks an element that will validate the result of it - *user acceptance testing*. As already mentioned, first three levels had been executed during the implementation process. The last level of testing was performed after the application was released to production in its first version.

Testing within the project requires additional packages to be installed. Four (recommended by Google) needed to be added to the project's dependencies:

- *test*[57] - provides the core framework for writing tests[58],
- *flutter_test*[59] - provides additional utilities for testing widgets[60],
- *flutter_driver*[61] - provides tools to create instrumented apps and drive those apps from a test suite[62].
- *mockito*[63] - allows for *mocking* dependencies that are not stable and could fail during the test run despite the tested functionality working properly.

[56] https://flutter.dev/docs/testing (accessed Dec. 12, 2020)
[57] https://pub.dev/packages/test (accessed Dec. 12, 2020)
[58] https://flutter.dev/docs/cookbook/testing/unit/introduction (accessed Dec. 12, 2020)
[59] https://api.flutter.dev/flutter/flutter_test/flutter_test-library.html (accessed Dec. 12, 2020)
[60] https://flutter.dev/docs/cookbook/testing/unit/introduction (accessed Dec. 12, 2020)
[61] https://api.flutter.dev/flutter/flutter_driver/flutter_driver-library.html (accessed Dec. 12, 2020)
[62] https://flutter.dev/docs/cookbook/testing/integration/introduction (accessed Dec. 12, 2020)
[63] https://pub.dev/packages/mockito (accessed Dec. 12, 2020)

8.2 Unit tests

Unit tests, as the name suggests, are designated for testing single units of the code - functions, methods or classes. They should be concise and as atomic as possible. An example of the unit test is presented in Listing 8.1.

```
test('getUserHomeScreen() returns belonging route on null deviceBelonging
    parameter', () {
  expect(RouterCustom.getUserHomeScreen(null, null),
      RouterCustom.BELONGING_ROUTE);
  expect(RouterCustom.getUserHomeScreen(null, 'childId'),
      RouterCustom.BELONGING_ROUTE);
});
```

Listing 8.1: A project unit test example

The test checks if the `getUserHomeScreen()` method, responsible for returning an initial home screen, returns a Device Belonging Screen in case of `deviceBelonging` parameter, responsible for storing the device assignment, having no saved value (being equal to `null`).

8.3 Widget tests

User interface testing needs cannot be covered with a standard unit testing. *Widget tests* utilize special methods for analysing the visual components building the application. They ensure the widgets are interactive and contain all expected elements. Listing 8.2 contains an example widget test from the project.

```
testWidgets('Belonging Screen displays necessary elements',
(WidgetTester tester) async {
    await tester.pumpWidget(
      MaterialApp(
        home: BelongingScreen(),
      ),
    );

    expect(find.text('This device belongs to a ... '), findsOneWidget);
    expect(find.widgetWithText(Card, 'Parent'), findsOneWidget);
    expect(find.widgetWithText(Card, 'Child'), findsOneWidget);
  });
```

Listing 8.2: A project widget test example

The test checks that Belonging Screen contains “This device belongs to a...” text and two cards, one with ’Parent’ and one with ’Child’ text on it.

8.4 Integration tests

While unit and widget tests are sufficient for testing individual functions and components, they do not provide means of testing the system as a whole, while running on a real device. Generally, this kind of testing is called an *integration testing*, and in Flutter, it is possible thanks to the `flutter_driver` package. Integration tests require a real device (or its emulation) to be run on. An example test from the project is shown in Listing 8.3.

```
group('Parent login scenario', () {
    final parentCardFinder = find.byValueKey('Parent');
    final singInInsteadButtonFinder = find.byValueKey('Sign in instead');
    final singInButtonFinder = find.byValueKey('Sign In');

    FlutterDriver driver;

    setUpAll(() async {
      driver = await FlutterDriver.connect();
    });

    tearDownAll(() async {
      if (driver != null) {
        await driver.close();
      }
    });

    test('Clicks the parent card', () async {
      await driver.runUnsynchronized(() async {
        await driver.waitFor(find.byType('BelongingScreen'));
        await driver.waitFor(parentCardFinder);

        expect(await driver.getText(parentCardFinder), 'Parent');
      });
    });

    test('Navigates to Register Screen', () async {
      await driver.runUnsynchronized(() async {
        await driver.tap(parentCardFinder);
        await driver.waitFor(singInInsteadButtonFinder);

        expect(
            await driver.getText(singInInsteadButtonFinder), 'Sign in instead');
      });
    });

    test('Navigates to Login Screen', () async {
      await driver.runUnsynchronized(() async {
        await driver.tap(singInInsteadButtonFinder);
```

```
        await driver.waitFor(singInButtonFinder);

        expect(await driver.getText(singInButtonFinder), 'Sign in');
      });
    });
  });
```

Listing 8.3: A project integration test example

The test follows a path of a Parent user, who installs the application on a new device and wants to login to their account. The following steps are executed:

1. Turn on the application.
2. Wait for Belonging Screen to load.
3. Check that the *Parent* card is present.
4. Click the *Parent* card.
5. Wait for the *Sign in instead* button (it is only present on the Register screen).
6. Check that the *Sign in instead* button is present.
7. Click the *Sign in instead* button.
8. Wait for the *Sign in* button (it is only present on the Login screen).
9. Check that the *Sign in* button is present.

8.5 Cross-device tests

8.5.1 Test execution

The following section presents the application behaviour on various devices, thus proving it meets the requirements of cross-device (not cross-platform) compatibility (including the responsiveness). The tests will be conducted on three devices (two physical and one emulated) with different Android operating system versions and different screen sizes:

1. Samsung Galaxy A50 with Android 10 and 6.4 inches diagonal screen size.
2. Samsung Galaxy S5 with Android in version 6.0.1 and 5.1 inches diagonal screen size.
3. An emulated device with Android 9 and 3.7 inches diagonal screen size.

The screens used in the test were chosen in such way that the same types of layout are not duplicated. The screens are:

1. Belonging screen
2. Login screen

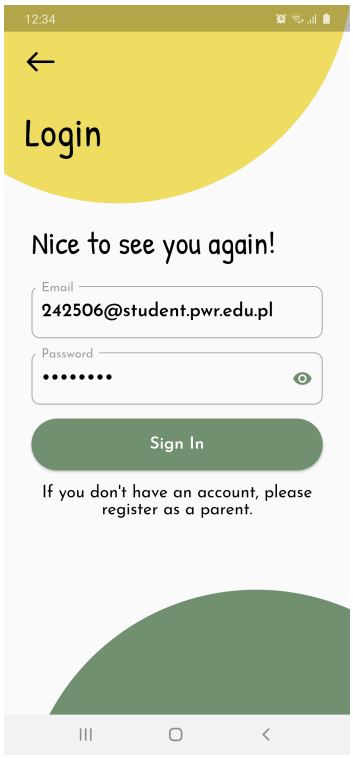
3. Children's profiles list screen
4. Child's tasks screen
5. Task edit screen
6. Delete task confirmation

The results of the test are presented in Figures 8.1, 8.2 and 8.3.

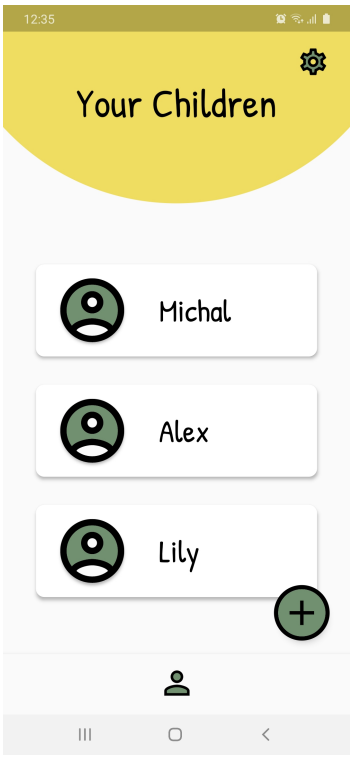
8.5.2 Summary

The presented screenshots from different devices prove that the cross-device requirement has been satisfied. The layout displays correctly on all of the provided screen sizes, which means the application is responsive. It also functions properly, no matter the device it is used on. It ensures that the end users get a high-level experience without interruptions.

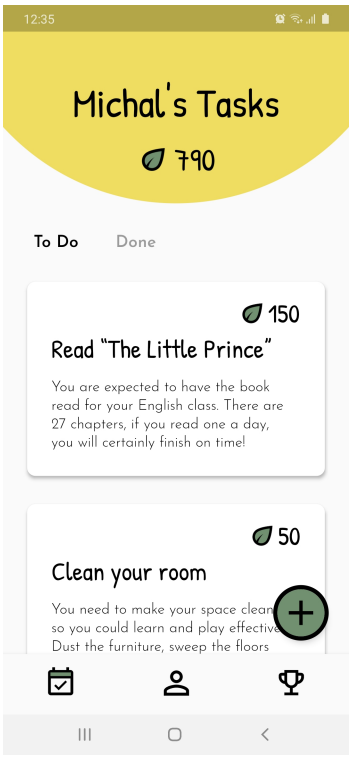
(1) Belonging screen



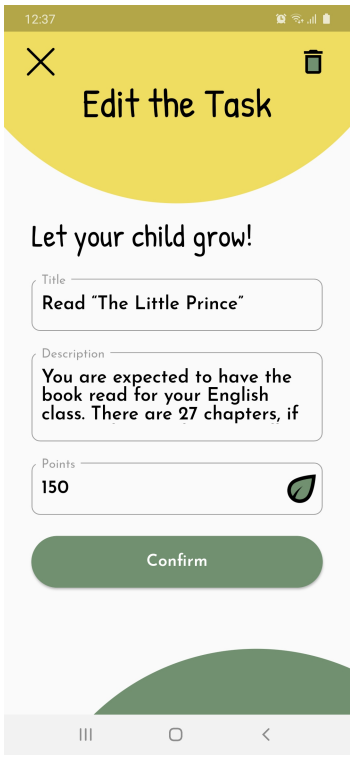
(2) Login screen



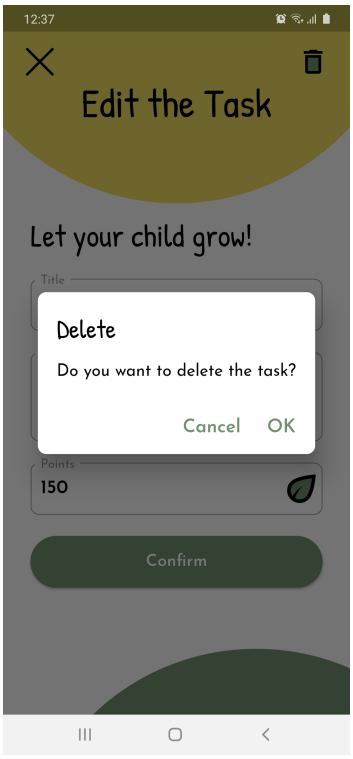
(3) Children's profiles list screen



(4) Child's tasks screen

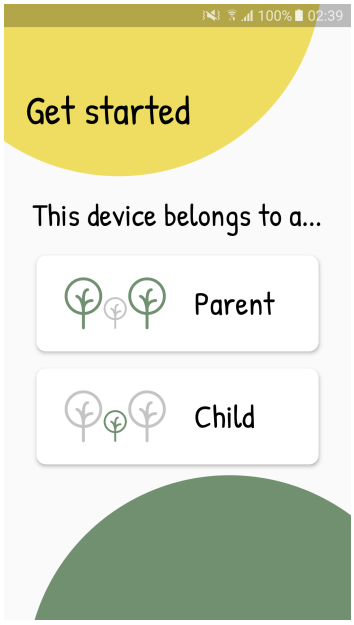


(5) Task edit screen

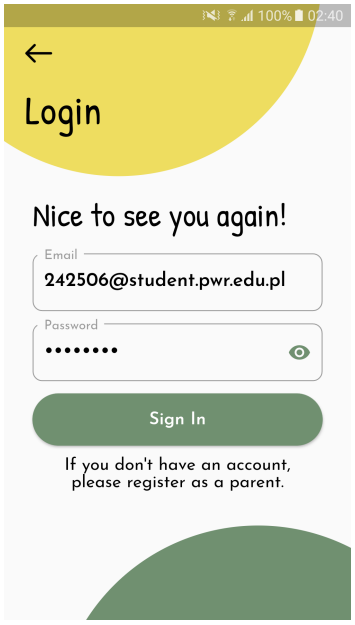


(6) Delete task confirmation

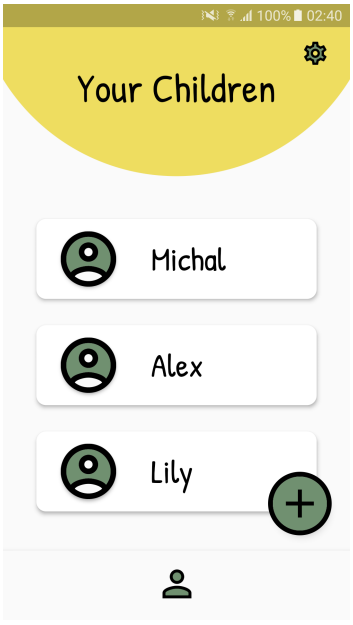
Figure 8.1: *Raise App* selected screens displayed on Samsung Galaxy A50



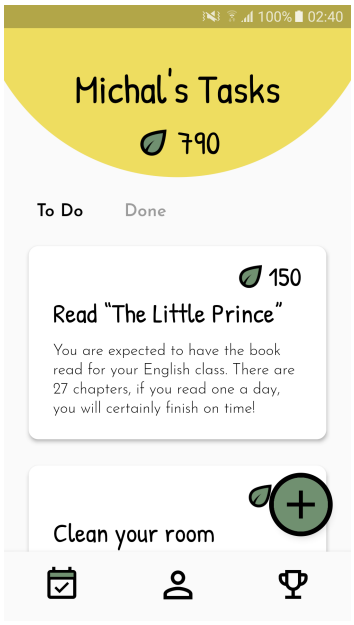
(1) Belonging screen



(2) Login screen

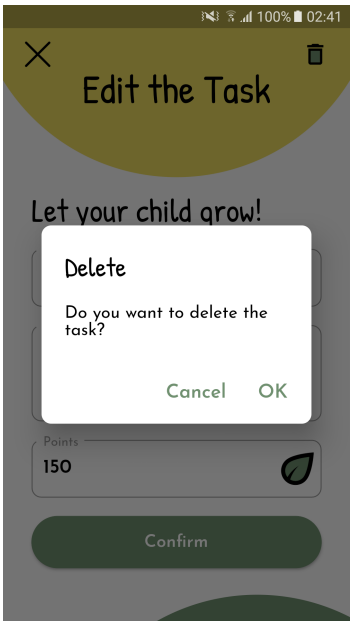


(3) Children's profiles list screen



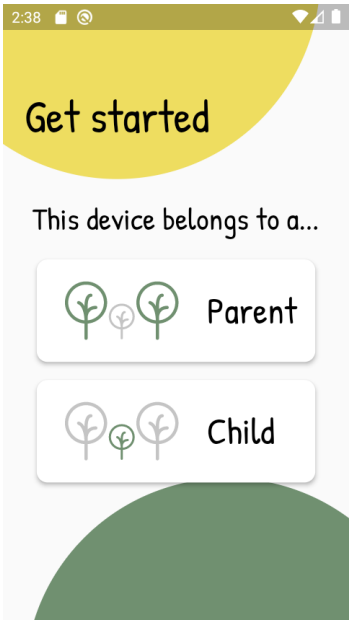
(4) Child's tasks screen

(5) Task edit screen

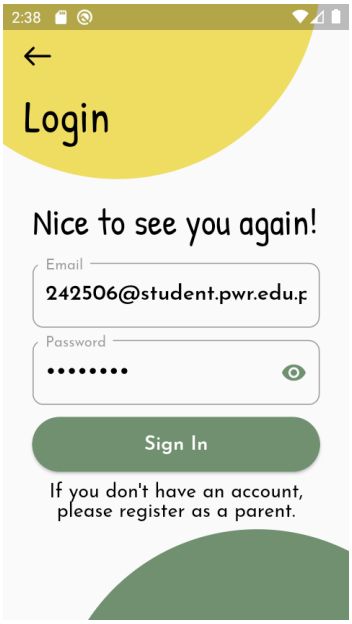


(6) Delete task confirmation

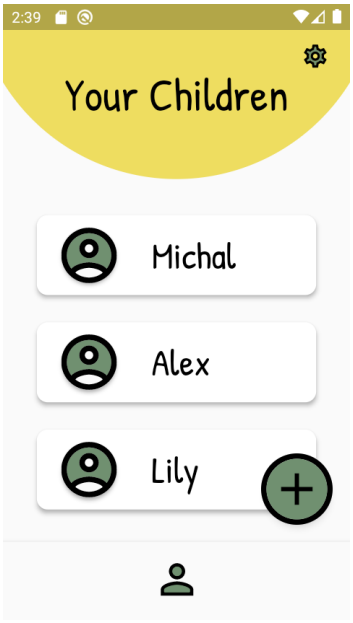
Figure 8.2: *Raise App* selected screens displayed on Samsung Galaxy S5



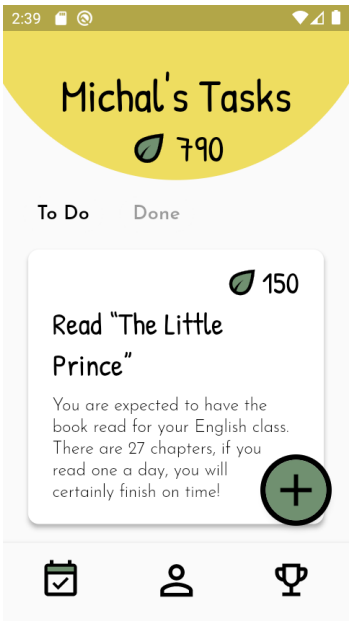
(1) Belonging screen



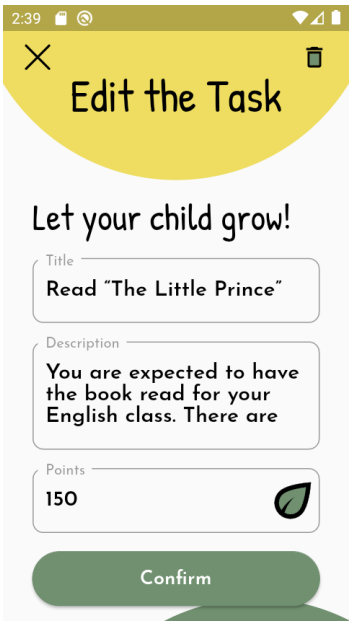
(2) Login screen



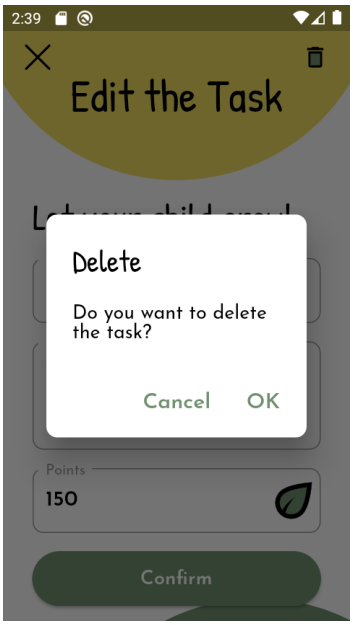
(3) Children's profiles list screen



(4) Child's tasks screen



(5) Task edit screen



(6) Delete task confirmation

Figure 8.3: *Raise App* selected screens displayed on the emulated device

8.6 User Acceptance tests

User acceptance tests is a process of verifying that a solution works for the end user. One of the most important metrics of this is *usability* and *user experience*. The former is a term precisely defined in the *ISO 9126*, "Software engineering — Product quality" and *ISO 9241*, "Ergonomics of Human System Interaction" standards [32, 41]. They distinguish four aspects of usability:

- Understandability
- Learnability
- Operability
- Attractiveness

The standards, however, do not provide implementations of usability test.

"Userbility: A Technique for the Evaluation of User Experience and Usability on Mobile Applications" is the name of a 2016 article that, based on the aforementioned standards, introduces a new approach to usability and user experience testing. It proposes using "selected questions aim to assist in capturing the experience and emotions of non-specialist evaluators about the application" [43].

Inspired by those, a questionnaire aiming to test both the usability and user experience of the created application was built and sent out to the first users of the application. The following sections will describe its structure and summarize the results.

8.6.1 Questionnaire structure

The questionnaire was prepared on the Google Forms platform and, at the time of the writing, can be found under this link[64] (also, as the audience was partly Polish, a translated version can be found here[65]).

The form consists of:

1. An introduction explaining the questionnaire's goal and structure.
2. Five sections, whose structure is based on the *userbility* technique, regarding:
    - attractiveness
    - understandability
    - learnability
    - in-app communication
    - performance
3. An overall mark.

[64] https://forms.gle/vUELFR6BDmpUxsam6 (accessed Dec. 13, 2020)
[65] https://forms.gle/gGbony1Ktq47MmwBA (accessed Dec. 13, 2020)

4. A *support* section, where users can provide feedback regardless of the asked questions.

The most important, from the evaluation point of view, are the five sections measuring usability. Each of starts with a brief explanation of the term it is concerning, followed by three questions that have a following structure:

1. What do you feel regarding the *measure* of the application?
2. What do you think could improve the *measure* of the application?
3. How satisfied are you with the *measure* of the application?

where *measure* is one of the five aforementioned usability measures. First two, not mandatory, questions are open-ended and allow for expressing the user's opinion, while the last, mandatory, question is a mark, ranging from 1 ("extremely unsatisfied") to 10 ("extremely satisfied").

8.6.2 Results

At the time of writing[66], 21 answers have been gathered. The results will be broken down into section, according to the questionnaire's structure.

Attractiveness

The arithmetic average of the users' marks for the attractiveness was **8.86**. The users highlighted the interface, its simplicity, elegance and modernity. They praised the colour and font choices. Many were attracted by the design. Some of them noted that the loading animation should be more pronounced. One user asked for a colour customisation option depending on a child.

Understandability

The arithmetic average of the users' marks for the attractiveness was **8.0**. Users were able to understand how the application can be used for the particular tasks and conditions of use. However, there were reports of a need for an onboarding process of some sort. Users expected an explanation of certain elements of the application (i.e. reward filters) and how could they benefit from them.

Learnability

The arithmetic average of the users' marks for the learnability was **8.1**. Users were generally satisfied with the learnability process. They could quickly get familiar with the application and make good use of most of its features. Because nearly half of the testers were Polish, one of the most frequent remarks was about introducing translations to the applications. One user mentioned being confused about the *add* button, which allowed for adding elements not related to the displayed screen (i.e. rewards on the tasks screen).

[66] Dec. 13, 2020

In-app Communication

The arithmetic average of the users' marks for the in-app communication was **8.71**. In general, users were very satisfied with the amount of information about what has happened or is going to happen. They praised the clarity of messages. One user highlighted the confirmation alert appearing before the app is closed, as it prevents accidental exits. Some users expressed a need for "help" tooltips.

Performance

The arithmetic average of the users' marks for the application performance was **8.8**. Users did not notice any performance issues, and if, they were usually related to their internet connection. They did not notice any delays during the application execution, or an increased battery usage.

Overall

The arithmetic average of the users' marks for overall satisfaction was **9.14**. This is the highest average of all the categories and suggest that despite sparse shortcomings, they have positive associations with the applications, which is important on the current market, where most of the decisions are made on a sub-conscious level.

8.7 Summary

Tests introduced to the project provided a much higher quality of the final product. It was important to introduce them on different levels, in order to maintain a broad perspective of the feedback loop. Unit and widget tests ensured correctness of execution of the functions, methods and widgets on the lowest level of granularity. Integration tests allowed for a more comprehensive check of the system elements' interoperability. Cross-device tests confirmed that the application will work on the expected devices, specified in the requirements. Finally, the user acceptance tests provided the target group feedback, which will set direction to the future development of the product.

Chapter 9

Work summary

9.1 Conducted work

The objective of the work was to develop a project of a mobile application with gamification mechanisms, to implement a prototype of a working product and verify it.

The work was preceded by shaping project assumptions. It helped with visualising a bigger picture of the product that was going to be created. Market analysis ensured that the existing solutions left a niche that needed to be filled. Forming project requirements established guidelines for further work. Created project and tools selection provided a foundation for a well-executed implementation phase. Finally, tests validated the product's proper operation and successful delivery.

Having said that, the project goal can be considered as achieved. Even though, there is room for improvements, the application has been completed and is already used by the targeted audience. The gamification elements in the form of tasks, points and rewards help parents to boost their children's productivity.

9.2 Further work suggestions

Despite the system being highly rated by the first group of users, it still has a room for improvement. In this project, there are three established sources of gaining knowledge about future improvements:

- implementation challenges (described in Section 7.4),
- user needs (revealed in Section 8.6),
- business needs (discussed in Chapters 2, 3 and 4),.

All of them will be briefly analysed in the following sections.

9.2.1 Implementation challenges

Even though during the development, standard conventions and guidelines have been followed, the application's code needs to go through a refactor process. While the

naming and structure on the macro level are acceptable, it is the micro level that needs additional attention.

Moreover, an alternative solution for both, the `Canvas` class used for the header composition, and the `Provider` package for the state management, should be considered. While their choice was backed by deep research and analysis, they did not fulfill the needs sufficiently.

9.2.2 User needs

Opportunities to improve in relation to user needs were discovered during the user acceptance tests. The remarks that happened the most often concerned the understandability and learnability of the application. The most important aspects regarded the onboarding process and multiple language support - these should be taken into account in the future iterations.

9.2.3 Business needs

Business needs often overlap with user needs. Sometimes, however, additional steps are expected to be taken in order to meet them. The initial projects did not any other kinds of measuring users' behaviour and satisfaction than questionnaires. Heatmaps, A/B testing, in-app reviews and gathering more data about the application itself are suggested.

9.3 Conclusions

It is important to remember that the generation of children, whose future we do shape now, will once shape our future. Utilizing technology in this process is inevitable. However, if only we use adequate tools, we can change the future, our common future, for the benefit of all.

As a closing note, I would like to quote the pioneer of the modern education of children, who also happens to be a patron of the kindergarten I used to attend, Maria Montessori:

> Children are human beings to whom respect is due, superior to us by reason of their innocence and of the greater possibilities of their future... Let us treat them with all the kindness which we would wish to help to develop in them.

Bibliography

[1] Ambler, S.W., *The Object Primer*, 3rd edition edition (Cambridge University Press, 2004).

[2] Atlassian Corporation Plc, *Jira Documentation*. https://confluence.atlassian.com/jira (accessed Dec. 08, 2020).

[3] Bittner, K., Spence, I., *Use Case Modeling* (Addison-Wesley Professional, 2002).

[4] Biørn-Hansen, A., Rieger, C., Grønli, T., Majchrzak, T.A., Ghinea, G., *An empirical investigation of performance overhead in cross-platform mobile development frameworks*. 2020.

[5] Boyatzis, C.J., Varghese, R., *Children's Emotional Associations with Colors*. 1993.

[6] Chacon, S., Straub, B., *Pro Git* (Apress, 2014).

[7] Chou, Y., *Actionable Gamification* (Packt Publishing, 2019).

[8] Christiana, Kathleen, Caitlin, Sunali, *What is the size of kids educational games and apps global market?* 2017.

[9] Christians, G., *The Origins and Future of Gamification*. 2018.

[10] Cockburn, A., *Writing Effective Use Cases* (Addison-Wesley, 2001).

[11] Cohn, M., *User Stories Applied: For Agile Software Development* (Addison-Wesley Professional, 2004).

[12] Coonradt, C., *The Game of Work*, revised, updated ed. edition edition (Gibbs Smith, 2012).

[13] Deterding, S., *Gamification: Designing for motivation*. 2012.

[14] Deterding, S., Khaled, R., Nacke, L., Dixon, D., *Gamification: Toward a definition*. 2011.

[15] Drake, M., *A Comparison of NoSQL Database Management Systems and Models*. https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models (accessed Dec. 08, 2020).

[16] Driessen, V., *A successful Git branching model*. http://nvie.com/posts/a-successful-git-branching-model/ (accessed Dec. 08, 2020).

[17] Drott, I., *Ambiguities and Limited Expressiveness in the Use Case Notation*. 2001.

[18] Feng, Y., Ye, H., Yu, Y., Yang, C., Cui, T., *Gamification artifacts and crowdsourcing participation: Examining the mediating role of intrinsic motivations*. 2018.

[19] Fowler, M., *Patterns for Managing Source Code Branches*. https://martinfowler.com/articles/branching-patterns.html (accessed Dec. 11, 2020).

[20] Fowler, M., *Presentation Model*. https://martinfowler.com/eaaDev/PresentationModel.html (accessed Dec. 11, 2020).

[21] Google, *Cloud Firestore Documentation*. https://firebase.google.com/docs/firestore (accessed Dec. 15, 2020).

[22] Google, *Dart Documentation*. https://dart.dev/guides/index (accessed Dec. 15, 2020).

[23] Google, *Firebase Authentication Documentation*. https://firebase.google.com/docs/auth (accessed Dec. 15, 2020).

[24] Google, *Flutter - Dart API Documetation*. https://api.flutter.dev (accessed Dec. 13, 2020).

[25] Google, *Flutter Documentation*. https://flutter.dev/docs (accessed Dec. 11, 2020).

[26] Google, *Material Design Documentation*. https://material.io (accessed Dec. 03, 2020).

[27] Google, *Structure Your Database - Firebase Realtime Database*. https://firebase.google.com/docs/database/android/structure-data (accessed Dec. 08, 2020).

[28] Hamari, J., Koivisto, J., Sarsa, H., *Does Gamification Work? – A Literature Review of Empirical Studies on Gamification*, in: *2014 47th Hawaii International Conference on System Sciences* (2014).

[29] Hamill, P., *Unit Test Frameworks: Tools for High-Quality Software Development* (O'Reilly Media Inc., 2004).

[30] Heiskari, J., Kauppinen, M., Runonen, M., Männistö, T., *Bridging the Gap Between Usability and Requirements Engineering* (2009).

[31] Hygen, B.W., Hamari, J., Stenseng, F., Skalicka, V., Kvande, M.N., Zahl-Thanem, T., Wichstrøm, L., *Time Spent Gaming and Social Competence in Children: Reciprocal Effects Across Childhood*. 2020, vol. 91, 3, p. 861–875.

[32] Idri, A., Moumane, K., Abran, A., *On the Use of Software Quality Standard ISO/IEC9126 in Mobile Environments* (2013).

[33] IEEE-SA Standards Board, *IEEE Recommended Practice for Software Requirements Specifications* (1993).

[34] Interaction Design Foundation, *What is Responsive Design?* https://www.interaction-design.org/literature/topics/responsive-design (accessed Dec. 05, 2020).

[35] Kaur, K., Rani, R., *Modeling and querying data in NoSQL databases*, in: *2013 IEEE International Conference on Big Data* (2013).

[36] King, D.L., Potenza, M.N., *Not Playing Around: Gaming Disorder in the International Classification of Diseases (ICD-11)*. 2019, vol. 64, 1, p. 5–7.

[37] Kondracka-Szala, M., *Games for developing selected competences of key children – reviewing literature of the subject*. 2017. (Polish).

[38] Kök Eren, H., Örsal, , *Computer Game Addiction and Loneliness in Children*. 2018, vol. 47, 10, p. 1504–1510.

[39] Lightsey, B., *Systems Engineering Fundamentals*. 2001.

[40] Morschheuser, B., Hamari, J., Koivisto, J., *Gamification in Crowdsourcing: A Review*, in: *2016 49th Hawaii International Conference on System Sciences (HICSS)* (2016), p. 4375–4384.

[41] Moumane, K., Idri, A., Abran, A., *Usability evaluation of mobile applications using ISO 9241 and ISO 25062 standards*. 2016, vol. 5, 1, p. 548.

[42] Nand, K., Baghaei, N., Casey, J., Barmada, B., Mehdipour, F., Liang, H., *Engaging children with educational content via Gamification*. 2019, vol. 6, 1, p. 6.

[43] Nascimento, I., Silva, W., Gadelha, B., Conte, T., *Userbility: A Technique for the Evaluation of User Experience and Usability on Mobile Applications*, vol. 9731 (2016).

[44] Norman, D., *The Design of Everyday Things: Revised and Expanded Edition* (Basic Books, 2013).

[45] Object Management Group, *The Unified Modeling Language*. https://www.uml-diagrams.org/ (accessed Nov. 29, 2020).

[46] Olivera, H.V., Holanda, M., Guimarâes, V., Hondo, F., Boaventura, W., *Data Modeling for NoSQL Document-Oriented Databases* (2015).

[47] Pereira, P., Duarte, E., Rebelo, F., Noriega, P., *A Review of Gamification for Health-Related Contexts*, in: *Design, User Experience, and Usability. User Experience Design for Diverse Interaction Platforms and Environments*, edited by A. Marcus, Lecture Notes in Computer Science (Springer International Publishing, 2014), p. 742–753.

[48] Rahman, A., *A Comparative Study of Hybrid Mobile Application Development*. https://easychair.org/publications/preprint/KLPH (accessed Dec. 06, 2020).

[49] Roberts, M., *Serverless Architectures*. https://martinfowler.com/articles/serverless.html (accessed Dec. 06, 2020).

[50] Sadalage, P.J., Fowler, M., *NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence* (Addison-Wesley, 2012).

[51] Shah, K., Sinha, H., Mishra, P., *Analysis of Cross-Platform Mobile App Development Tools*, in: *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)* (2019).

[52] Smith, J., *Patterns - WPF Apps With The Model-View-ViewModel Design Pattern*. https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern (accessed Dec. 11, 2020).

[53] Software Freedom Conservancy, *Git Reference*. https://git-scm.com/docs (accessed Dec. 15, 2020).

[54] Vaala, S., Ly, A., Levine, M.H., *Getting a Read on the App Stores: A Market Scan and Analysis of Children's Literacy Apps. Full Report*. 2015.

[55] Vassallo, C., Panichella, S., Palomba, F., Proksch, S., Zaidman, A.E., Gall, H.C., *How Developers Engage with Static Analysis Tools in Different Contexts*. 2020, vol. 25, 2.

[56] Visual Paradigm International Ltd., *Conceptual, Logical and Physical Data Model*. https://www.visual-paradigm.com/support/documents/vpuserguide/3563/3564/85378_conceptual,l.html (accessed Dec. 08, 2020).

[57] Whelan, B.M., *Color Harmony, 2: A Guide to Creative Color Combinations* (Rockport Publishers, 1994).

[58] Wijman, T., *The World's 2.7 Billion Gamers Will Spend $159.3 Billion on Games in 2020; The Market Will Surpass $200 Billion by 2023*. 2020. https://newzoo.com/insights/articles/newzoo-games-market-numbers-revenues-and-audience-2020-2023/ (accessed Nov. 29, 2020).

[59] World Health Organization, *Addictive behaviours: Gaming disorder*. 2018. https://www.who.int/news-room/q-a-detail/addictive-behaviours-gaming-disorder (accessed Nov. 26, 2020).