UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Creating a cross-platform application for children with cancer

Author:
Viacheslav BERNADZIKOVSKYI

Supervisor:
Serhii MISKIV

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

I, Viacheslav BERNADZIKOVSKYI, declare that this thesis titled, "Creating a cross-platform application for children with cancer" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"Heart is what separates the good from the great"

Michael Jordan

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Creating a cross-platform application for children with cancer

by Viacheslav BERNADZIKOVSKYI

Abstract

An objective of the project is to create volunteer charity platform while investigating all of the stages of mobile and web software development. The expectation is that both children and volunteers will find theirs benefits on the platform. Main project stages include creating a design system, development of a RESTful API and development of website, Android and iOS applications. Back end part is written with FastAPI on Python, while front end one is written with Flutter on Dart. The implementation code can be accessed until September 2022 here:
Server
UI

Acknowledgements

First of all, I want to thank my team, those people were with me through the whole course of the project: Mariia Moskalyk, Karyna Volokhatiuk, Inna Alanbousi and Ievgeniia Iurevych. Secondly, I am obliged to the university and everyone, who has contributed into Computer Sciences program for all the knowledge I gathered during my 4-year studies at Ukrainian Catholic University. I am grateful to all the teachers of the university, in particular my supervisor Serhii Miskiv, for his time and guidance in the project…

Contents

List of Figures

List of Tables

List of Abbreviations

API	**A**pp **P**rogram **I**nterface
BLoC	**B**usiness **Lo**gic **C**omponent
CRUD	**C**reate **R**ead **U**pdate **D**elete
JIT	**J**ust-**I**n-**T**ime
DB	**D**ata**B**ase
JSON	**J**avaScript **O**bject **N**otation
JWT	**J**SON **W**eb **T**oken
MVP	**M**inimum **V**iable **P**roduct
NCRU	**N**ational **C**ancer **R**egistry of **U**kraine
ORM	**O**bject **R**elational **M**apping
REST	**RE**presentational **S**tate **T**ransfer
SQL	**S**tructured **Q**uery **L**anguage
UI	**U**ser **I**nterface
URL	**U**niform **R**esource **L**ocator
UX	**U**ser **EX**perience

Dedicated to my friends and family…

Chapter 1

Introduction

1.1 Problem

It is not a secret for everyone that cancer remains a serious problem nowadays even though understanding of the illness and its treatment has been significantly improved over the last 50 years. Although, the progress is quite considerable, the treatment still remains accessible for the minorities. In 2019, **NCRU** registered 137968 cases of malignant tumours in Ukraine, and while the disease is likely to hit elder people, still 958 cases were detected in children aged below 19 years.

1.2 Idea

An importance of the problem caused establishing multiple cancer organisations and charities. The most famous one, which comes from Ukraine, is *Tabletochki* charity foundation. Their main goal is to prevent new cases of mortality and to once make sure, that no Ukrainian child dies of cancer. There are 4 main directions that the organisation is currently working in:

- Delivering support to children with cancer and their families
- Systematic cooperation with children's oncology departments.
- Protection of the rights of children with cancer.
- Professional development and growth of medical staff.

Besides fundraising costs for treatment, *Tabletochki* also engage volunteers for organising a massive amount of educational and entertaining events for the children of the foundation. They can offer different activities like:

- Field trips in different cities and countries.
- Meetings with celebrities and influencers.
- Celebrating holidays together.
- Bunch of various educational events online.

That is where the idea for the project comes up. The plan is to create a platform, which is reachable for everyone no matter whether they use a laptop or mobile phone, iOS or Android. There, different volunteer occasions and actions could be published and highlighted, so that no one could miss them and everyone interested could join.

1.3 Implementation Steps

1.3.1 Devising Requirements

On that stage every single desired functionality of the project vision should be described. All the requirements should be covered by carefully thought-out features, part of which would be included into a minimum viable product.

1.3.2 Estimating Work

After concluding the requirements, work should be estimated for few reasons:

- Team should know where to prepare for release.
- Estimation is reported to sponsors and partners.
- Deadlines are followed and met easier when work is estimated.

1.3.3 Design System Elaboration

Independent of writing code stage where **UX** should be scrupulously constructed and **UI** should be correspondingly painted.

1.3.4 Database Establishing

The stage includes selecting a proper **DB** should be selected, configured and prepared for work.

1.3.5 Backend API Development

Here is developed all the logic, that allows a database and an application to communicate with one another. Mostly, here is maintained a part, that a user doesn't see.

1.3.6 Frontend Interface Development

On this stage, web and mobile applications for Android and iOS should be covered. Since it is an opposite to the previous step, user sees and also interacts with everything, which is developed here.

1.3.7 Testing

Besides writing tests for the code, application should also be checked by people. During a testing sessions errors could be spotted and documented as bugs. Fixing all the defects is also covered in the step.

1.3.8 Deployment

When everything is ready and tested, the application is ready for it's first release. Website should be hosted and mobile apps should be pushed to *Google Play Market* and *App Store*.

1.3.9 Further Development

After the first release the development is not stopped. Basically, almost all the steps, described earlier are combined into a cycle, except **1.3.4 Database Establishing**. While database demands only running new migrations, the requirements for a new version should be established, which adds a new part of work in every step of the cycle. The development continues until all the requirements are met.

Chapter 2

Requirements

Application should support three types of entities:

- User.
- Event.
- Post.

2.1 Events

An entity, which is displayed in a calendar. Should be named, have a description (where location and format details would be specified), and appointed for a date and exact time. Optional: events could have tags for location or format, since options for that are given.

2.2 Posts

Sometimes, volunteers or child-care specialists can create posts with announcements or results from the events. The post should be named and have a text, it's creation date should be displayed for the users. There should be an option to comment a post, since users may have questions about events. Optional: posts could have various tags.

2.3 Roles

Final application should have three types of users:

- Regular User (Child or Parent).
- Volunteer.
- Child-care specialist.

2.3.1 Regular User

A user with no additional permissions. Should be able to:

- Register and log in to the account.
- See events in a calendar and apply to them, and cancel applications.
- Search for events, with filters.
- Ask for help from a child-care.

2.3.2 Volunteer

Middle-privileged user. In addition to a regular user abilities also should be able to:

- Create an event, edit and delete own events.
- See applications for own events. Accept or deny them.
- Create a post if child-care specialist approves, edit and delete own posts.

2.3.3 Child-care specialist

The most privileged user on the platform. Basically, is an administrator and has a full access to the database. In addition to abilities, mentioned earlier, should also be able to:

- Do **CRUD** operations on users. However, no **WRITE** permission on other child-care specialist.
- Do **CRUD** operations on events.
- Do **CRUD** operations on posts.
- Do **CRUD** operations on comments.
- Do **CRUD** operations on posts.
- Apply or deny volunteer actions, that need an approval.

2.3.4 Estimation

These requirements are approved with the team on March 1st 2022, and non-optional part is estimated to be finished by June 20th 2022.

Chapter 3

Design

3.1 Tool

Consistent design systems should be responsive, scalable and provide meaningful and relevant experiences to the users. All of those qualities could be reached with a design tool like, *Sketch*, *Adobe XD*, and so on. There is a huge variety of design tools, so which one suits the case in a best way? The answer is *Figma*...

All of the mentioned tools have multiple features in common, however there is something that makes *Figma* a more pleasant choice:

- Works directly in a browser. That option allows a quick setup and convenient use.
- Real-time collaboration. The tool was designed with a collaboration in mind, which makes it a good fit, because the design team consists of two people.
- Considered to be a beginner-friendly, which is preferable option, because the author, who is a part of the design team, has no knowledge of **UI**/**UX**.
- *Figma* offers more flexibility when it comes to a vector manipulation. That is a result of using vector networks, which allow a user to connect multiple lines to a single point.
- Prototyping of the application.

Besides all the advantages listed above, *Figma* has a free subscription. However, there is also a disadvantage of the tool, since it comes with a high productivity requirement. If the network connection is poor or computer characteristics are low, *Figma* might be annoying to work with.

3.2 Brand Book

Since *Tabletochki* foundation takes part of a main partner the project's brand book also looks similar.

3.2.1 Colors

There are three primary colors:

- Light blue (hex: 67C6D7)
- Dark blue (hex: 046493)
- Red (hex: FF3600)

3.2.2 Rounded Corners

- Regular border radius is 20 logical pixels.
- Doubled border radius is 40 logical pixels.

Regular size rounded corners is used almost everywhere.

3.2.3 Logo

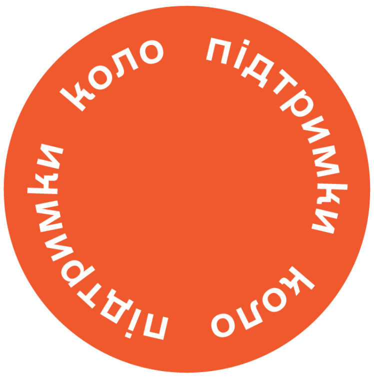

FIGURE 3.1: Care 4 Child Logo

3.3 Mock-up

For analyzing user and improving user experience was agreed to design a mock-up firstly. After few testing sessions with the target audience and other volunteers, that is how the first mock-up for the mobile application looked:

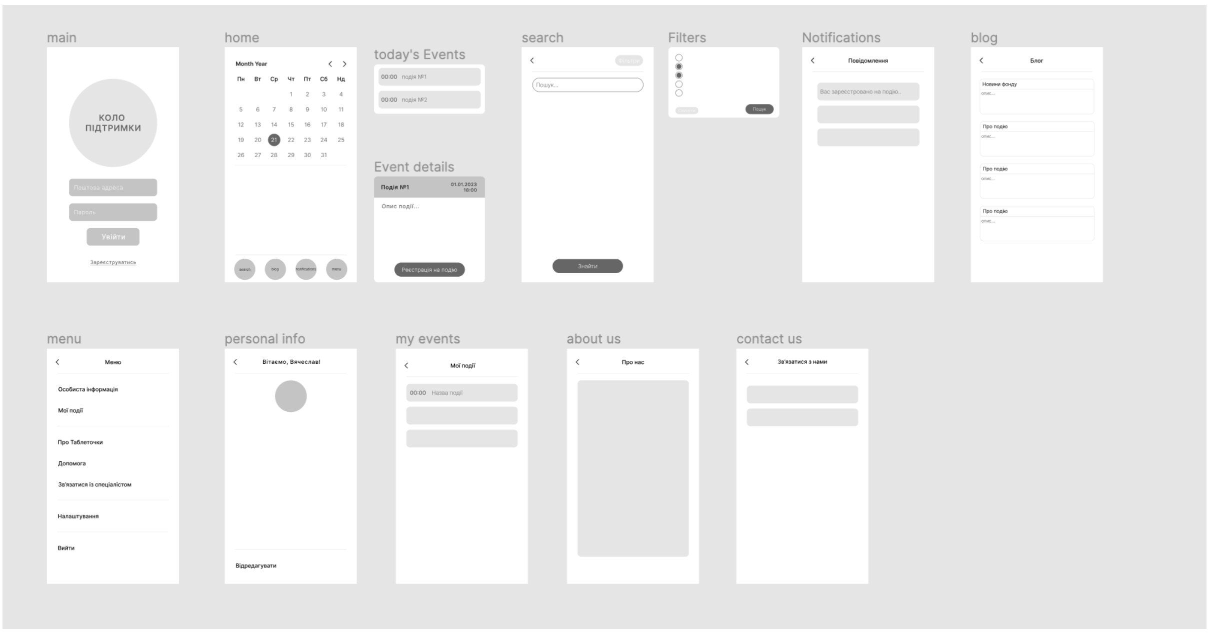

FIGURE 3.2: Mobile Design Mock-up

The goal is to provide maximal amount of the information while keeping a distraction on its minimum. Final version was considered as the most user-friendly.

3.4 Final Appearance

The final step of the stage was to paint mock-ups, using contents from the brand book:

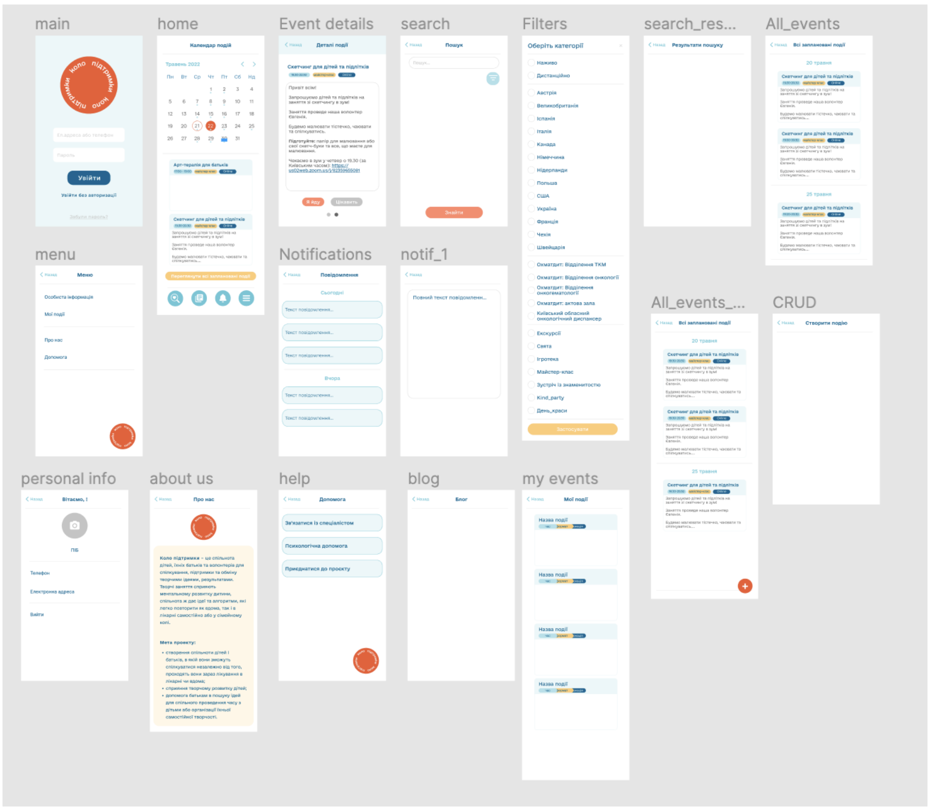

FIGURE 3.3: Final Mobile Design

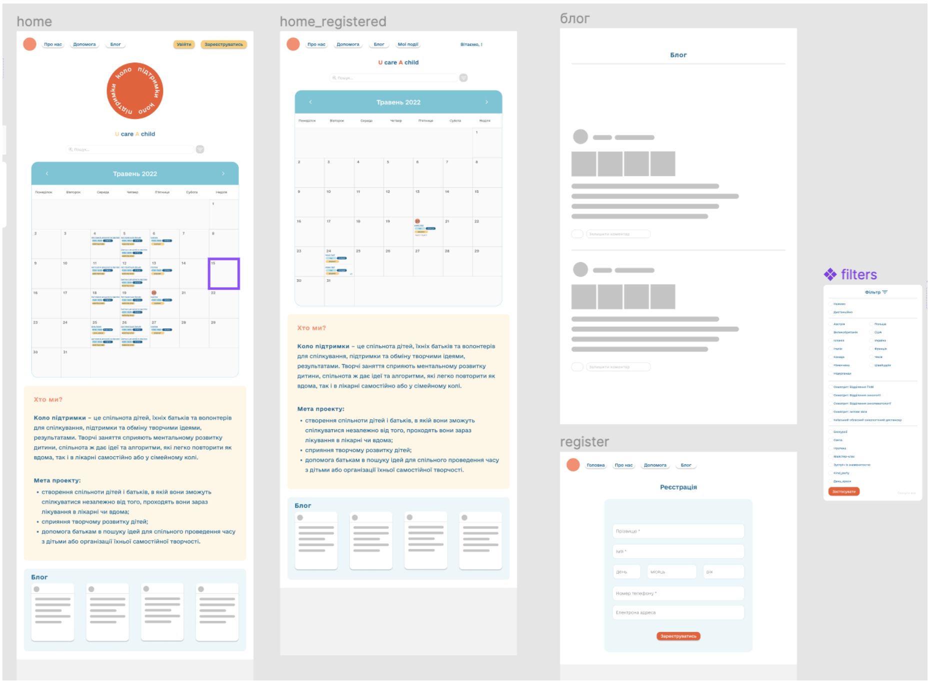

FIGURE 3.4: Final Website Design

Chapter 4

Database

4.1 Choosing a Database

4.1.1 SQL vs NoSQL

When it comes to choosing a database, first step of the stage is to decide whether it should be **SQL** or not. Besides using different query languages, there is a huge amount of differences between those two:

- **Relativity**. Relational databases save data in tables, while non-relational - in documents, which is something more of a list.
- **Schema**. **SQL** databases use predefined schema for tables, while **NoSQL** use dynamic - for documents. In the first case is much more easier to parse, which becomes especially sensitive when a database grows larger. Second case allows to store documents with different fields inside a single collection.
- **Scalability**. Horizontal scalability for first and vertical - for second. **SQL** databases productivity can only be improved by boosting physical characteristics (*CPU*, *RAM*). NoSQL can also use sharding, which is a type of database partitioning, what in turn allows dividing large slow databases into a smaller faster and better-performing ones.

After comparing two types of databases with characteristics above the final choice is to use **SQL DB**. Potential data analysis session opened few reasons for that. Firstly, the data can be separated into collections with fixed fields. Second reason is that database entities require tight connection between each other which is easily achieved with tables.

4.1.2 Which SQL DB to use?

Within the scope of the project three relational databases were analyzed:

- *SQLite*.
- *MySQL*.
- *PostgreSQL*.

SQLite. It is a library, which provides a small, high-reliability and fast **SQL DB** engine, written in C. The authors claim it to be the most used database engine in the world, since it is built into most mobile phones and computers, meaning it is used a lot on a daily basis. However, it is serverless, meaning no access to the network provided, what in turn means users can't share mutual data. The size of the database also doesn't support scaling large amount of a data. Conclusion for *SQLite* is that it could be used for testing or data caching in some future releases.

The other two databases, being *MySQL* and *PostgreSQL* are said to be top 2 free open-source databases, which serve for a long period of time and are widely involved for using in a huge variety of commercial, open-source or corporative mobile/web applications. Those two have much in common, however also there is a bunch of features that are different for them.

TABLE 4.1: Fivetran. PostgreSQL vs MySQL

	PostgreSQL	MySQL
Architecture	Object-relational	Relational
Data Types	Numeric, date/time, character, boolean, enumerated, geometric, network address, JSON, XML, HSTORE, arrays, ranges, composite	Numeric, date/time, character, spatial, JSON
Performance	Suitable for applications with high volume of both reads and writes	Suitable for applications with high volume of reads
Security	Access control, multiple encrypted connection	Access control, encrypted connections
Support	Community support. Companies that have their own release of PostgreSQL may offer support around it	Community support, plus vendor-provided support contracts

From the table above one can make a conclusion that *PosgreSQL* seems to be a better choice, since it offers a little bit more than *MySQL*, even though also requires more effort in return...

4.2 Establishing a Database

Since the choice is to work with a relational database, second step, being establishing the database, is about designing tables and connections between them. After analyzing potential data one more time, it is divided into such collections:

- **Users**.
- **Events**.
- **Applications** to events.
- **Posts**.
- **Comments**.
- **Notifications**.

Connections between the tables are called associations or relationships. There are three types of them:

- **One-to-One**. The easiest type of an association, record of first table can be connected to a single record of second one.
- **One-to-Many**. A record of first table can be connected to many records of second one, while second table's instances are still related to a single first table's record.
- **Many-to-Many**. A record of first table can be connected to many records and vice versa. Is achieved by creating an intermediate table, which has a Many-to-One association with both tables.

Most used type of connection in the project is a One-to-Many relationship. Many-to-Many is established once, and no usage of One-to-One.

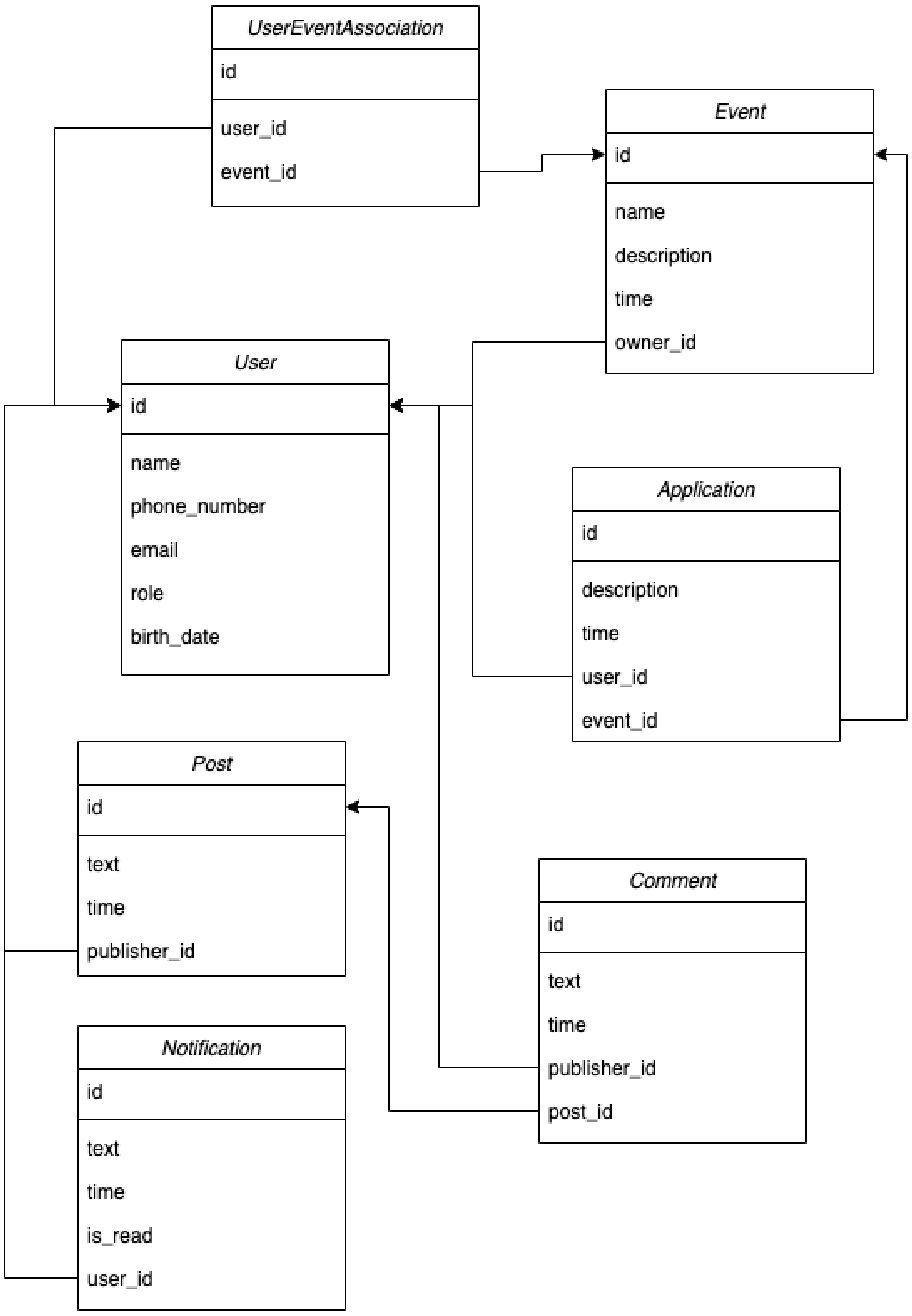

FIGURE 4.1: Database Structure

The structure of the database is presented on the figure above. Besides, all of the entities mentioned earlier, it also contains a table *UserEventAssociation* which establishes a Many-to-Many connection between *User* and *Event* tables.

Chapter 5

Backend Development

5.1 REST APIs

Main goal of that stage is a development of a service, that would allow to transfer the data between the database and platform. Such services are called application program interfaces (**APIs**). They introduce a standardized way for its clients to receive and send data of different types. There is a massive variety of approaches to create the interface, however most popular and effective one is a **REST API**. Key reasons to make an API to be **REST** are flexibility, meaning using different types of data and requests, and scalability - an ability to increase a number of requests easily and fast. Also, such interfaces are hosted, which makes them easy to use, because every piece of functionality can be reached by providing a **URL**. All together, that makes **REST API** a good choice for developing web and mobile applications.

5.2 FastAPI

Team's choice of a development tool for the **REST API** fell on *FastAPI*. It is a modern, high-performance web-framework for building app program interfaces with Python. Main advantages of the framework are:

- **Performance**. One of the fastest Python frameworks, also allows asynchronous programming, since can be run with *Unicorn*
- **Code Writing**. Framework makes it fast, short and easy.
- **Documentation**. *FastAPI* automatically generates a *Swagger* docs for every request that a user writes.

The framework also can be used combined with a lot of different Python packages, for example, using *Pydantic* allows declaring a request's body using standard Python types. Various famous international companies use *FastAPI* for own services, among them *Microsoft*, *Uber*, *Netflix* and so on.

5.3 Connection with Database

5.3.1 Database toolkit

The most popular **SQL** database toolkit for Python is *SQLAlchemy*. It allows to connect to the database, make all available types of requests and retrieve subsets of information with object relational mappers. *SQLAlchemy* considers the database to be a relational algebra engine, not just a collection of tables. Rows can be selected from not only tables but also joins and other select statements; any of these units can be composed into a larger structure. *SQLAlchemy*'s expression language builds on this concept from its core. Tool description basically divides it into two parts:

- **Core**. All the functionality for reading, creating, modifying and deleting data is stored here. **DB** setup also belongs to the part, creating an engine, establishing a connection and so on.
- **ORM**. Object Relational Mapping allows retrieving the data from each table as a collection of Python classes. Moreover, it also simplifies a procedure of creating, updating and deleting table instances making it close to modifying a regular Python list.

Such division allows users to use the parts separately and independently, for example, getting rid of the mapper, since it is an additional layer of functionality and vice versa, using it without core features, which is a popular approach according to *SQLAlchemy* team's reports.

5.3.2 Migration tool

SQL Database Migration is a process of changing a schema of any table and then adjusting the data to the new format. That operation can be done manually, however it becomes bulky and it is easy to make a mistake, especially when database grows bigger and bigger. The solution is to use a migration tool. There is a package, which is written as a migration tool for using together with *SQLAlchemy*, it is called *Alembic*. It provides a change management system, where every version of the schema is called revision. The procedure is relatively easy and requires few steps:

- Creating a revision.
- Writing a Python function, which is triggered when a user upgrades to current revision.
- Writing a Python function, which is triggered when a user downgrades current revision. This and previous functions use *SQLAlchemy* package inside.

Every revision has an id, and the previous revision's id. So after the procedure, user can safely move in both directions along the chain.

5.4 Access

5.4.1 Authorization

Authorization to the API is quite easy, user should provide something unique for identification (phone number for current case) and a password. During a registration process user is created inside of the database and also the password is specified. However, to add an additional layer of security the password record is hashed with *PBKDF2* derivation function, which at the moment has no known security issues.

5.4.2 Authentication

Since the **API** contains data, that is personal for every user, there is a necessity in adding an authentication process, so that data could be read and written securely. Most common way of authentication to the interfaces, which are used for web/mobile development is **API** or access token. Access token is something of a key to the interface, which is sent to user, after password authorization is successfully completed.

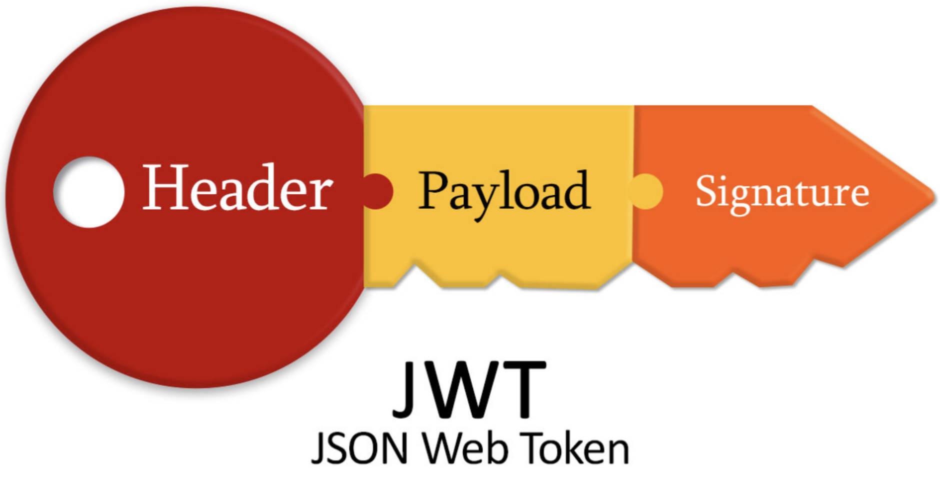

FIGURE 5.1: JWT Structure

JWT is a popular token format, which consists of three parts:

- **Header**. Basically containing metadata. Token format, creation algorithms and so on.
- **Payload**. Contains an information, which is needed for the app programming interface. User identification, permissions, token expiration time.
- **Signature**. Part, that verifies the user. Header and payload are taken together with provided secret and signed with an encryption algorithm.

Those three parts base-64 encoded and put into a single string, divided by dots. Token cryptographic algorithm, used in the project is *HS256* and secret for it is saved as an environment variable of the server.

5.5 Result

In result, current version of the **API** for **MVP**, mainly contains Create, Read, Update, Delete operations for the database tables. However, it also has requests with more complicated logic, for example, accepting/denying an application for an event should also cause a new notification for the applicator, and established procedure of authorization/authentication.

Chapter 6

Frontend Development

6.1 Flutter

Since the task is to develop applications for web, iOS and Android all at the same time, a choice of the tool fell on *Flutter*. It is an open-source **UI** software development kit, introduced by Google in May 2017. This tool is a declarative framework, which is written in Dart (The language is also introduced by Google).*Flutter* is used to create mobile, web and desktop applications, all from the same codebase. Besides being multi-platform it also has a lot of different advantages presented during the 5 years of production. These are the frameworks strengths:

- **Writing a code**. Writing a **UI** with *Flutter* becomes easier and faster over the time as community grows. New packages and elements are added on a weekly basis.
- **Perfomance**. The framework offers close to native development perfomance, with using ahead-of-time compilation on release mode.
- **Debug mode**. While an application is in a stage of development, programmers can use a debug-mode where the project is compilated at a run-time (**JIT** compilation). Since it is a declarative framework, **UI** is rendered as a tree of widgets, what allows reloads after changes in a very short time. The tool provides *hot-reload* and *hot-restart* options, where application is refreshed with current state or restarted completely.
- **Complex, animated UI**. From the beginning *Flutter* was a tool, which people used for own small projects or playgrounds. Now, however, users can create complex and compound **UI**, which is as flexible as the one, written with native approaches. One more huge plus of the framework is a decent support of a variety of animation types, which make an interface feel more intuitive, beautiful, and improve user experience.
- **Widget Catalog**. *Everything in Flutter is a widget*. If interface is puzzle, then every piece of it is a widget. *Flutter* provides a *material* package which is full of predefined widgets like:
  - Buttons, bars and dialogs.
  - Texts, and inputs.
  - Rows and Columns.
  - Tables, grids and cards.
  - Bunch of different scroll views.

6.2 State Management

6.2.1 Stateful Widget

The framework provides an interface with multiple approaches for creating custom widgets. One of them are Stateless and Stateful widgets. First type can have own properties, however they are immutable. Stateful widgets, as opposite, can change their state, by rebuilding all the descendants.

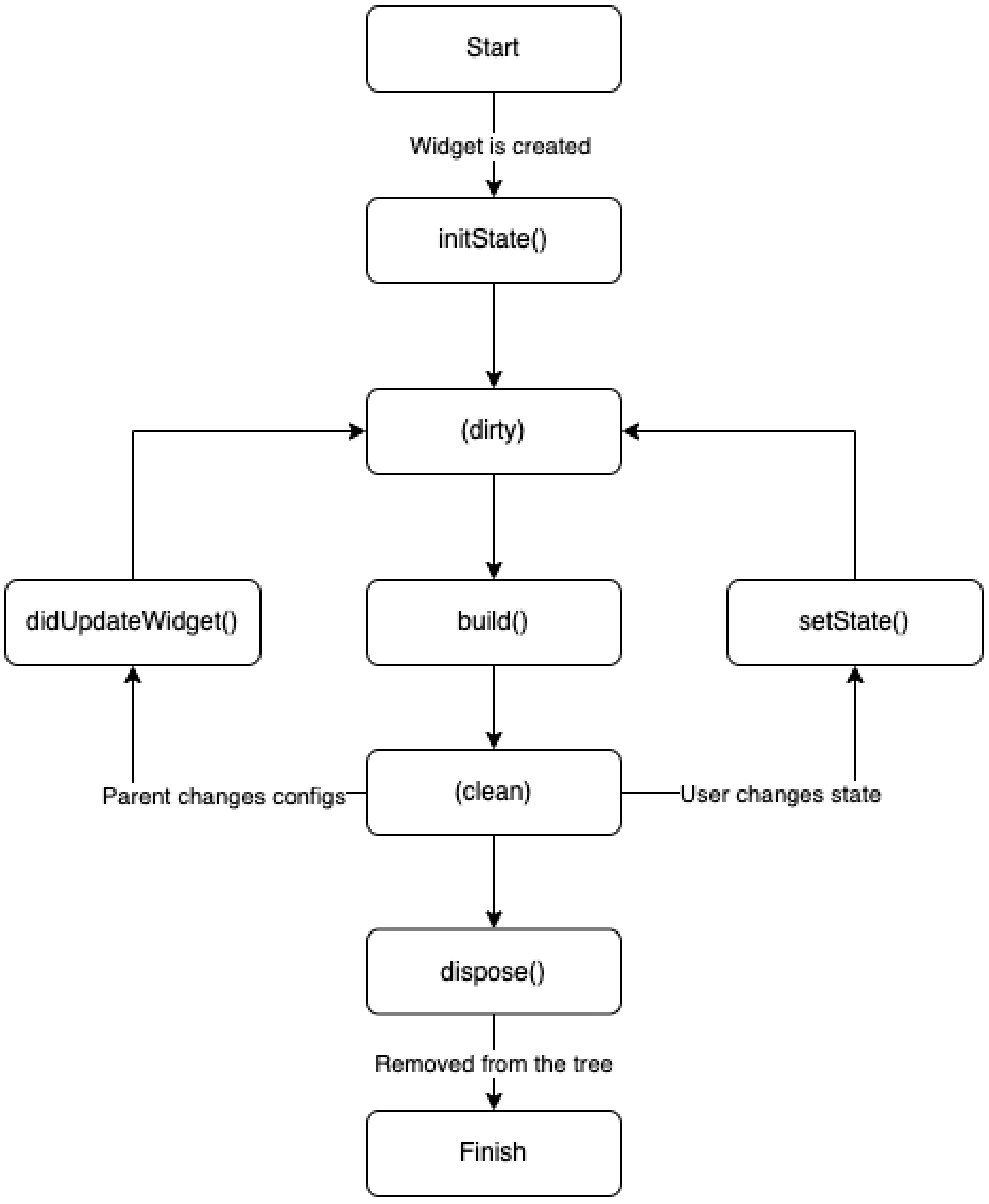

FIGURE 6.1: Simplified stateful widget lifecycle

Such state-managing approach is good for creating small independent widgets, which should function separately and have personal state. When an application and states grow large, that approach is uncomfortable. Stateful widgets are rarely used for the platform.

6.2.2 BLoC

Native tools for developing UI software widely use an imperative paradigm of programming, meaning describing how the program should behave by explicitly ordering every single step. Flutter developers specify what should program without delivering a control flow. It is a relatively new approach for UI software development and that resulted in emerging of software pattern named **BLoC**. **BLoC** stands for **B**usiness **Lo**gic **C**omponent and it allows to separate a busineess logic from the user interface completely. The pattern consists of such parts:

- **UI**. Ordered collection of widgets with interaction.
- **BLoC**. Bussiness Logic Component, which manages a state accordingly to different events.
- **Event**.
- **State**.
- **Action** (implicit).

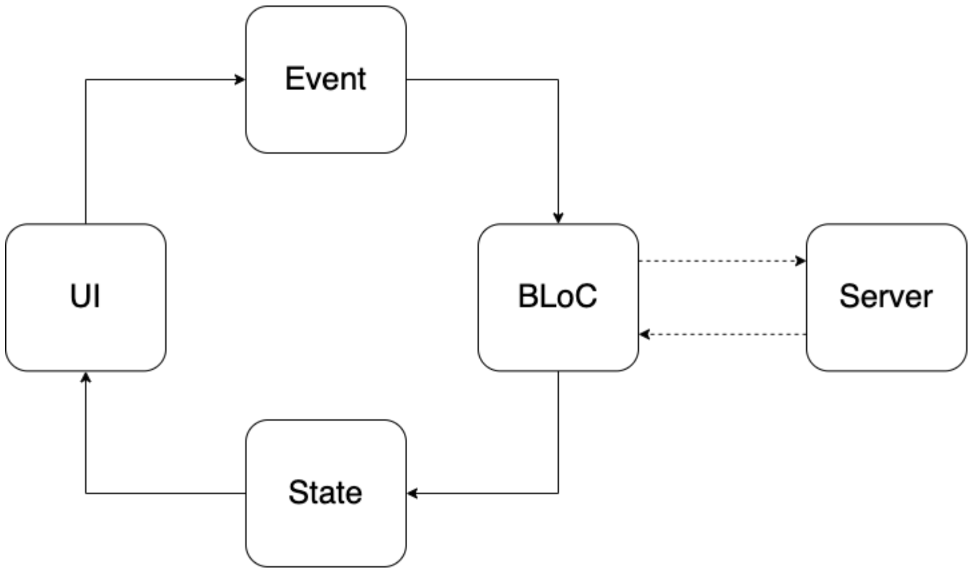

FIGURE 6.2: BLoC Pattern Schema

For better understanding of the logic, here are two quick step-by-step examples of using the pattern:

- Offline. Let the task is to create a button with number below and every time user taps the button - value increases by 1.
  1. First of all bloc generates an initial state with starting value of 0. **UI** reads it and displays a button and 0 below.
  2. User taps the button.
  3. **UI** generates an event, that button is tapped, and passes it to the **BLoC**.
  4. Bloc processes the event, and increases the value by 1. Then passes a new updated state to **UI**.
  5. **UI** reads it and displays a button and 1 below.

- Online. Let the task is to authorize user..
    1. User interacts with **UI**, enters credentials taps a sign-in-button.
    2. **UI** creates a sign in event with credentials and sends it to the **BLoC**.
    3. **BLoC** validates the credentials, if valid - request to get a token is sent to the server, else error-state is created and sent to **UI**, which displays the error, user moves backwards to step 1.
    4. Server sends response back to **BLoC**.
    5. If credentials are correct, state with success is created and user is moved to next screen, otherwise error-state is created and sent to **UI**, which displays the error, user moves backwards to step 1.

Using the pattern allows to completely separate a bussiness logic from the user interface, which results in a clean separated structure event when the code base is large, that is the reason why state in the project is mostly managed by **BLoC**.

6.2.3 Hooks

Flutter Hooks are a clean way to improve a management of a widget life cycle, while promoting reusing of a code and decreasing level of duplication. *Flutter* provides a huge amount of widget, which have any kind of a controller, which should be properly initialized, disposed and managed during the widget time being in a widget-tree. Using those, not only forces developer to have *StatefulWidget* a lot, but increases code duplication. Hooks allow to manage those controllers automatically with calling the only function, which returns an instance of the desired controller. Since, using of **BLoC** makes all of custom widgets to be a *StatelessWidget*, *Flutter Hooks* are a nice addition to the pattern.

6.3 Adaptive and Responsive Layout

Initial idea is to use single code base for web and mobile application, but even if layouting for web and smartphones is separated, each of them still has multiple versions of screen sizes. *Figure 6.3* illustrates the difference in rendering of a square sized 300px/300px on two Iphones, while second one could easily contain 2 such squares, aligned vertically, first IPhone would need a scroll for that:

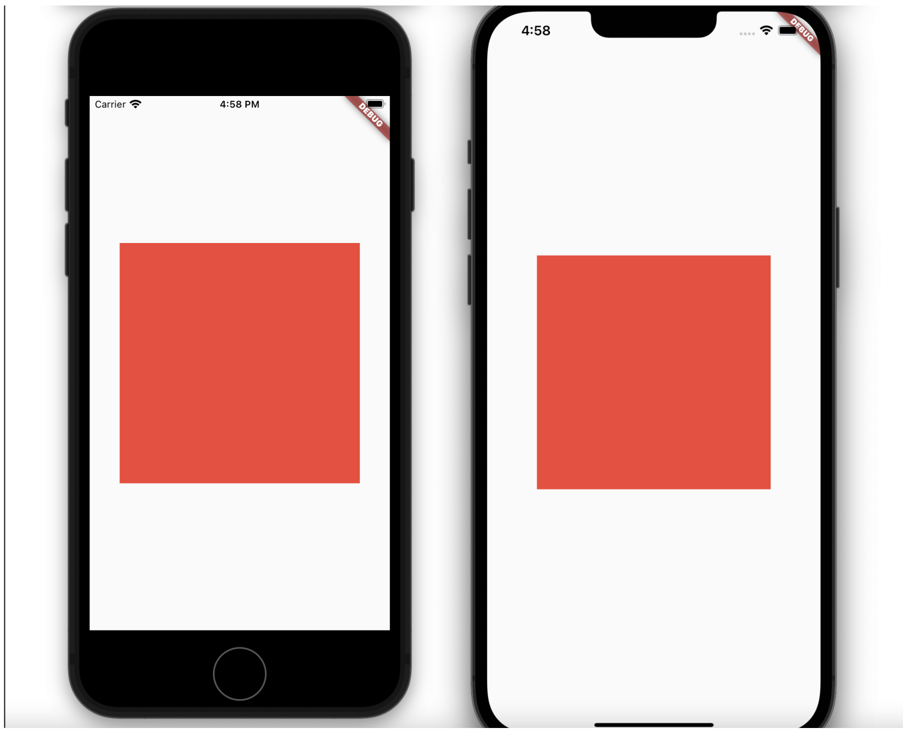

FIGURE 6.3: IPhone 7 and IPhone 13 Pro Max Screen Sizes

Such size mismatch can lead to different overlay errors and while one smartphone shows a perfect layout, other one can break it completely. But how can a single code satisfy all of available screen sizes? Well, that is a simple mathematics. Firstly, let's divide screen into a grid:

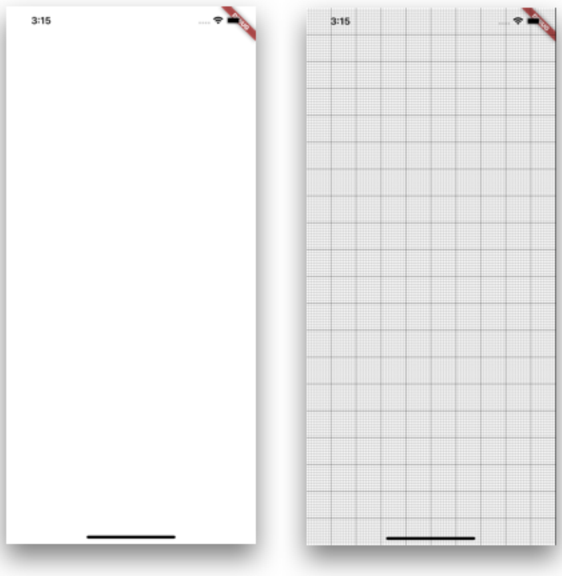

FIGURE 6.4: Screen Division with Grid

Every cell of the grid is now a logical block, which would help a user to align sizes. A screen has height of 20 blocks and width of 10 blocks. After that we can calculate height and width of a single logical block. The width is screen width / 10, and height - screen height divided by 20. Finally, we can create a red container, which is 10 logical block's heights vertical and 5 logical block's widths horizontal, in result there will be no visible difference on various screens. The platform uses a package *Flutter Screen Util*, which has already implemented a logic described earlier, however provides a wider scope of features.

Chapter 7

Testing

7.1 Unit tests

Unit test is a small code which checks single piece of functionality if it gives a correct output for all the possible inputs. During implementing the platform logic and writing a code, developer should definitely cover second with unit tests. There are few reasons for that:

- Team can be calm about the product quality. There are some cases which cannot be spotted even with human eye.
- The code base is changing and supplementing all the time, what might lead to new errors or break some of the old features. Running single file with all the tests is much more faster and easier than checking every connected feature yourself.
- *Flutter API* provides it's users a way to write unit tests for widgets.

All of the unit tests for the platform are written with *flutter test* package and can be found in the project repository.

7.2 Physical Testing

After all of the features from requirements are implemented, before the release platform should be checked 5-10 users, since the more people test it, the bigger is chance to find a flaw. If any bugs are found, they are followed with such procedure:

- Documenting a bug with steps to reproduce.
- Finding it and fixing.
- Writing regression unit tests.
- Delivering for review one more time.

Chapter 8

Conclusion

8.1 Summary

The main goal of the project was to research and implement main levels of web/mobile software development, while delivering a useful volunteering service. During the work such software development stages were researched:

- Planning.
- Designing.
- Creating a RESTful API.
- Developing and delivering web and mobile applications.

Huge advantage of the work is that fast modern comfortable tools are found for every technical stage of the project and every one of them is now known by the author in-depth, meaning that half of the main goal is already achieved. There could be a little downside, that if every of those phases would be developed by a separate person in the team, the platform would be finished in a more short terms, however, research of every tool also took quite a time. What is really hard in such projects for one developer is switching between those stages.

8.2 Further steps

Since the project is in an early phase of development, there is a lot of to do and improve in a future. However, ideas for the next release are these:

- **Technical**. Adding optional functionality from the requirements. To be more specific, it is really important to add a support of the images for users, posts and events on the platform.
- **Social**. Since Russia launched a full-scale invasion of Ukraine on February 24 2022 a lot of women with children were forced to leave the country. The idea is to attract new volunteers into the platform, who would organise different relevant events of various types for them. Simplest example of such events is psychological support.

Bibliography

Bayer, Michael (2006). *SQLAlchemy Website*. URL: https://www.sqlalchemy.org/ (visited on 05/29/2022).

Cambi, Daniele (2019). *Flutter — Effectively scale UI according to different screen sizes*. URL: https://medium.com/flutter-community/flutter-effectively-scale-ui-according-to-different-screen-sizes-2cb7c115ea0a (visited on 05/29/2022).

Fivetran (2021). *Comparison of PostgreSQL and MySQL*. URL: https://www.fivetran.com/blog/postgresql-vs-mysql#:~:text=PostgreSQL%20is%20an%20object%2Drelational,%2C%20ACID%2Dcompliant%20storage%20engine. (visited on 05/29/2022).

Google (2017). *Flutter Website*. URL: https://flutter.dev/ (visited on 05/29/2022).

Kudynenko, Olha (2009). *Tabletochki Website*. URL: https://tabletochki.org (visited on 05/29/2022).

National Cancer Registry, Ukraine (2019a). *Cancer in Ukraine, 2018 - 2019, All malignant tumours*. URL: http://www.ncru.inf.ua/publications/BULL_21/PDF_E/10-11-all.pdf (visited on 05/29/2022).

— (2019b). *Cancer in Ukraine, 2018 - 2019, All malignant tumours (Children)*. URL: http://www.ncru.inf.ua/publications/BULL_21/PDF_E/70-71-dity.pdf (visited on 05/29/2022).

Ramírez, Sebastián (2018). *FastAPI Website*. URL: https://fastapi.tiangolo.com/ (visited on 05/29/2022).