University for Applied Sciences Informatics Department Applied Informatics

LionsApp-Documentation

|  |  |
| --- | --- |
| Clients: | Prof. Dr. Stephan Kurpjuweit,  Jens Kohler |
| Professors: | Herbert Thielen,  Werner König |
| Semester: | Summer Semester 2023 |
| Due Date: | 30. March 2023 |

Table of Contents

[Team Introduction: 3](#_Toc130983191)

[1. Introduction and Goals: 4](#_Toc130983192)

[1.1: Requirements Overview: 4](#_Toc130983193)

[1.2: Stakeholders: 7](#_Toc130983194)

[1.3: User Stories: 8](#_Toc130983195)

[2. App Planning 10](#_Toc130983196)

[3. App Development 17](#_Toc130983197)

[3.1: Widgets used in the Process: 22](#_Toc130983198)

[3.1.1: Sized Box - Elevated Button: 22](#_Toc130983199)

[3.1.2: IconButton: 22](#_Toc130983200)

[3.1.3: TextFormField: 23](#_Toc130983201)

[3.1.4: Floating Action Buttons: 24](#_Toc130983202)

[3.1.5: CheckboxListTile: 25](#_Toc130983203)

[3.1.6: Datepicker + TimePicker: 26](#_Toc130983204)

[3.1.7: Calendar: 27](#_Toc130983205)

[3.1.8: Searchbar: 28](#_Toc130983206)

[3.1.9: DropdownMenu: 28](#_Toc130983207)

[3.1.10: Snackbar-Widget: 30](#_Toc130983208)

[3.2.1: Firebase: 31](#_Toc130983209)

[Firebase Authentication 32](#_Toc130983210)

[Confirmation Email before login 35](#_Toc130983211)

[Forgot Password Function 35](#_Toc130983212)

[Reset Password 36](#_Toc130983213)

[Login 36](#_Toc130983214)

[checkRool 37](#_Toc130983215)

[3.2.2 Firebase Firestore 37](#_Toc130983216)

[Donations 37](#_Toc130983217)

[Events 38](#_Toc130983218)

[Meetings 38](#_Toc130983219)

[Projects 39](#_Toc130983220)

[Rooms 39](#_Toc130983221)

[User chat 39](#_Toc130983222)

[Users 40](#_Toc130983223)

[3.2.3 Firebase Storage 41](#_Toc130983224)

[3.2.4 Firebase Hosting 42](#_Toc130983225)

[3.2.5 Firebase Cloud Functions 43](#_Toc130983226)

[flask-backend 43](#_Toc130983227)

[sendEmailwithAttachment 44](#_Toc130983228)

[sendNotification 45](#_Toc130983229)

[Payment-Backend 46](#_Toc130983230)

[Paymethode 48](#_Toc130983231)

[3.2.6 Receipt.pdf creation 54](#_Toc130983232)

[4.0: Deployment – Things you will need to do 55](#_Toc130983233)

[Glossary: 56](#_Toc130983234)

# Team Introduction:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Rolle | Email | Matrikelnummer |
| Tom Brauns | Product Owner | [Inf3621@hs-worms.de](mailto:Inf3621@hs-worms.de) | 676672 |
| Ayoub El-Mrabet | Scrum Master | inf3822@hs-worms.de | 677623 |
| Philipp Muders | Developer | [inf3455@hs-worms.de](mailto:inf3455@hs-worms.de) | 675908 |
| Kristiyan Ivanov | Developer | inf3563@hs-worms.de | 676281 |
| Omer Guimdo-Achoungo | Developer | [inf3249@hs-worms.de](mailto:inf3249@hs-worms.de) | 674699 |
| Marc Wieland | Developer | [inf3457@hs-worms.de](mailto:inf3457@hs-worms.de) | 675904 |
| Mikulas Willaschek | Developer | [inf3825@hs-worms.de](mailto:inf3825@hs-worms.de) | 677636 |
| Nico Hofmann | Developer | inf3207@hs-worms.de | 674656 |
| Saadet Ibrahimova | Developer | inf3530@hs-worms.de | 676242 |

# 1. Introduction and Goals:

The Focus of this application is to develop an application working as both an [Application](#_Glossary:), as well as a Web-App. It’s supposed to enable a simplistic donation process for guests and should have the option for [users](#_Glossary:_1) to sign up and therefore have access to more functionality and benefits on the application. The app includes a calendar, a project-catalogue, as well as events, a donation screen and a chat.

## 1.1: Requirements Overview:

#### A summarization of the functional requirements would look as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| ID | Requirement | Description | Reasoning |
| F-1 | Scanning a QR-Code | A User at an Event can scan a QR-Code portrayed on a Flyer | The QR-Code handles the Navigation to a specific Screen and therefore is essential |
| F-2 | Register | Users can register on the Application to save their data and have more functions on the Application | In order to avoid having to enter required data repeatedly, it is necessary that the User can register. |
| F-3 | Login | Users can choose to log in in the Application if their Account is already existing. | As an Account was already created, logging into that account saves the effort to have to enter important data repeatedly. |
| F-4 | Change User Data (logged in) | A Logged in User can change their given Data. | Contact info might change considering the address or the Email, or the user wants to change the password. All these functions need to be provided. |
| F-5 | Continue as Guest | A User can donate without being required to sign up. | Just because a User doesn’t want to sign up doesn’t mean he should be hindered from donating to a good cause. |
| F-6 | Log out | A user can log out | For navigation purposes, if there is a possibility to log in, you should also be able to log out |
| F-7 | Delete account | A user can delete the account | If you create an account requiring you to input data, you must be able to delete said account as well. |
| F-8 | Sign in with Google | A User can log in using their google credentials. | In order to make the account creation as simple as possible, offering google as an option (as a lot of people use it) helps. |
| F-9 | Sign in with Single Sign on | A User can log in using their apple sign on. | In order to make the account creation as simple as possible, offering single sign on as an option (as a lot of people use it) helps. |
| F-10 | Donate | Users can donate | As the core functionality of our application is the donation to specific causes, this MUST work. |
| F-11 | Donate using PayPal | Users can donate by using PayPal | As one of the most used payment methods, PayPal needs to be included in our application |
| F-12 | Donate using Stripe | A User can donate by using Stripe as payment method | Stripe is the currently used payment method for the prototype. It would be foolish to throw out already existing payment methods |
| F-13 | Donate using Google Pay | A User can donate by using Google Pay as payment method | Most people have access to a google account. By enabling google pay as a payment method |
| F-14 | Donate using Apple Pay | A User can donate by using Apple Pay as payment method | Apple users can have a simpler time donating to a good cause by using Apple Pay (less data required and simple donation process) |
| F-15 | Download receipt as pdf | A user can receive their receipt as a downloadable pdf | For legal reasons, users should be able to receive their donation receipts as downloadable pdfs |
| F-16 | Share on Facebook | A user can share their donation on Facebook | To Add a social Media application and to advertise the events further, Facebook seems like a good choice for our target group |
| F-17 | Share on Twitter | A user can share their donation on Twitter | To add an additional social media application, Twitter seemed like a good second option. |
| F-18 | Send Receipt using Email | A registered User receives their receipt per email as well | As donating can gain you some tax benefits, we added a functionality that sends automatically generate Email to the donators. |
| F-19 | See Events | Users can see the Events in a list | Enabling the users an overview of the Events, upcoming and current, can help the general attendance as well as motivation to donate. |
| F-20 | See Event Details | Users can click on Events to see detailed information of the Event | Getting further information on the Events is necessary to get the people motivated to attend and donate |
| F-21 | Create Events | Members and Admins can create Events | In order to have a fluent active app, you need the functionality to create new events. |
| F-22 | Create Chat  (Event creation) | Members as well as Admins can create a Chat for certain Events | The function to have a chat to communicate and plan for events is important |
| F-23 | Edit Events | Admins, as well as the people creating the Event can edit Events | In case there are changes to the events, it is important that you can change data entered in the Event |
| F-24 | Create Chat  (Chat tab) | Members and Admins can create Chatrooms | In case there is general need for discussion, an additional way to create a chat is provided. |
| F-25 | Edit Chat | Members and Admins can edit the chat | Chats can be edited afterwards, adding new participants, changing descriptions, or even adding a picture |
| F-26 | Delete Chats  (Chat tab) | Members and Admins can delete a created chat | For administrative purposes it is required that if you can create a chat (on whatever way), you can also delete it afterwards |
| F-27 | Delete Events | Event-creators and Admins can delete an existing Event | For administrative purposes it is required that if you can create an event(activity) (on whatever way), you can also delete it afterwards |
| F-28 | Search Events | Users can use the Search bar to search for specific Events | If the list of Events in the final version gets out of hand, a search bar can help to filter through the list. |
| F-29 | See Calendar | Members and Admins can see the Calendar | To have a visual realization, those two “administrative” roles should have a better overview of the Events |
| F-29a | Create Meetings | Members and Admins can create Meetings (Events with different inputs) | As some Meetings are purely organizational and of inner Circle nature, the whole public shouldn’t have access to these. |
| F-30 | See Projects | Users can see the Projects available | In order to know what projects are available they need to be listed somewhere |
| F-31 | See Project Details | Users can click on Projects to see detailed information of the project | As a name generally doesn’t tell you enough, a more detailed overview is necessary |
| F-32 | Create Projects | Admins can create projects | In case a new catastrophe occurs, admins can create a new project |
| F-33 | Edit Projects | Admins can edit projects | As updates might be required to an existing project, an option to edit projects should be provided |
| F-34 | Delete Projects | Admins can delete projects | If a project is no longer required, it can be deleted. |
| F-35 | Contact | Users can use a Contact Form to contact someone | Some sort of Contact should always be provided in case there is a need to get in contact with someone. |
| F-36 | Contact  (Feedback) | Users can use the Contact Form to give Feedback. | In case Users want to share opinions about the application, they have a place to share these with us. |
| F-37 | Imprint | Users can read up on the Imprint of the Application. | The Imprint includes all of the necessary Information the User could look for. |
| F-38 | Terms of Service | Users can read up on the Terms of Service | Users are required to have access to the ToS afterwards if they want to read up on them |

#### Graphic 1: This graphic depicts the functions in accordance with the User-types. The further you move to the right, the more functions that User has access to.

## 1.2: Stakeholders:

#### As Stakeholders we consider all the people benefitting from the direct implementation and release of the application

|  |  |  |
| --- | --- | --- |
| Stakeholders | Type of Stakeholder | Reasoning |
| Donators | Primary | As we aim to make the donation process as easy as possible, we always must think of the donators. |
| Sponsors | Primary | As with all projects, keeping an application running as well as running actual events that we use the application for requires funding. Those can be companies supporting the events, as well as organizations using it. |
| Lions | Primary | While the App isn’t directly in development for the Lions-Club, we must consider them our main audience group. |
| Lions Members | Secondary | As |
| Organizations | Secondary | If available, those are the organizations that handle contributing the funds to the actual problem. |
| State | Tertiary | Considering that donating involves money transactions, the state must at least be considered for legal reasoning |
| Donation Receivers | Secondary | As they are the ones who actually benefit from the gathering of money, we should keep them in mind. |

## 1.3: User Stories:

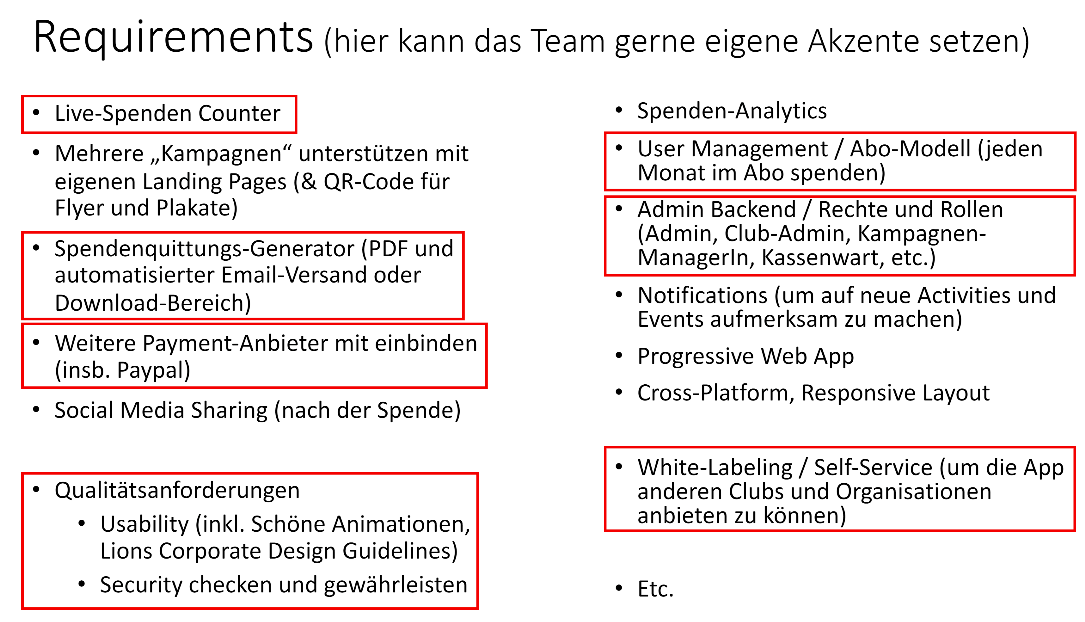
#### The core of each app is to focus on the main functionality at first, so that an application, having the core features implemented, can be used already. To best fulfill this approach, we’ll focus on the most important Use-Cases and User-Stories.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | User-Story | Function | Reference |
| US-1 | As a User I want to be able to log in to access my profile and save time. | Log in | F-3 |
| US-2 | As a User I want to register in order to save my data | Register | F-2 |
| US-3 | As a registered User I want to be able to change my Entered Data to keep it up to date in case something has changed. | Change User Data | F-4 |
| US-4 | As a Guest I want to be able to continue donating without having an account to have the less trouble as possible in trying to donate | Continue as Guest | F-5 |
| US-5 | As a User I want to be able to Scan a QR-Code to get to a specific Event. | Scanning a QR-Code | F-1 |
| US-6 | As a registered User I want to be able to log out to no longer be considered a registered User. | Log out | F-6 |
| US-7 | As a registered User I want to be able to delete my account in order to avoid my data from being saved. | Delete Account | F-7 |
| US-8 | As a User I want Sign in Using alternative Options (google / Apple single sign on). | Sign in with Google,  Sign in with Single Sign on | F-9, F-10 |
| US-9 | As a User I want to be able to Donate to help a cause | Donate | F-10 |
| US-10 | As a User I want to be able to use PayPal as a Payment method to ease the process of donating. | Donate using PayPal | F-11 |
| US-11 | As a User I want to be able to use Stripe as a Payment method to ease the process of donating. | Donate using Stripe | F-12 |
| US-12 | As a Donor I want to be able to receive a receipt as a downloadable pdf to have a receipt for tax returns. | Download receipt as pdf | F-15 |
| US-13 | As a User I want to be able to share my donation on Facebook to increase the traction of the Event. | Share on Facebook | F-16 |
| US-14 | As a User I want to be able to share my donation on Twitter to increase the traction of the Event. | Share on Twitter | F-17 |
| US-15 | As an Owner I want to send registered Users an E-Mail including their donation receipt. | Send Receipt using Email | F-18 |
| US-16 | As a User I want to see the Events in a list | See Events | F-19 |
| US-17 | As a User I want to See the Event Details to know what I’m “signing up for” | See Event Details | F-20 |
| US-18 | As a Member or Admin I want to be able to Create an Event(Activity) to collect funds for a good cause. | Create Events | F-21 |
| US-19 | As a Member or Admin I can create a Chatroom for a specific Event when creating the Event. | Create Chat  (Event Creation) | F-22 |
| US-20 | As a Member that created that specific Event or Admin I can edit the Event using | Edit Events | F-23 |
| US-21 | As a Members or Admin I can create an Chat using the Chat tab to plan and communicate with others | Create Chat  (chat tab) | F-24 |
| US-22 | As an Admin or Member I can edit the Chat to invite more Users or details. | Edit Chat | F-25 |
| US-23 | As an Admin I can delete existing Chats to clean up unnecessary chats | Delete Chats  (Chat tab) | F-26 |
| US-24 | As an Admin or Member that created an Event, I can delete that specific Event to declutter the Event List and the Calendar | Delete Events | F-27 |
| US-25 | As a User I can use the Search bar to search for specific Events | Search Events | F-28 |
| US-26 | As a Member or Admin, I can see the Calendar | See Calendar | F-29 |
| US-27 | As a Member or Admin, I can create Meetings which are only shown in the Calendar | Create Meeting | F-29a |
| US-28 | As a User I want to see the projects to know where the money I donate ends up. | See Projects | F-30 |
| US-29 | As a User I want to see a projects details page to get further Information about the project. | See Project Details | F-31 |
| US-30 | As an Admin I want to be able to create Projects to start attempts in helping when an catastrophe occurs. | Create Projects | F-32 |
| US-31 | As an Admin I want to be able to edit existing Projects in case something has changed to keep the information up to date | Edit Projects | F-33 |
| US-32 | As an Admin I want to be able to delete Projects in case they are no longer in need of assistance | Delete Projects | F-34 |
| US-33 | As a User I want to be able to contact someone in charge to help me with a problem I’m having. | Contact | F-35 |
| US-34 | As a User I want to be able to give Feedback in case I really liked / disliked something. | Contact  (Feedback) | F-36 |
| US-35 | As a User I want to be able to read the Imprint to get further details about the Application and their deployers. | Imprint | F-37 |
| US-36 | As a User I want to be able to read up on the Terms of Service to know what I signed up for | Terms of Service | F-38 |

# 

# 2. App Planning

Before we could get to work, we had to do some planning. So, we started by taking another look at the project we had presented to us and the requirements that our customer had defined:



#### Graphic 2.1: This is a Screenshot from the presentation in: [Spendenapp.pdf · main · TOP / 23s / TOP 23 project proposals and teams · GitLab (rlp.net)](https://gitlab.rlp.net/top/23s/top-23-project-proposals/-/blob/main/Spendenapp.pdf)

This helped us get a general idea but wasn’t enough to fully understand what the final version would have to look like, so we continued our research and started brainstorming. For this we used the previously designed brainstorming board that all of the groups created before:



#### Graphic 2.2: This depicts the brainstorming-board that all of the participants of the TOP-Project created regarding the “Spenden-App”

Now that we had a bunch of input combined with the original ideas from our customer, we started working on our understanding of the application and how it should work. In doing so we made a board ourselves, gathering our ideas that we thought would be necessary in defining a united understanding of the product we were about to develop.

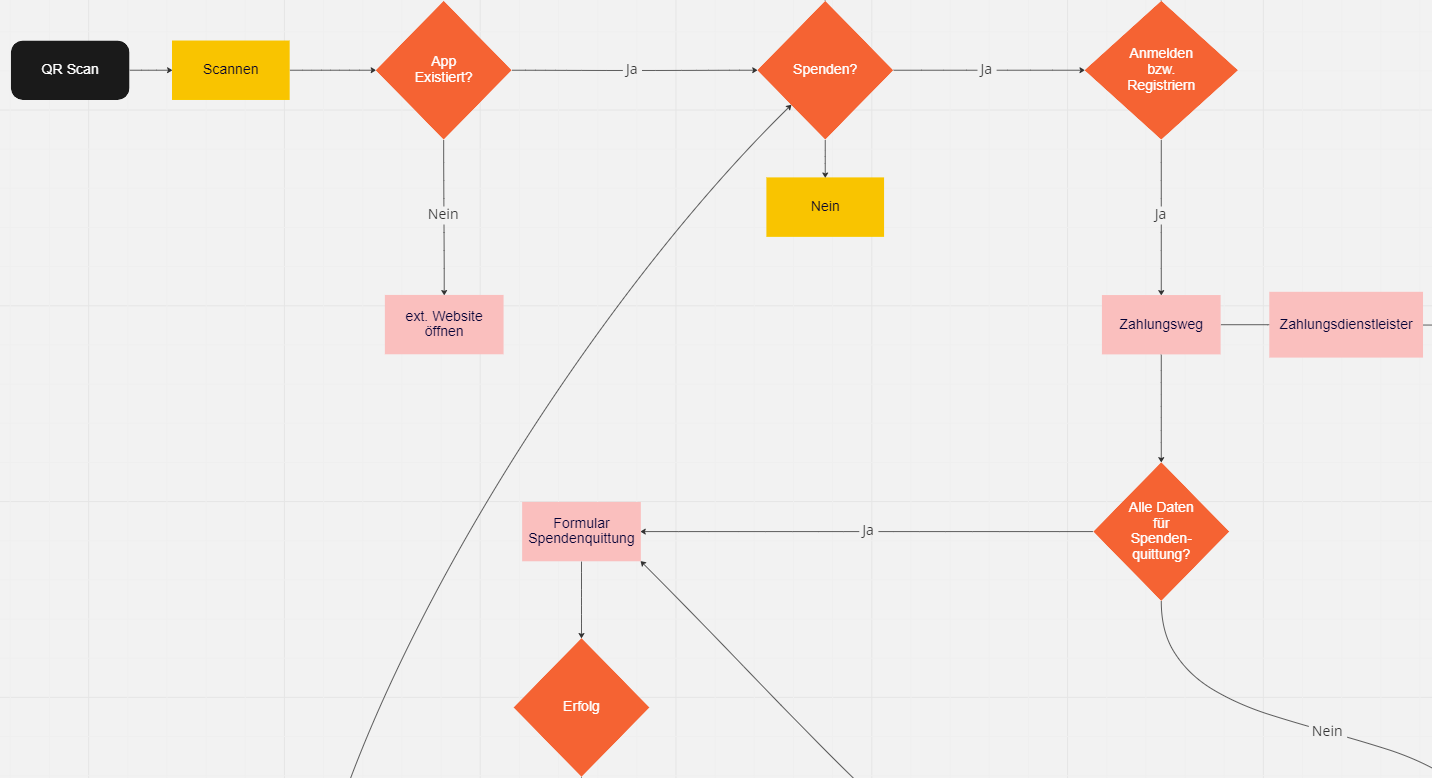
We then used those Ideas and compared them to the ones that were previously gathered by all the groups, adding a few of those ideas to our final realization, before starting to think about functionalities we wanted in that app.

In doing so, we realized quickly, that we would need a Database to store all the Data (be it user-related or for the events itself). Luckily, we had a bunch of dedicated Network-Security majors that volunteered to start the learning process regarding those Databases and decided on the Firebase.

We still wanted some sort of visualization. Afterall we were working with an application, so Mockups would be helpful for all of us to best understand which Screens we would need and what each of the Screens would require. We turned to <https://miro.com/> for this, as we could all work on it together and have a lot of features customizing it.

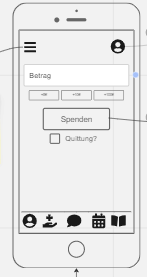
But where does one start when it comes to thinking about an application. We referred to something our [customer](#_Glossary:) defined previously. The focus of the application should be, that a guest could SCAN A QR-Code and donate to an Event (or Activity), without jumping through too many hoops. Considering that this was one of the core things our customer wanted, we started off thinking about how that would look like and created an “Epic” just for that occasion. The customer would scan a QR-Code, open the App, donate to the cause, would then have an option to log in/ register, OR just continue as a guest, select a payment method, for legal reasons input the required data for a donation certificate, before finally being presented with the Notification of the successful donation done.

The created a happy path of that process in Miro that looked something like this:



#### Graphic 2.3: This picture and the following wireframes are out of [Team Lion - Mockup, Online Whiteboard for Visual Collaboration (miro.com)](https://miro.com/app/board/uXjVPpZkFKk=/)

This implied, that a guest wouldn’t be required to input too much data, or even create an account in order to be able to donate, so we wanted to visually display the simplicity of the process.

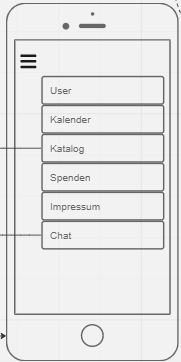
Now that we had a general process portrayed, we started thinking about how each of those pages would generally look like. In doing so, we created simple wireframes for each of the required Screens.

In doing so, we quickly realized, that this would require a bunch of pages and therefore some sort of Navigation to make it useable. So we started working with an Burger menu. In addition of that we thought it to be a nice feature to add a Bottom Navigation Bar for the core Screens that you would want to access. By thinking about it in detail, we started realizing, that this wouldn’t apply to the guest that would just want to donate without a lot of distractions. This started a discussion on which roles there actually are, so we agreed upon a guest, a member and some sort of admin initially.

#### Graphic 2.4: This Wireframe – taken from our miro board – represents our first interpretation of how the donation-page would look like.

We continued to create wireframes for each of the required Screens in the process, including buttons, text fields and first Icons that would require functions behind it in the respective screens later on.

In defining and using our Burger menu, we required an united understanding of what the application would have as Screens. So, we started thinking about what we would want in that.

After some discussing, we ended up on the 6 most important Screens, or rather Categories at this point, as each of these screens would have a bunch of screens following in the process. Those screens are the User tab, to customize data, change settings and administrate subscriptions etc, the calendar to portray events, the Catalogue to show the projects that the Lions Club does their events (activities) for, the donation tab, which would be our starting page, and the impressum (for legal reasons). We also noted that there would be some sort of an chat, even though we weren’t quite clear on how that would look in realization.

We created Wireframes for all of the screens we thought necessary in those categories aswell as further categories (such as the events, the contact form, the login page etc) and created the flutter repository.

#### Graphic 2.5: This Wireframe – taken from our miro-board – depicts our initial design idea of the burger menu (or custom drawer)

A few days later we got the chance to have a first chat with our customer, presenting him our general understanding and presenting him our Miro-board and the epic that we created for the general donation process. In this conversation, there were a bunch of right assumptions, but also a lot of newly presented topics, such as the difference between a member and a friend.

This led to a lot of discussions about the roles in specific, which we then pinpointed to avoid further confusion and to ensure a likeminded understanding of it:

#### 2.6: This graphic shows our united understanding of the roles involved in the application.

While we have 4 different Users of the application, only 3 of these are registered. Each of the roles has more rights and Screens available to them, ranking from the friend (registered guest) to the member (someone who received additional rights by an Admin), to an Admin (someone with the capabilities to create catalogue entries (projects), and give other friends the member role). We went ahead and defined each of the functions and screens available to each of the roles to understand them individually.

Having the screens available to us, we started working out our user stories and issues specific to the roles.

Now that we had the core screens thought of and wireframes created, we were able to start with the development process. We created a git repository, created a flutter project and started learning about flutter, as it was some time for each of us since we worked with the dart wireframe or flutter in general.

Since there is a lot of changes in understanding while working on an application, we realized that it would help realizing how the projects worked with the events regarding the databases, so we started to visualize it for us as well:

Ein Bild, das Text, Whiteboard enthält.

Automatisch generierte Beschreibung

#### Graphic 2.7: Whiteboard-Design of how we imagined the entire system to function regarding the databases.

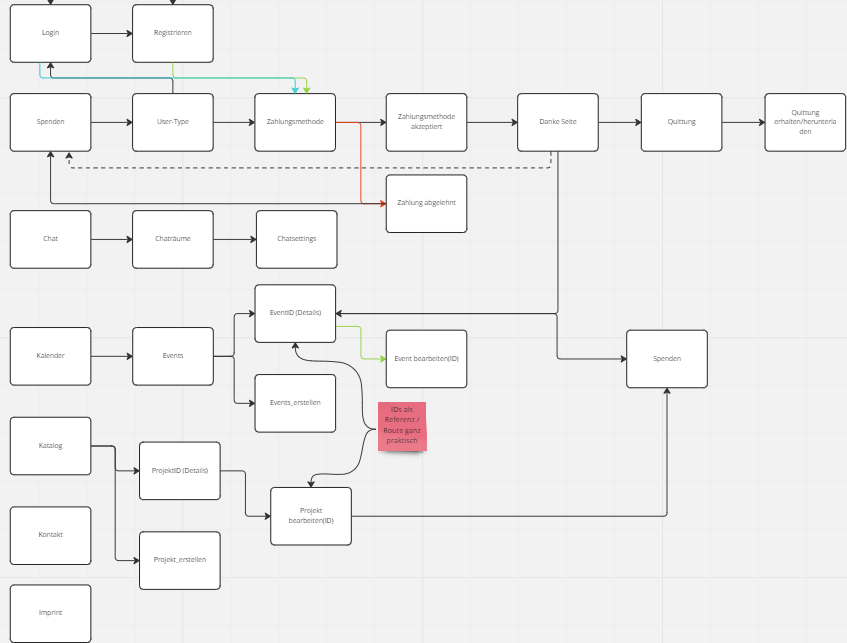
We’d have an Admin create a project with certain attributes (project name, description, further details, as well as a unique identifier). This project would then be stored in the Firebase storage.

Members can create an Event customizing the date of the event (starting date as well as ending date), define an description, select a purpose ( linked to the created projects) and an optional goal of the donation, as well as sponsors, a counter and an unique identifier.

In summarization, a member can create an event and optionally select from the existing projects as to what the money is saved for.

In the process of planning we also discussed, how we would best realize the QR-Code. For the QR-Code to work, we’d need some way to create an QR-Code, and for the QR-Code to link us to a specific page.

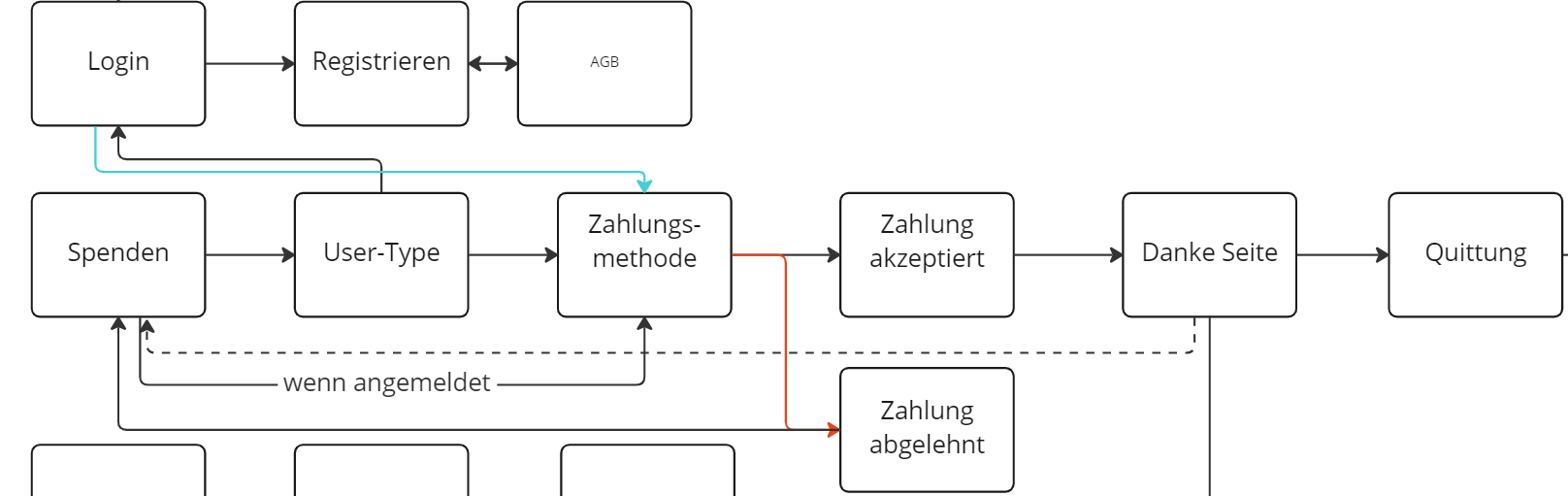
In thinking about the specific pages, we realized that we would have to implement routes into our structure. In doing so we thought of the official routes known to us and mapped them for us:



#### Graphic 2.8: The shown graphic represents parts of the routing process, as the entirety would be hard to depict in just one snapshot.

This graphic helped us figuring out the dependencies of the screens. It also encouraged the thinking process of where the navigation could take us. We started with core Navigation processes. Those being the Chat, the donation page, the calendar, the catalogue aswell as the user page and the login / register. We mapped where each of those processes would take us and in doing so got an even better understanding of how the app we tried to implement would work out in the end.

Let’s take a look at how the navigation and the routes in the donation process would look like:



#### Graphic 2.9: This is the custom diagram we used to modelize the routes required focusing on the donation process.

# 3. App Development

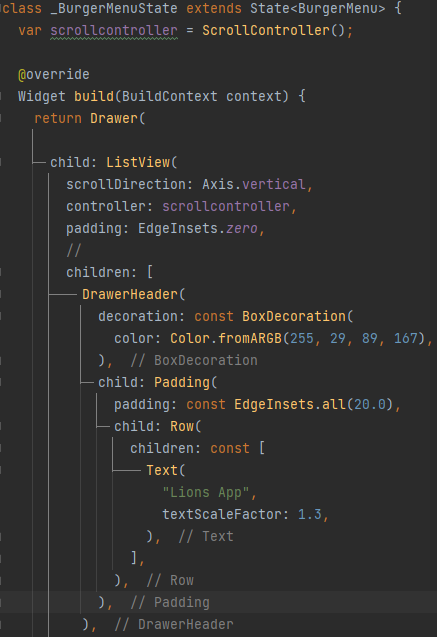
We started off with an empty Flutter-project and begun by creating a few very simplistic screens.

In doing so, we created classes (e.g. UserManagement) and defined a State for these classes. The general structure of flutter uses a build widget that returns a Scaffold. As one can already see, we customized a drawer with a class called BurgerMenu().

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Now let’s look at the Burgermenu, as it is included in most of our screens.

The Burgermenu is a StatefulWidget. It is scrollable and uses a scrollcontroller. In its core it is a Drawer, which is why it was defined as Drawer in the UserManagement-class earlier. The entire BurgerMenu functions as a ListView. All of the objects included are defined as children of that ListView. As the general implementation of its elements is similar, let’s take a look at some exemplary ListItems: First of all, there is the DrawerHeader. As one might assume, it it the Caption of the List.

We customized the Header a bit, although none of these changes are final and are up to be changed in the near .

Furthermore we created Header-ListTiles, which are just used to group the pages for the user.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Those Header-ListTiles look as follows: they have a title, which is just a Text with a certain Style ( here: bold), and a specific background Color: (here: grey).

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungThe Second type of ListTiles are the ones handling navigation in the BurgerMenu. While the header tiles were just there for organizational purposes, these ones actually have functionality attached to them. First we define, which role sees the specific ListTile (not every role can see every screen). We do that by accessing the privileges file that we created (will be looked at shortly). If the User has the certain privileges required to see that tile, they see this screen in the Burgermenu with an Icon at the start of it (leading) and a title ( name of the screen). In the onTap we define, where the Navigator sends us. First we pop the current State (the current Screen). This is required for performance reasons. Then we push to the Named Screen (here the ‘/Events’-Screen). The whole tile is considered an Container.

The general design of the Burgermenu looks something like this:

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

As already hinted at when discussing the Burgermenu-class, let’s look at the privileges:

Just like the previous classes, this one is a StatefulWidget aswell, with a starting privilege attached to it as a String. As there are four viable roles, those could be “Guest”, “Friend”, “Member”, or even “Admin”.

We return a placeholder object in this class.

Similar to us modifying our drawer with the BurgerMenu, we also modified our appbar Ein Bild, das Text enthält.

Automatisch generierte Beschreibungand created a class called “MyAppBar”. All that this widget required was a title.

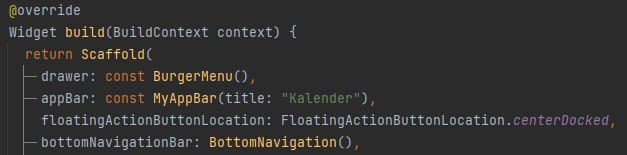
Let’s look at how that class works in detail:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungThis Stateful Widget has a build function including the context as BuildContext, and returns the AppBar, which takes the widgets used title (the parameter we need to provide). Additionally, this is only shown for registered Users ( Friends, Members and Admins). We did this because we added an Iconbutton , that’s supposed to lead to the User-Settings. Showing such an Icon for a User that’s not registered would be pointless.

The Appbar is the Bar portrayed on the top of the Screen.

Now that we looked at the Appbar and the drawer, lets look at the BottomNavigationBar aswell:

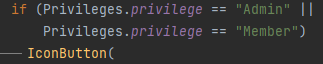


For this, we created our own class called “BottomNavigation()” which is being used on some pages.

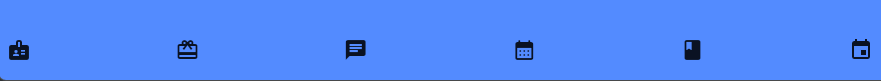
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In it we used a Row to display the Icons (being IconButtons), with each of them navigating to our specific routed screens. In this example it would handle the Navigation to ‘/User’.

 Furthermore we only displayed some of the Icons in the Navigationbar if the User had the specific privileges to access that page.

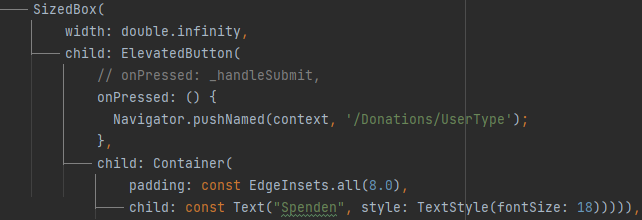
#### This is the portrayed BottomNavigation for either an Admin or an Member:



## 3.1: Widgets used in the Process:

#### Instead of going over each of the files that are bound to change in the development process, it would be more interesting to just talk about the main widgets we used in the application:

### 3.1.1: Sized Box - Elevated Button:

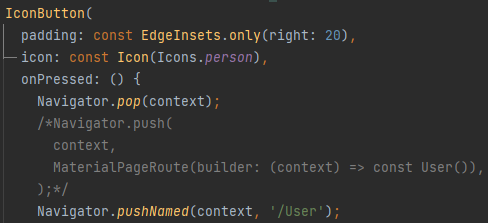


This was one of our most used buttons. It has an onPressed() function at which you can define, what actually happens if the Button is pressed, as well as an child attribute, which can be something like an Container as we used here to display a Text which will be portrayed on the Button. We use a Sized Box to wrap the Button into to keep the spacing simple.

#### Quick depiction of how such a button can look like:



### 3.1.2: IconButton:



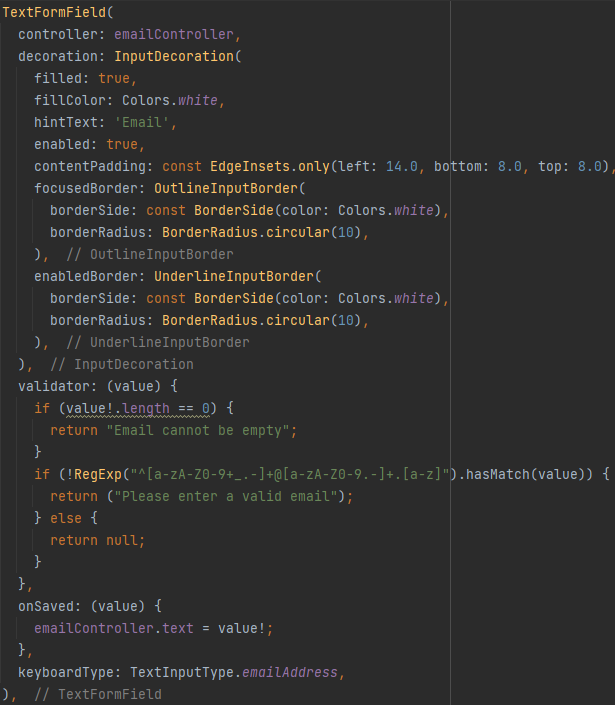
The Icon Button consists of an icon attribute, as well as an onPressed: ()-function, that defines how the button handles a press. Furthermore, it can have padding as attribute, which describes the spacing to other objects.

#### App-Representation of how an IconButton can look like:



### 3.1.3: TextFormField:

As we want to validate some Data that is being input by the user, we used the TextFormField. In here we have an controller, a hintText leading the user to what sort of data is required here. We also have a validator which we use to validate the input of the users by just checking the provided value. The TextFormField has an onSaved: ()-function which updates the value after accessing the validator. We can also provide a keyboardType for that input so the user gets presented with the right medium required for this field (e.g., wouldn’t want numbers if it’s a text field).



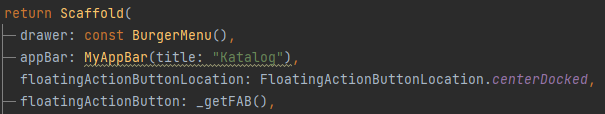
#### The following code snippet depicts an example of a textformfield:

#### 

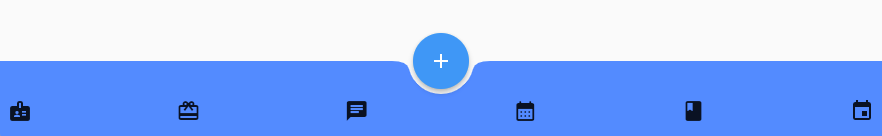
### 3.1.4: Floating Action Buttons:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungFloating Action Buttons (FABs) can be valuable tools when it comes to displaying Features (such as adding / removing elements). We used this coupled with the privileges previously defined, as not every user has access to the buttons. FAB’s function similar to an IconButton, with the exception that they are floating a layer above the initial page and are not affected by the widgets below it. They have an onPressed-function as well as a child attribute that’s an Icon. We call it in the Scaffold and define the Location of the Floating Action Button.



#### This snippet portrays the Floating Action Button in the catalogue.dart-file:



### 3.1.5: CheckboxListTile:

Checkboxes are used in various ways, be it in the Form of an AGB, or in this case the option, if a Textfield should be portrayed:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

This CheckboxListTile has a Title, aswell as a value. If the value changes, the onChanged-function is accessed. We also added a TextField, which is only portrayed, if the \_hasDonationTarget function returns a true (as it is used as a Boolean).

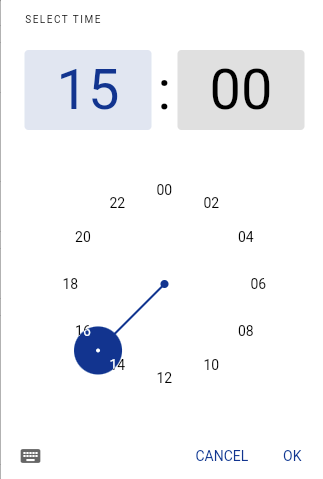
### 3.1.6: Datepicker + TimePicker:

Used in the creation of Events, Members and Admins can select a date, starting off with the initialDate (which is the current Date), and then get to select the time, being presented with a starting time of 15 (being 3pm). If both of the inputs are valid, the Date is added as completeDate and is added in the Calendar.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

#### The following two snippets display the date picker, as well as the time picker in the event\_editor.dart:



### 3.1.7: Calendar:

For our Calendar, we used the “calendar\_view”-dependency:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungTo fill the Calendar with actual data, we used the CalendarEventData()-function, which is provided in the calendar\_event\_data.dart. By using this Class, we were able to link our input Data to the actual Calendar. Further, we used the event\_controller.dart.

The eventController Function is provided in each of the three Views (Daily, Weekly, Monthly), and handles the linking of the events provided from the snapshot to the actual Calendar.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

#### This is how our implementation of the calendar looks like:

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

### 3.1.8: Searchbar:

We use the Searchbar for Lists that have increasingly a lot of Data stored. We use it as a Textfield and have a setState()-Function that filters the input that we enter. Further, we added a hint and an Icon to help with the usability:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The actual filtering and Search-function looks something like this: We grab the data that we pulled from our snapshot, and show all those, where the event, or rather the ‘eventName’ attribute (converted to lower cases) matches the searchQuery(converted to lower cases):



#### This looks something like this in our application:



### 3.1.9: DropdownMenu:

As the Users can select between multiple Options, Burgermenues are a nice way of dealing with just that:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungWe use the DropdownButtonFormField, with a value as reference. For the items we make use of the Category class we defined:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungEach of the elements has a “.png”-Element and a categoryname defined as Text.

the onChanged-Function handles the category change. If the value is null, the current value remains.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

#### This depicts the Burger menu we used in the project\_editor.dart:



### 3.1.10: Snackbar-Widget:

Communication is important when it comes to usability. In keeping that in mind, we added Snackbars to portray messages on the users screens on certain points of the app. E.g. if a User logs out, a snackbar is portrayed letting the user know he is now logged out in addition to the navigation to another screen:

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungIn either an onTap or onPressed-function, the user can implement a snackbar. Further, you can customize it to help the user in understanding what happened. We chose a green background-Color here to let him know it was a success. Further, we added a Text, to deliver further context, and a duration defining how long the message will be displayed.

This looks as follows in our application: 

#### The previous graphic depics the code defined in the burgermenu.dart upon pressing “ausloggen”

3.2.1: Firebase:

At the start of the project, we decided to use Google Cloud, in particular Firebase, as our serverless backend. The Picture below shows our architecture in Google Cloud. The following readings explain our 5 major parts of Firebase which we are currently using.  
  
Ein Bild, das Diagramm enthält.

Automatisch generierte BeschreibungSystemkomponentendiagramm

First of all, the Firebase requires to initialize the instance as we did in our main.dart:   
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Firebase Authentication  
Basically, Firebase Authentication allows us to register several users within our project and stores  
the sensitive data, for example passwords, in an encrypted manner. Additionally, it provides a few more things like a “password forgotten” or “reset password”-function, which we are using and therefore explain. Simultaneously, we need to store some more information about users, which lead us to use Firebase Firestore besides Firebase Authentication alone.

As said before we had to modify the authentication process to fit our purposes.  
So in case an unregistered user (guest) wants to create a user account for our donation application, they need to specify their personal information like their first name, last name, email-Address, password and residence. In our case, the email-Address is the unique key for the users account. This is our function to do this.

For the actual verification we require the email and the password, while we save the rest of the data in the Firestore in the “postDetailstoFirestore”-function:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The “postDetailsToFirestore-function gets the following data:  
- string values firstname, lastname and residence  
- string rool which is “Friend” by default and can only be changed by Admins,   
- a device token String which is needed to send push notifications to mobile devices  
- a bool-value which gives users the ability to disable notifications (enabled by default)

The code regarding these functions looks as follows:

Ein Bild, das Text enthält.

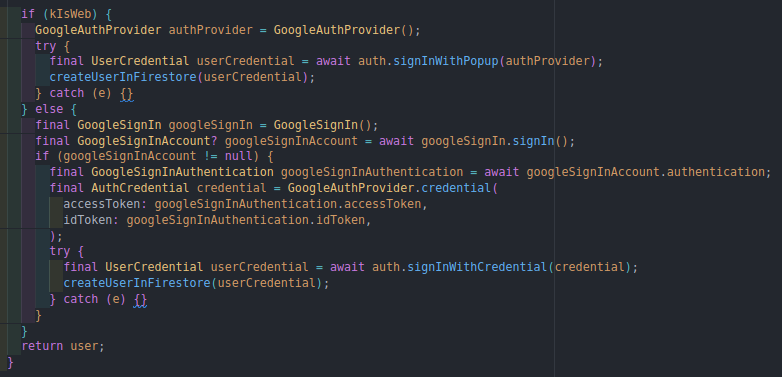
Automatisch generierte Beschreibung

We also want to give guests the possibility to register using their Google account, which is handled by our function signInWithGoogle() and requires a little bit more code, because the sign in process in the web browser differs from the one on mobile devices. But it basically does almost the same as the previously discussed signUp()-function.

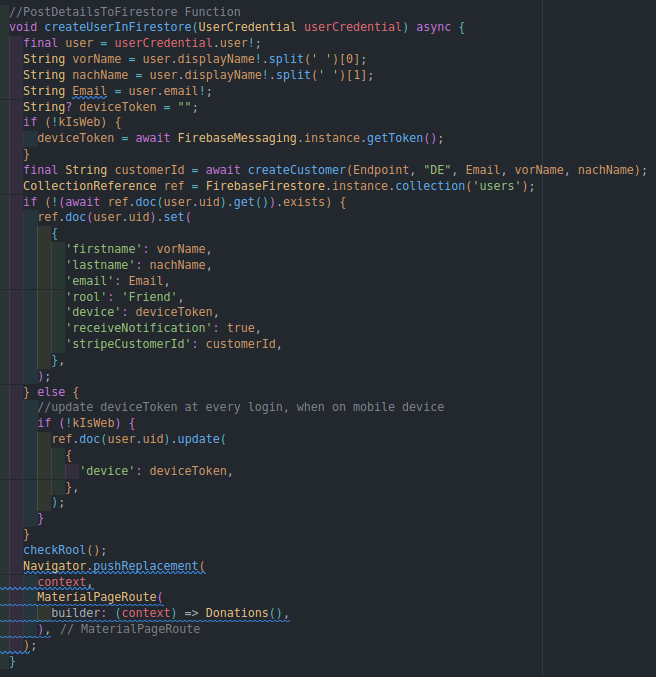
Instead of taking the password to reference the user, it takes an access token and ID token, which we get from Google.

In addition to that, we check if there is already an existing user with this specific Email Address in our Firestore-collection “users”, if not it will create an account with all additional data as mentioned before.

This is our code which handles the google Sign in different ways depending on the platform, the application is used. It works similarly to the above SignIn Function. It does the login process in google



and additionally creates a user in firestore, checks his role and navigates to our donation page



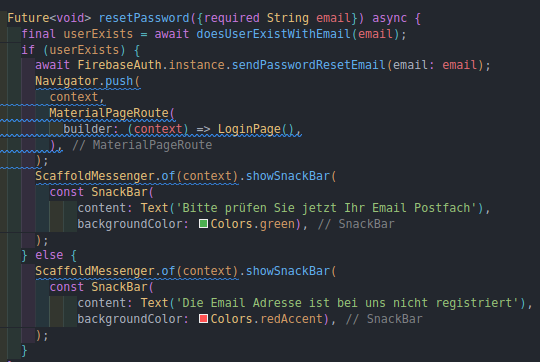
Confirmation Email before login  
After the registration process, the new user receives a Mail from us to confirm his Email Address, which is mandatory to proceed with the login. We further let the User know what he must do by implementing a Snack Bar:

Ein Bild, das Text enthält.

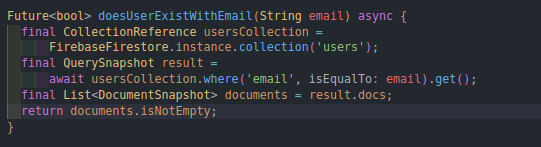
Automatisch generierte Beschreibung

### Forgot Password Function

We also provide the functionality for users to reset their password. Which requires the Email Adress of the user.



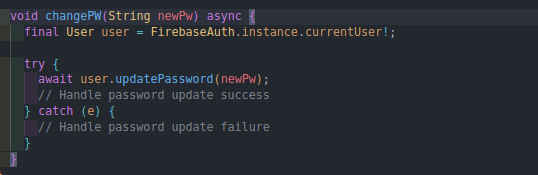
Before sending the mail, we verify if the mail does even exist in our database:



### Reset Password

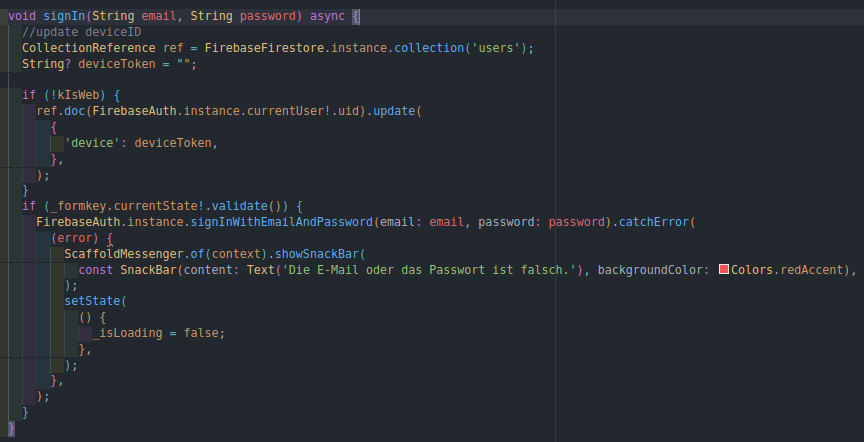
For logged in users who want to change their password by themselves we provide the changePw() function which takes the new password and updates it in Firebase Authentication.

The function is kept simple and just takes the new password as string. And updates the password in Firebase.



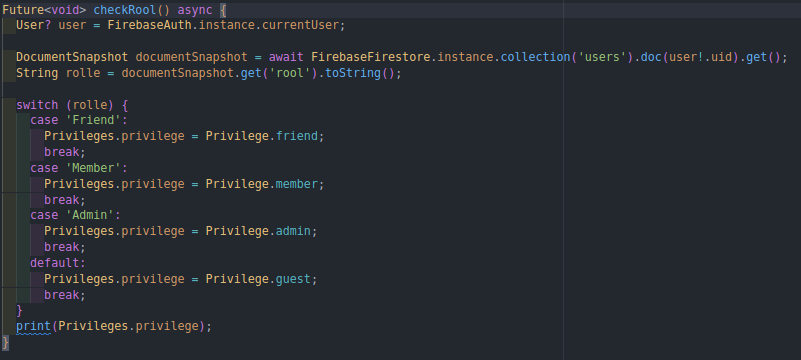
### Login

Here we have our signIn()- function which takes the entered email and password and returns a success message from firebase if the user exists, and the password is correct. Furthermore, it navigates to our Donations() Page:



### checkRool

As the user logs in, the privileges of the user are checked to display the correct burgermenu and give him the role-based access to resources. For this, we implemented our function checkRool(), which retrieves the value of the rool-value of the current “user” in the Firestore collection, referenced by the current logged in User ID. It compares the retrieved value and assigns the specific role to our enum Privilege.privilege through a switch-case.



## 3.2.2 Firebase Firestore

Our application requires several Firestore-collections which will be listed, and their use cases explained here. Entries are called documents in Firestore and they have their own unique document-id. As Example, the document (id: XYZ) has several fields such as:   
Firstname  
Lastname  
Email Adress

Fields that we do not use, but still get created in some cases, from some included flutter packages, are not listed and can be ignored.

### Donations

Collection of all successful donations with the Fields

|  |  |
| --- | --- |
| amount | donated amount in Euro |
| date | date of the donation |
| event\_id | references a certain event to which the donation was made |
| event\_name | stores the name of event to which the donation was made |
| receipt\_url | contains the url to our firebase storage which includes the receipt for this specific donation |
| user | contains the user, which executed the donation |

### Events

Collection of all available Lions events (for example Martini-Konzert) and has the following fields:

|  |  |
| --- | --- |
| chat\_room | in case the creator of the event, created a chat room for this specific event, we find the chat\_room id here |
| creator | User Id of the event creator |
| currentDonationValue | summarize the value of all made donations for that specific event |
| startDate | beginning of the event |
| endDate | end of the event |
| eventInfo | information about this event |
| eventName | name of the event |
| image\_url | stores the firebase storage url, which contains the picture for this event |
| ort | contains the location where the event takes place |
| projekt | events have an overarching project for which the donations are collected |
| spendenZiel | Donation target in euro that is aimed for |
| sponsor | if a certain company supports the event, sponsor is the name of the company |
| sponsor\_img\_url | stores the url to firebase storage, if there is a support his brand will get displayed in the donation process |

### Meetings

|  |  |
| --- | --- |
| startDate | The start date of the meeting |
| endDate | The end date of the meeting |
| Description | Further details to the meeting |
| Location | Place of the meeting |
| Name | Name of the meeting |
| url | If necessary, the user can enter an url |
| Creator | The reference as to who created the Meeting |

### Projects

Projects are overarching reason or happening for which the donations are being collected (for example a project that supports the fight against childhood cancer or a project that supports victims of natural disasters). The Regarding Data looks as follows:

|  |  |
| --- | --- |
| background | some information about the project |
| category | for example, environmental protection or humanitarian aid |
| image\_url | url to the picture in firebase storage to that specific project |
| name | name of the project |
| support | what Lions does with the donations to help out |

### Rooms

Chat rooms that can be created either when creating an event or manually by every member or admin. Every chat is either a direct chat between 2 persons or a group chat. Each chat creates their own instance inside this collection:

|  |  |
| --- | --- |
| messages | Sub-collection of all messages sent in this room |
| authorId | sender of the message |
| createdAt | date of sending the message |
| text | content of the message |
| type | either text, image or file |
| createdAt | date of group creation |
| imageUrl | firebase storage url that refers the Chatroom picture |
| metadata | additional information about the group |
| beschreibung | description of this group |
| name | Group name |
| type | either group or direct |
| userIds | List of all users in this group |

### User chat

Every time an authorized user load the page rooms.dart, our code checks, if the current user is already in the user\_chat collection. If not, it creates one by taking firstname, lastname and imageUrl from the current user out of the “users” collection:

|  |  |
| --- | --- |
| createdAt | date of creation of the instance of user\_chat |
| firstname | firstname out of collection “users |
| lastname | Lastname out of collection “users |
| imageUrl | ImageUrl out of collection “users |

### Users

Here, we find all users that created an account in our application:

|  |  |
| --- | --- |
| email (Primary Key) | Email of that user. |
| firstname | First name of the specific user |
| lastname | Last name of the specific user |
| streetname | Streetname of that user |
| streetnumber | Streetnr of that user |
| cityname | City he is from |
| postalcode | Postalcode of said city |
| device | stores the device Token of the user, if they register on a mobile device. Needed for Push Notifications |
| receiveNotification | true by default, can be disabled by the user if they don't want to receive notifications about upcoming events on their mobile device |
| StripeCustomerId | Id created to reference the user in the payment process and to allow subscriptions |

## 3.2.3 Firebase Storage

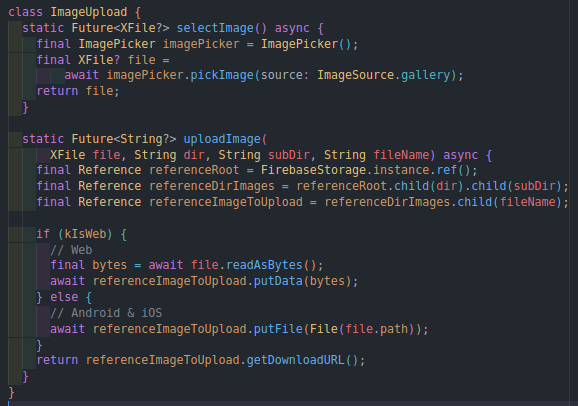
We are using several Firebase storage directories. The main purpose for using Firebase Storage is to store files or pictures that the user can upload and retrieve.

|  |
| --- |
| Basically, our Firebase Storage directories can be divided into 2 classes. The first one being the ones that store images: |
| **event\_images** |
| **images\_sent\_in rooms** |
| **room\_images** |
| **sponsor\_images** |
| **user\_images** |

|  |
| --- |
| and the latter of those being the ones that store files such as PDF: |
| **donator\_receipts** |
| **files\_sent\_in\_rooms** |

We need to upload images in several scenarios, so we did an own class that handles the image upload. The process is different on web and on mobile devices. Our Class opens ImagePicker for the user to be able to select an image from the local storage.

The file returned from this function is used for the function uploadImage, which also take a directory and subdirectory which determines where the uploaded file is stored in our Firebase storage.



Our second use case for our Firebase storage is the file upload which works similarly to the above example. Just instead of the imagePicker packet we use the FilePicker packet from Flutter and continue analogous.

3.2.4 Firebase Hosting

In this Chapter we will discuss the hosted webpages we have deployed. Those are being deployed by using an CI/CD-Pipeline of git in Firebase Hosting. Furthermore, Firebase hosts those pages by keeping the pipeline running and by updating the changes.

Those Pages would be our team page, presenting our team and our application in a summarized way, as well as including our Poster and our Video etc. The other one would be the actual Web-Application that we’ve discussed in detail in this documentation.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungAs the graphic out of the firebase.json depicts, the Firebase hosts a build / web version.

Ein Bild, das Text enthält.

Automatisch generierte BeschreibungFurthermore, whenever the Pipeline is deploying changes, it can be seen in the gitlab repository and once the build is finished, the changes will be deployed to either the team webpage, or the webapplication:

If the pipeline succeeded, it will be deployed, and it only succeeds if the build passed and the deploy passed (the two stages depicted as green checkmarks).

3.2.5 Firebase Cloud Functions

Cloud Functions for Firebase allow us to automatically run backend code in response to events triggered by Firebase features and HTTPS requests. The JavaScript code is stored in Google's Cloud and runs in a managed environment.

### flask-backend

We authenticate to Stripe and Paypal with secret keys. In order to keep these keys secret, it is important to implement all payment processes (i.e. Stripe and Paypal API calls) on the backend. For this purpose, we wrote a small REST API in the Python framework Flask that tunnels all Stripe and Paypal requests and adds the secret keys.

To host the backend cost-effectively, we decided to host it as a serverless Google Cloud Function. For a serverless cloud function, only the computing time that is actually used is charged.

However, a serverless function usually only has one entry point - but our Flask API has multiple routes. To avoid having to create a separate cloud function for each route (which would be quite a lot of work), we have added a function "run\_on\_google\_cloud()", which serves as the single entry point. Within this function, the request context is copied and a new context for Flask is created from it. The function run\_on\_google\_cloud() converts a Cloud Function Request to a Flask Request. This trick makes it possible to accommodate several routes in a single cloud function.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The code comes from this tutorial: <https://medium.com/google-cloud/use-multiple-paths-in-cloud-functions-python-and-flask-fc6780e560d3>

If the REST API is executed locally or on a "real" server, the function run\_on\_google\_cloud() is not needed.

### sendEmailwithAttachment

To be able to send mail with our cloud function we need to specify our SMTP host and a port that supports encrypted mail transfer, like we did in our code. Additionally, we give the function our credentials, so that we can send mails via our info@serviceclub-app.de email.

Here we have our function for sending mails that triggers our JavaScript function in index.js by an HTTP Request which we are doing in our contact.dart, to receive messages/feedback from our users.

Here we have our dart function that contains our HTTP Trigger for our JavaScript cloud function. We are using this once in contact.dart to be able to receive feedback and messages from our user. And again, in donation\_receviced.dart where we send mails automatically with the receipt attached to the mail, in case they have specified an email.

Ein Bild, das Text enthält.

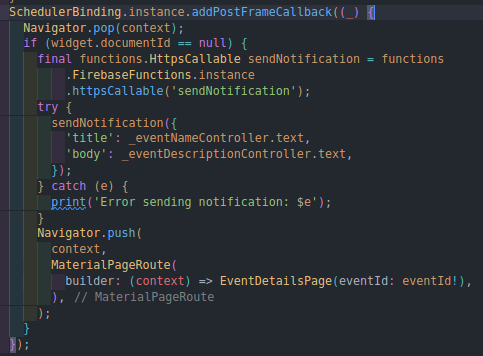
Automatisch generierte Beschreibung

### sendNotification

To send push notifications to mobile devices we use the previously mentioned device tokens. So, we are collecting all available device token from our firestore collection “users” and sent a notification to every token in our list.



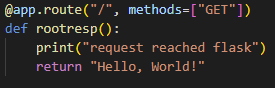
Right now, those notifications are being sent when a new event is created, so that every registered user receives a notification about the upcoming event. In our Flutter code, we use the HTTP Request as trigger to distribute the notification to the devices. The input for this function is the specified event name and event description, which we collect from the text controller.



### Payment-Backend

We use a Python Flask, deployed in Firebase Functions, to securely handle API calls to Stripe and PayPal services and our App.

A Python Flask Function looks like this:



The private/secret Keys are only available on the Backend and not Clientside and can only be changed there.

The Backend follows the Stipe and PayPal documentation for implementation of API calls to the respective services (PayPal API documentation: https://developer.paypal.com/api/ , Stripe API documentation: <https://stripe.com/docs/api> and implementation guide: <https://stripe.com/docs> ). To call to the Stripe API we use the ‘stripe’ Package, which is available for Python, which handles the API calls while providing readability.

A Python Flask Function with a Stripe API call looks like this:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

To call to the PayPal API we make a REST API call, which is done through the ‘request’ Package from Python. A first-Party Solution over GraphQL from Braintree was not an options, as we went with Firebase to host our Front- and Backend.

A Python Flask Function with a PayPal REST API call looks like this:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

## Paymethode

The Paymethode widget is given the Project/Event Id, the type of the Id, the Donation Amount, the status and whether the donation is being made as a subscription and how long or not.

The Paymethode widget changes depending on whether the user uses the site via a web browser or the mobile app. This is detected by the ‘GetPlatform’ class:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

When viewed with a web browser, the site displays a button for payments over Stripe with the label ‘Karte’ and a button for payments over PayPal. When viewed with the mobile app the site displays a for payments over Stripe with the label ‘Karte’ and a button for payments over GooglePay, which makes payments over stripe, but with the use of the users google account.



Note that when the user uses the subscribe functionality, only a button for payments over stripe is displayed.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung­

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The OnPressed classes are also handled via GetPlatform:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The GooglePay button is being provided by the ‘pay’ Package by Google and is configured using the ‘paymentItems’ in the widget and the ‘default\_payment\_profile\_google\_pay.json’. The Stripe button in the mobile app is using a paymentsheet provided by stripe over the ‘flutter\_stripe’ Package. The API calls to the Payment Backend are being made with the ‘http’ Package like this:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The response data is being returned as needed, in this case with the Checkout link, the description of the payment and the total amount in the String List ‘paypalObject’.

Depending on whether the payment was cancel or successful the return url from Stripe and Paypal directs the user to the ‘Paymethodecancel’ or ‘Paymethodesuccess’:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

When the user is redirected to ‘Paymethodecancel’ a Snackbar is displayed with a notification, that the Payment was cancelled and the user is redirected back to ‘Paymethode’, were they can try again.

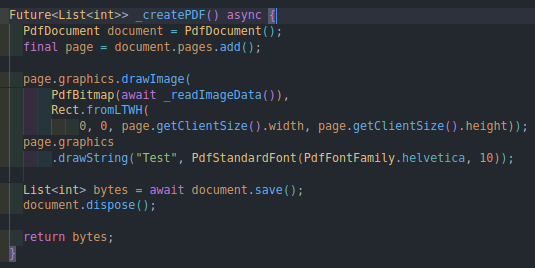
Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

When the Payment was successful the user is being redirected to ‘Paymethodesuccess’, where they are being immediately redirected to ‘DonationReceived’

3.2.6 Receipt.pdf creation

German laws require to create a receipt for every donation. So, we built a function that does this for us, which you can find below. After the file is created, it gets uploaded with the above explained function into our firebase storage directory. Additionally, if the donator is a registered user, they receives an automatic email with the receipt attached.



# 4.0: Deployment – Things you will need to do

In this chapter we will let you know what will need to be changed or what is required in order to run this application appropriately:

For starters we will assume that the developer understands how flutter works and how to properly set up the IDE to start the Flutter application.  
Now that the IDE should be running and the project is downloaded, lets go over the basic steps of ensuring that the application will run as intended:

* Run ‘pub get’ in the pubspec.yaml so you have the versions required to run the project properly.
* As we worked with our own data and accounts, those links will have been removed in the final version so the customer will need to input their own data so that the app and the payment will run accordingly. Changes will need to be made in the following files:

Payment:  
 -you need:   
 -a Paypal developer/business account   
 -a Stripe developer/business account  
 -a Google Pay & Wallet developer/business account   
 -change Keys in the Python Flask Backend (file app.py at the top)  
 -change following Data in the assets/default\_payment\_profile\_google\_pay.json  
 -enviroment (if you plan on going live)  
 -stripe:publishableKey  
 -merchantId  
 -merchantName  
 -allowedCardNetworks (depending on what paymethodes you want to allow)  
 -change Stripe.publishableKey in lib/payment/stripefunc

# Glossary:

Application: A final, portable version of the Software, in this case a mobile version.

User: Every type of person having access to the Application and making use of the functions.

Guest: Someone who uses the Application but does not log in / register to the page

Registered User: A registered User is a User that registered on the Application / logged into the Application. This refers to Friends, Member or Admins.

Logged In User: A logged in User is a User that is currently signed into the Application with an Email as well as a password.

Friend: A Friend is a person that registered on the Application

Member: A Member is someone who has more rights than a Friend. They have been given the role Member by an Admin and are the organizational Users that can create Events and access the Calendar.

Admin: An Admin is a Member in its core functionality, but has further rights, such as creating new Catalogue Entries (Considered as Projects in our Application ) as well as Manage the rights of Users.

Catalogue: The catalogue has all of the projects available portrayed in it in a List.

Projects: Projects are categories that summarize an catastrophe that occurred around the world. In the project, you can read up on the catastrophe, as well as see the measurements that are being taken to help. They are listed in the catalogue.

Events: Events are date bound Activities, that have the purpose of collecting money to donate for one of the Projects. They are being displayed in a List.

Donation Receivers: As Donations are gathered, the Receivers refers to the Organization or the people that benefit from that money.

Customer: This refers to the person that contracted us to develop this Application (in this case Prof. Dr. Kurpjuweit)

Database: Storage medium to store all of the Data , be it Events (Activities) , Catalogue entries (Projects), or User Data ( Email, Password etc.)

Firebase: Database handling our several data collections (such as storing user data)

Wireframe: Used as a Visual Prototype, without any functions.

Screen: A page that’s being portrayed in the application.

Burger menu: a navigation menu mostly implied at the top of the screen with an icon. By clicking on it, a List of entries is shown that can be used to navigate to specific screens.