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 |

Inhalt

[Team Introduction: 3](#_Toc128570236)

[1. Introduction and Goals: 4](#_Toc128570237)

[1.1: Requirements Overview: 4](#_Toc128570238)

[1.2: Stakeholders: 7](#_Toc128570239)

[1.3: Use Cases: 8](#_Toc128570240)

[1.4: User Stories: 8](#_Toc128570241)

[2. App Planning 10](#_Toc128570242)

[3. App Development 17](#_Toc128570243)

[3.1: Widgets used in the Process: 22](#_Toc128570244)

[3.1.1: Sized Box - Elevated Button: 22](#_Toc128570245)

[3.1.2: IconButton: 22](#_Toc128570246)

[3.1.3: TextFormField: 23](#_Toc128570247)

[3.1.4: Floating Action Buttons: 24](#_Toc128570248)

[3.1.5: CheckboxListTile: 25](#_Toc128570249)

[3.1.6: Datepicker + TimePicker: 26](#_Toc128570250)

[3.1.7: Calendar: 27](#_Toc128570251)

[3.1.8: Searchbar: 27](#_Toc128570252)

[3.1.9: DropdownMenu: 27](#_Toc128570253)

[3.2: Firebase: 28](#_Toc128570254)

[Glossary: 29](#_Toc128570255)

# Team Introduction:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Rolle | Email | Matrikelnummer |
| Saadet Ibrahimova | Product Owner | inf3530@hs-worms.de |  |
| 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 |
| Tom Brauns | Developer | [inf3621@hs-worms.de](mailto:inf3621@hs-worms.de) | 676672 |

# 1. Introduction and Goals:

The Focus of this application is to develop an application working as both an [Application](#_Glossary:), aswell as an Web-App. It’s supposed to enable a simplistic donation process for Guests and should have the option for Users to sign up and therefore have access to more functions and benefits on the application. The App in itself includes an calendar, a project-catalogue, aswell as Events, an donation screen and a chat.

## 1.1: Requirements Overview:

#### Functions include but are not limited to:

* Scanning a QR-Code
* The Login-Function
* The register-function
* The Change-User-Data-function
* Donate
* Continue as Guest
* Select Paymethode
* Share Donation
* Get a Donation-Receipt
* Display Events
* Display specific Event
* Create Event
* Edit Event
* Delete Event
* Display Catalogue
* Display specific project
* Create project
* Edit project
* Delete project

#### 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 | 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-23 | 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-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-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 | 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-37 | 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 |

#### 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: Use Cases:

#### Considering the in [1.1 defined functional requirements](#_A_summarization_of) the user is left with (but not exclusively) the following use cases:

|  |  |  |
| --- | --- | --- |
| ID | Use Case | Description |
| UC-1 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 1.4: User Stories:

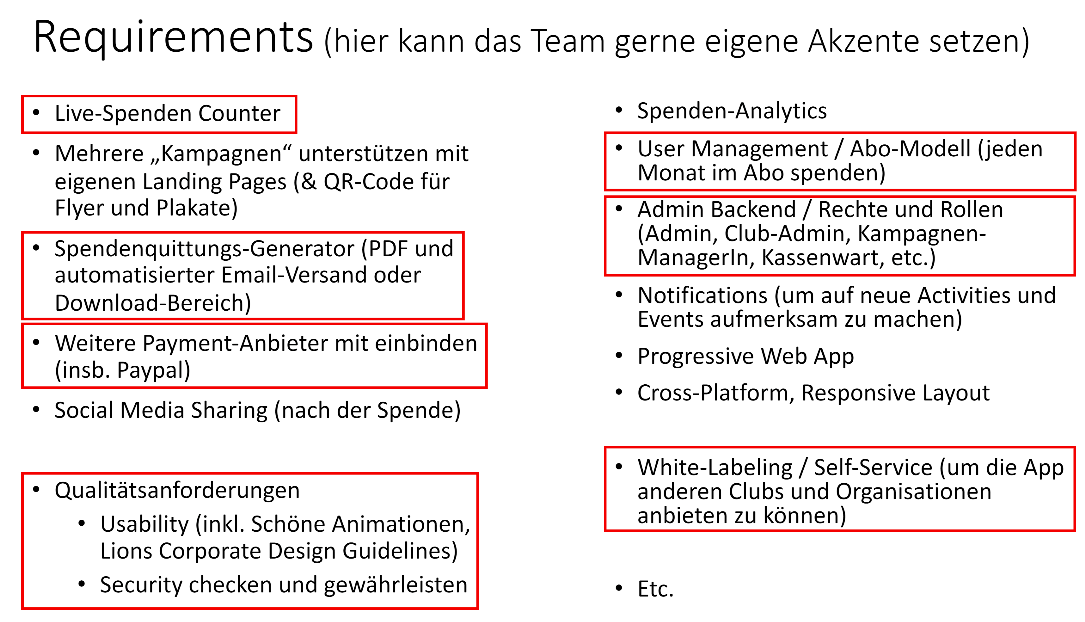
#### The core of each app is to focus on the main functionality at first, so that an application, having the core features implemented, can be used already. To best fulfill this approach, we’ll focus on the most important Use-Cases and User-Stories.

|  |  |  |  |
| --- | --- | --- | --- |
| ID | User-Story | Function | Reference |
| US-1 | As a User I want to log in to access my profile. | 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 | Change User Data | F-4 |
| US-4 | As a Guest I want to be able to continue donating without having an account | 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. | 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. | Donate using PayPal | F-11 |
| US-11 |  |  |  |

# 

# 2. App Planning

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



#### 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 once again. For this we used the previously designed Brainstorming board that all of the groups created before:



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.

[Insert our Brainstorming-Whiteboard]

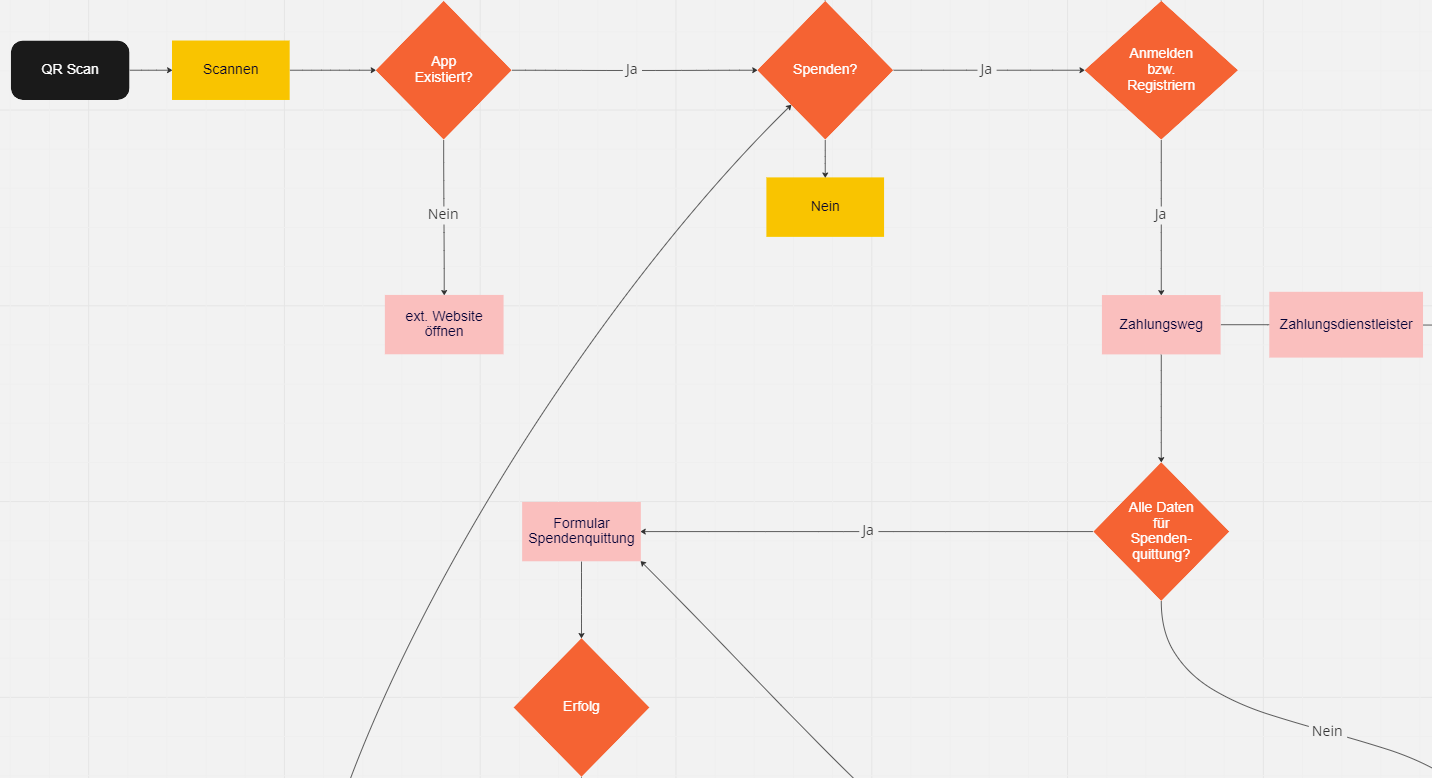
We then used those Ideas and compared them to the ones that were previously gathered by all the groups, adding a few of those ideas to our final realization, before starting to think about functionalities we wanted in that app.

In doing so, we realized quickly, that we would need a Database to store all of 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 an happy path of that process in Miro that looked something like this:



#### 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=/)

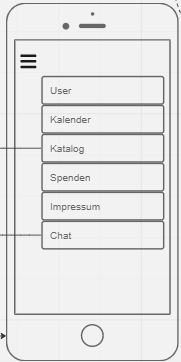
This implied, that a guest wouldn’t be required to input too much data, or even create an account in order to be able to donate, so we wanted to visually display the simplicity of the process.

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.

We continued to create wireframes for each of the required Screens in the process, including Buttons, Textfields and first Icons that would require functions behind it in the respective screens lateron.

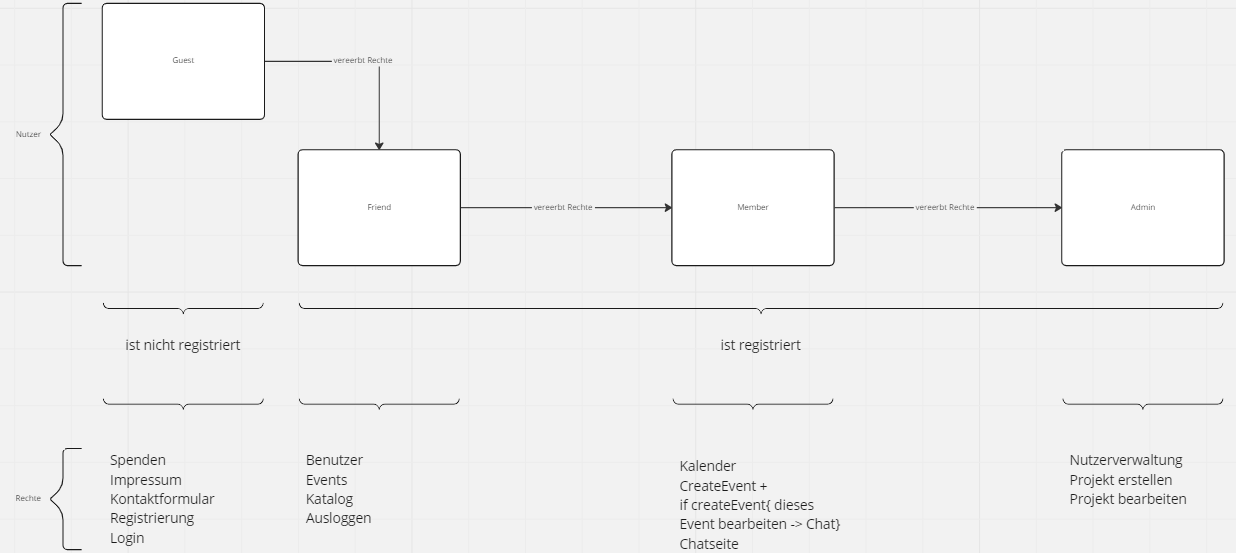
In defining and using our Burgermenu we required an united understanding of what the application would have as Screens. So, we started thinking about what we would want in that.

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.

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:



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 aswell:

Ein Bild, das Text, Whiteboard enthält.

Automatisch generierte Beschreibung

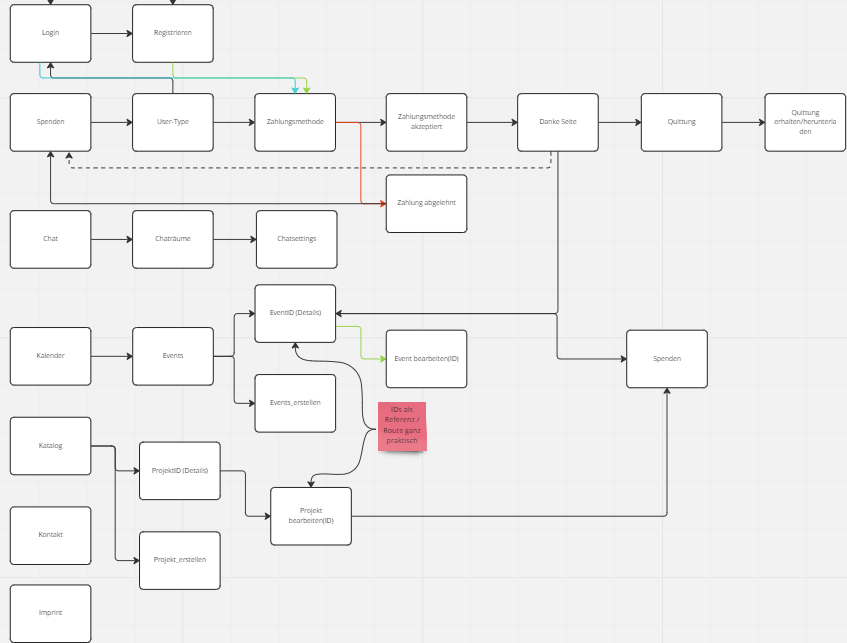
We’d have an Admin create a project with certain attributes (projectname, description, further details, aswell as a unique identifier). This project would then be stored in the Firebase storage.

Members can create an Event customizing the date of the event (startingdate aswell as ending date), define an description, select a purpose ( linked to the created projects) and an optional goal of the donation, aswell 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:



#### 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.

# 3. App Development

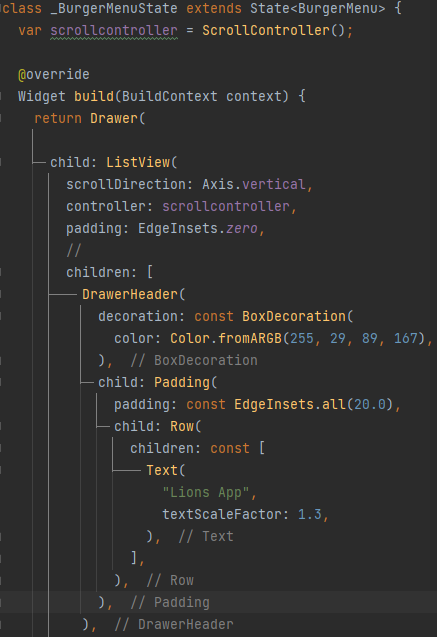
We started off with an empty Flutter-project and begun by creating a few very simplistic screens.

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.

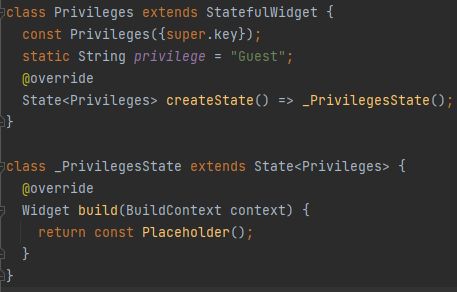
Automatisch generierte BeschreibungThe Second type of ListTiles are the ones handling navigation in the BurgerMenu. While the header tiles were just there for organizational purposes, these ones actually have functionality attached to them. First we define, which role sees the specific ListTile (not 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.

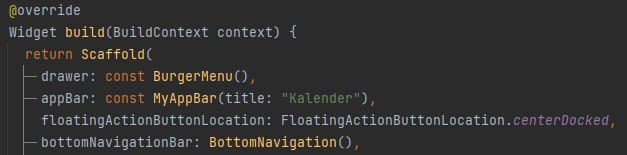
Let’s look at how that class works in detail:

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:

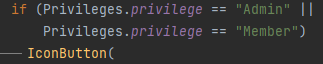


For this, we created our own class called “BottomNavigation()” which is being used on some pages.

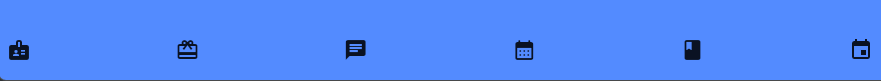
Ein Bild, das Text enthält.

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.

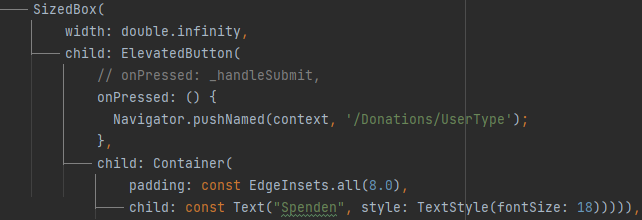
#### This is the portrayed BottomNavigation for either an Admin or an Member:



## 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:

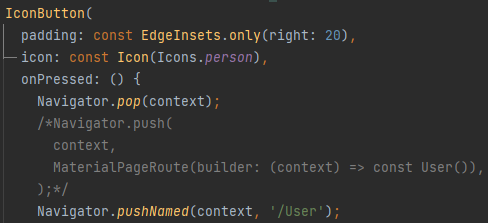


This was one of our most used buttons. It has an onPressed() function at which you can define, what actually happens if the Button is pressed, as well as an child attribute, which can be something like an Container as we used here to display a Text which will be portrayed on the Button. We use a Sized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



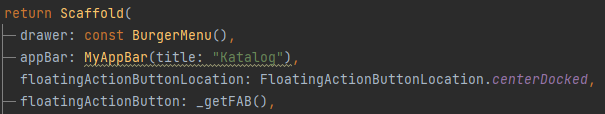
#### The following code snippet depicts an example of a textformfield:

#### 

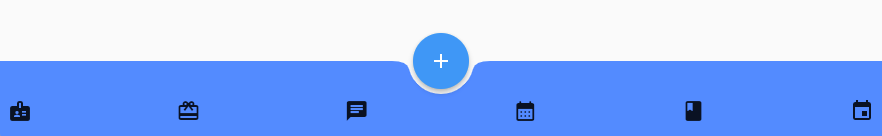
### 3.1.4: Floating Action Buttons:

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).

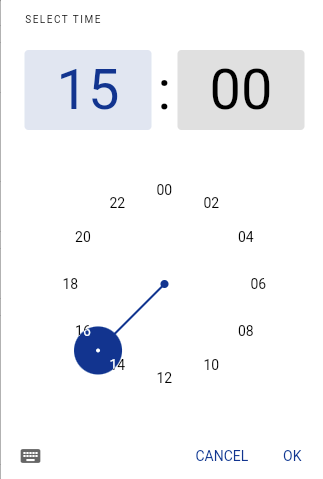
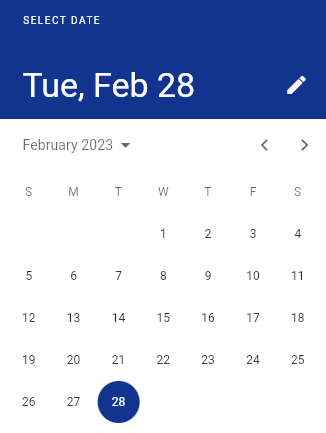
### 3.1.6: Datepicker + TimePicker:

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:

### 3.1.8: Searchbar:

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

In its general concept it is a TextField with an onChanged-function.





### 3.1.9: DropdownMenu:



## 3.2: Firebase:

# 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 aswell as an 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 Catastrophy that occurred around the world. In the project, you can read up on the catastrophy, as well as see the measurements that are being taken to help. They are listed in the Catalogue

Events: Events are datebound 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:

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

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

Burgermenu: 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.