Fridgify

Stay cool. Stay organized.

WEEK 1 – INTRODUCTION

October 6, 2019
Categories: Fridgify's Journey, Week 1

Imagine spontanously walking into the supermarket of your choice and not knowing what is inside your fridge. Now you have the following options:
– Trusting your guts and praying you buy the right things
– Driving home looking up the things you already have
– Using the new **Fridgify** app on your smartphone

Fridgify allows you to keep track of the insides of your fridge as well as their expiration date at any time and in any place. Register contents of your fridge with a single tap on iOS as well as Android.

Utilizing the power of *Flutter* as an upcoming frontend technology in combination with a *Python Django backend* we ensure the best performance, saving data inside of a *PostgreSQL Database*.

Your handy fridge organizer is split into the following projects:
1. Check-in and check-out with item scanning (dependent on image recognition)
2. User management (creating groups, managing your profile, etc.)
3. Database setup
4. REST API implementation
5. Notifications (eg. milk is empty)

Possible features:

- Recipes (recommending recipes based on the content of your fridge)
- Raspberry Pi scanner (attach it to your fridge and scan your product)

Used Tools:

Fridgify – GitHub

Fridgify – YouTrack

Fridgify – TeamCity

PyCharm

Android Studio

We, the **Fridgify Team**, are thrilled to keep you updated on our development progress. Stay tuned for the next blog.

Stay cool. Stay organized.

— Your **Fridgify Team**

WEEK 2 – ROLES AND TECHNOLOGY

October 13, 2019

Categories: Fridgify's Journey, Week 2

Tags: Roles, RUP, Technology, Week2

Each successful project has one aspect in common:
Every contributor has a definite role inside of the team.
Having distinct roles grants the opportunity to maximize productivity.
In this weeks blog, we are going to delve into role assignments as well as our technology stack.

Roles

Thinking about roles inside a team is the key for suceeding in projects. In dividing each contributor into different positions, we achieve a perfect balance of workload and productivity for each individual. Utilizing the *Rational Unified Process (RUP)* roles, defined by IBM, allows us to distribute assignments accordingly.

RUP Role	Description	Assignee
Business Desginer	Details a single set of business use cases	Joschua Goetz, Dennis Rein, Duc Vo Ngoc
Requirement Specifier	Details a single set of requirement use cases	Joschua Goetz, Dennis Rein, Duc Vo Ngoc

RUP Role	Description	Assignee
Designer – Frontend	Details the analysis and design for a single set of use cases which are frontend related	Joschua Goetz, Dennis Rein
Designer – Backend	Details the analysis and design for a single set of use cases which are backend related	Joschua Goetz, Duc Vo Ngoc
Test Designer – Frontend	Implements automated portions of the test design for the iteration concerning the frontend	Joschua Goetz, Dennis Rein
Test Designer – Backend	Implements automated portions of the test design for the iteration concerning the backend	Joschua Goetz, Dennis Rein
Blog Writer	Writes blog entries to keep people up to date	Joschua Goetz, Dennis Rein, Duc Vo Ngoc
Project Manager	Plans, tracks, and manages risk for a single iteration	Dennis Rein
Scrum Master	Is responsible for promoting and supporting Scrum	Joschua Goetz
Tool Specialist	Creates guidelines for using a specific tool	Duc Vo Ngoc
Deployment Manager	Oversees deployment for all deployment units	Joschua Goetz
Configuration Manager	Creates a deployment unit, reports on configuration status, performs audits, and so forth	Duc Vo Ngoc

RUP Role	Description	Assignee
Change Control Manager	Reviews and manages change requests	Joschua Goetz, Dennis Rein, Duc Vo Ngoc

Technology

Assigning team roles is highly reliant on knowledge and experiences of each individual team member. Due to its dependency, we have to define a proper technology stack, to ensure the eminent success of **Fridgify**.
Side Note: We were forced to move WordPress, YouTrack and TeamCity to a new Server. There were problems with our server provider. We apologize for any downtime.

Field of Development	Technologies
Backend	Python 3.7.4, unittest, Django 2.2
Frontend	Flutter 1.9 (Test Framework integrated)
Database	PostgreSQL
Project Management	YouTrack
Continiuous Integration	TeamCity
Version Control System	Git, GitHub

After assigning roles and defining our technology, we are thrilled to start conceptualizing **Fridgify** and are eager to kick off modelling and implementation phase in the near future. Stay tuned for the next blog and don’t forget:

Stay cool. Stay organized.
— Your **Fridgify Team**

WEEK 3 - SRS AND OUCD

October 20, 2019

Categories: Fridgify's Journey, Week 3

A project can only achieve astonishing success if the planning aspect is also done in a lengthily manner. As of this week we are happy to present you our Software Requirements Specification. It aims to keep the **Fridgify** team organised and focused to achieve greatness.

But the most important factor for us are our users. We want to give them all functionalities they deserve. We created an Overall Use Case Diagram to keep up to this promise.

So without further ado we present you our Software Requirements Specification and the Overall Use Case Diagram.

- Software Requirements Specification
- Overall Use Case Diagram


Overall Use Case Diagram

We are thrilled to hear your thoughts about the current state of the **Fridgify** project.

Stay cool. Stay organized.

— Your **Fridgify Team**

WEEK 4 – WIREFRAMES, MOCKUPS AND MODELLING

October 24, 2019

Categories: Fridgify's Journey, Week 4

User Interfaces

Wireframes and Mockups are crucial key points to a convenient, coherent and appealing user interface. Both of those play a huge role in designing and are necessary for a successful and smooth frontend development.
While Wireframes make up the foundation for Mock Ups and represent a first draft of the user interface, Mock Ups should display the definite concept of a user interface.

This is why we proudly present our Wireframe as well as our Mock Up.
You can find our Wireframe here.
Functionalities and correlation between views will become apparent in this Wireframe. We added comments to make the Wireframe more coherent.

On top of our Wireframe, we created a Mock Up. This Mock up should show you our final idea of how we want to visualize our User Interface. You can find our Mock Up here. (We are currently working on the Mock Up. The link will be updated as soon as possible.)

Links:

* Wireframe
* Mock Up (To be done.)

Use Cases

Designing an interface is highly dependent on required functionalities. When examining processes developers have to delve into use case creation. Use Cases help massively with understanding possible challenges and its inconvenices. Besides use cases make developing inherently easier.

To allow us effortless programming, we decided to take on and specify the following use cases:

Use Case	Description
Get Fridges	List all available fridges for a user
Create Fridge	Create a fridge, which you want
Remove Fridge	Remove a fridge from your list
Join Fridge	Join a fridge to use a fridge as a group
Get Fridge Content	Show a list of all items inside of a fridge
Add Fridge Content	Add an item to a fridge
Remove Fridge Content	Remove an item from a fridge
Change Content Volume	Change the volume (ger.: Anzahl) of an item
Login* (To be done)	Logging in and the process of getting a token

* Login is listed as use case, because our idea of a login is a bit more complex than the usual. We try to enable a Single Sign On(SSO) with the help of a token. This token is furthermore required for communication with the backend.

Other Topics

In a few weeks we are able to finally start developing **Fridgify**. Therefore we want to talk about *Database* and *API Endpoints* modelling as well.
So stay tuned for an update in the next blog regarding those topics.

We are thrilled to keep on working on **Fridgify** and are excited to present you even further progress in the future blogs. Until then…

Stay cool. Stay organized.
— Your Fridgify Team

WEEK 5 – VEGETABLES, FEATURE FILES AND MODELING

November 3, 2019

Categories: Allgemein

Testing and Feature Files

Tests are crucial to keep up the quality of every project. It is therefore an essential part of **Fridgify's** focus. With Gherkin and feature files we strive to give everyone the opportunity to understand our test cases even without prior coding skills. Due to the fact of us using **Python**, we default to *Behave* as Testing Framework instead of Cucumber (or other vegetable frameworks).
Now have a look at some of our .feature-files in our use cases:

- Use Case: Get Fridge Content
- Use Case: Add Content
- Use Case: Remove Content

More feature-files will be included in our testing repertoire. We are thrilled to get tests to run in the near future!

Let's get rolling

This week marks the first big milestone for the **Fridgify** team. A new chapter has started with the end of our planning phase. The *database structure* was defined and we decided which *API Endpoints* to focus on. We are thrilled to start with the actual implementation of our ideas and want to bring first prototypes as soon as possible.

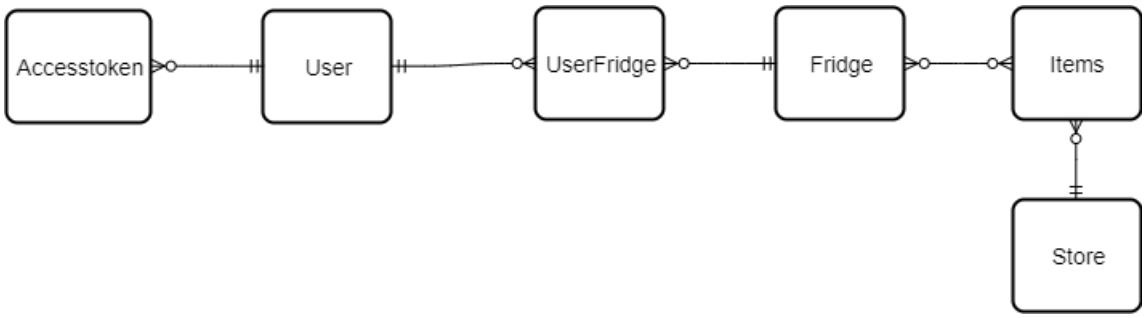
But each claim has to be backed up. So we present to you…

… our API Endpoints:

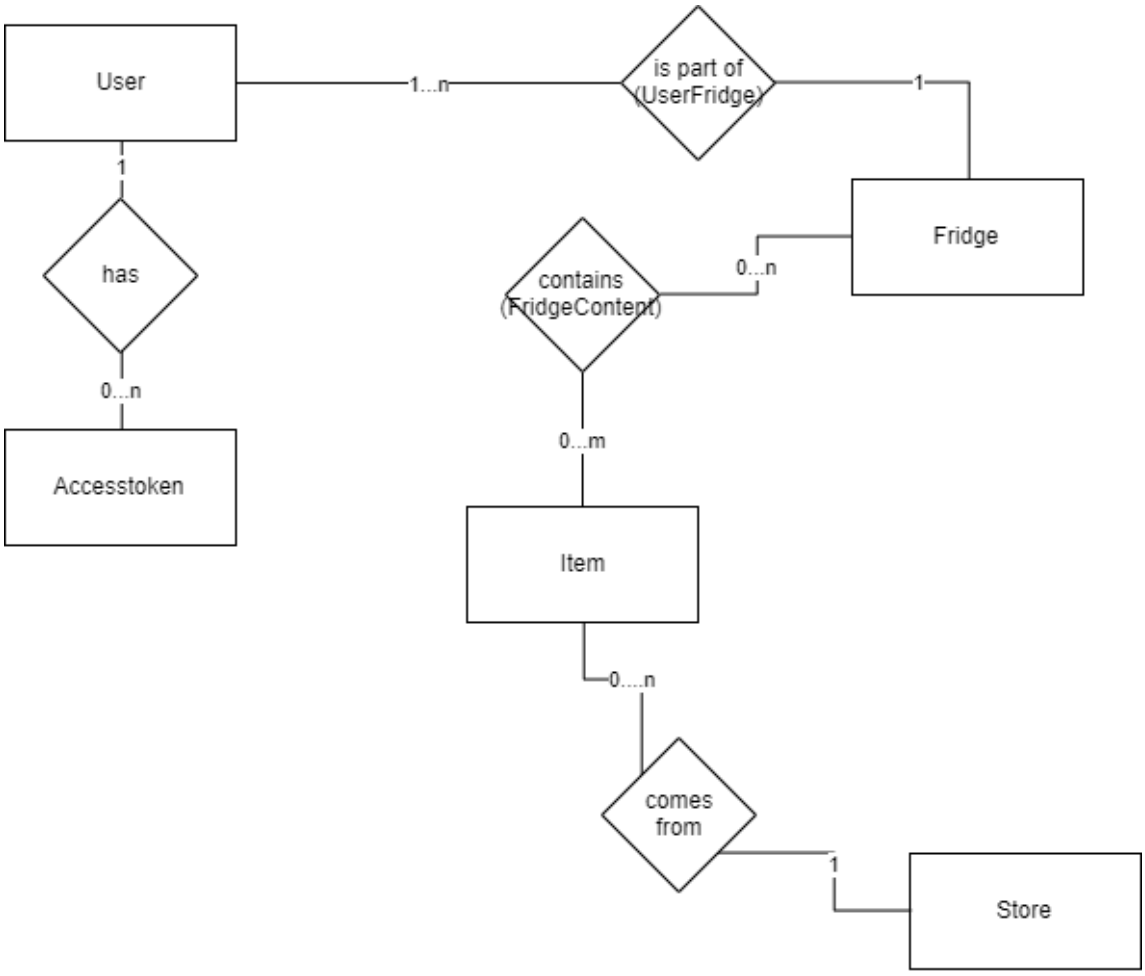
- API Endpoints – Definition

… as well as our database structure, including:

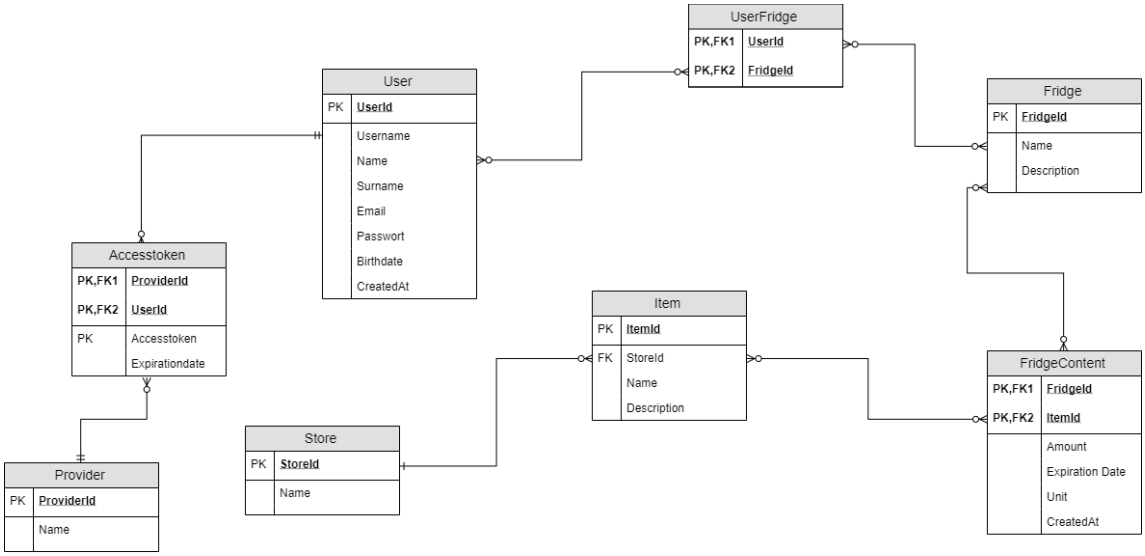
- a quick database model



- an Entity Relationship Diagram



- and our table structure



We are thrilled to hear your thoughts about the current state of the **Fridgify** project. Until then…

Stay cool. Stay organized.

— Your **Fridgify Team**

WEEK 6 – THE ART OF SCRUM

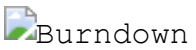
November 9, 2019

Categories: Allgemein, Fridgify's Journey, Week 6

Scrum

In today’s world of development, organizing work processes, communication and team work are a high priority. In an ever changing environment, where customer’s and user’s desires and wishes change in a blink of an eye, teams are required to act **agile**. Utilizing methods like waterfall will result in a painful experience for customers and especially developers.
Allowing developers to work more dynamically, requires them to use more modern and agile methods. Scrum is such a ‘*tool*‘, which enables developers to react to changes more appropriately.
In having short, so called, *sprints*, developers are required to think about upcoming challenges more frequently. This enables a team to react to changes more quickly and effortlessly.
Having Joschua in charge as our Scrum Master, who oversees our Scrum Process, we try to be as agile and as true as possible to the principle and idea of Scrum.
To have control of our progress and our execution, we have several tools to check our current project status.

Here is a Burndown Chart for Week 5, which allows us to see our productivity in a sprint and if we were able to retain a healthy productivity:


Burndown Chart (Week 5)

Another tool to keep track of Scrum execution is a Cumulative Flow. Unfortunately due to misconfiguration on our side, we currently cannot provide a fancy chart. Please excuse the inconvenience and be patient with us.

Every Scrum Project should have multiple Agile Boards. The following link will lead to our current Agile Board (Week 6). Or have a quick overview of our current project status with our sophisticated dashboard.

Further Progress

We have finally started development. As you can take from our issues, we try to implement authentication and define our endpoints until next week.
In our frontend the login screen shall be designed.
We are looking forward to even further progress in the next week.
Hopefully you are as thrilled as we are. Until then…

Stay cool. Stay organized.
— Your **Fridgify Team**

WEEK 7 – THE ART OF SCRUM II

November 17, 2019
Categories: Allgemein

Retrospective

After last week's introduction into the world of Scrum, we used this week to learn about looking back and learning from our past with the help of **Retrospectives**.
The **Retrospective** plays an important role in the world of Agile Software development, by answering the following questions regularly:
- What worked well for us?
- What did not work well for us?
- What actions can we take to improve our process going forward?

We spent this Monday answering these questions with the help of a professional scrum master. But before we look at the answers, we opened the day with a small game.

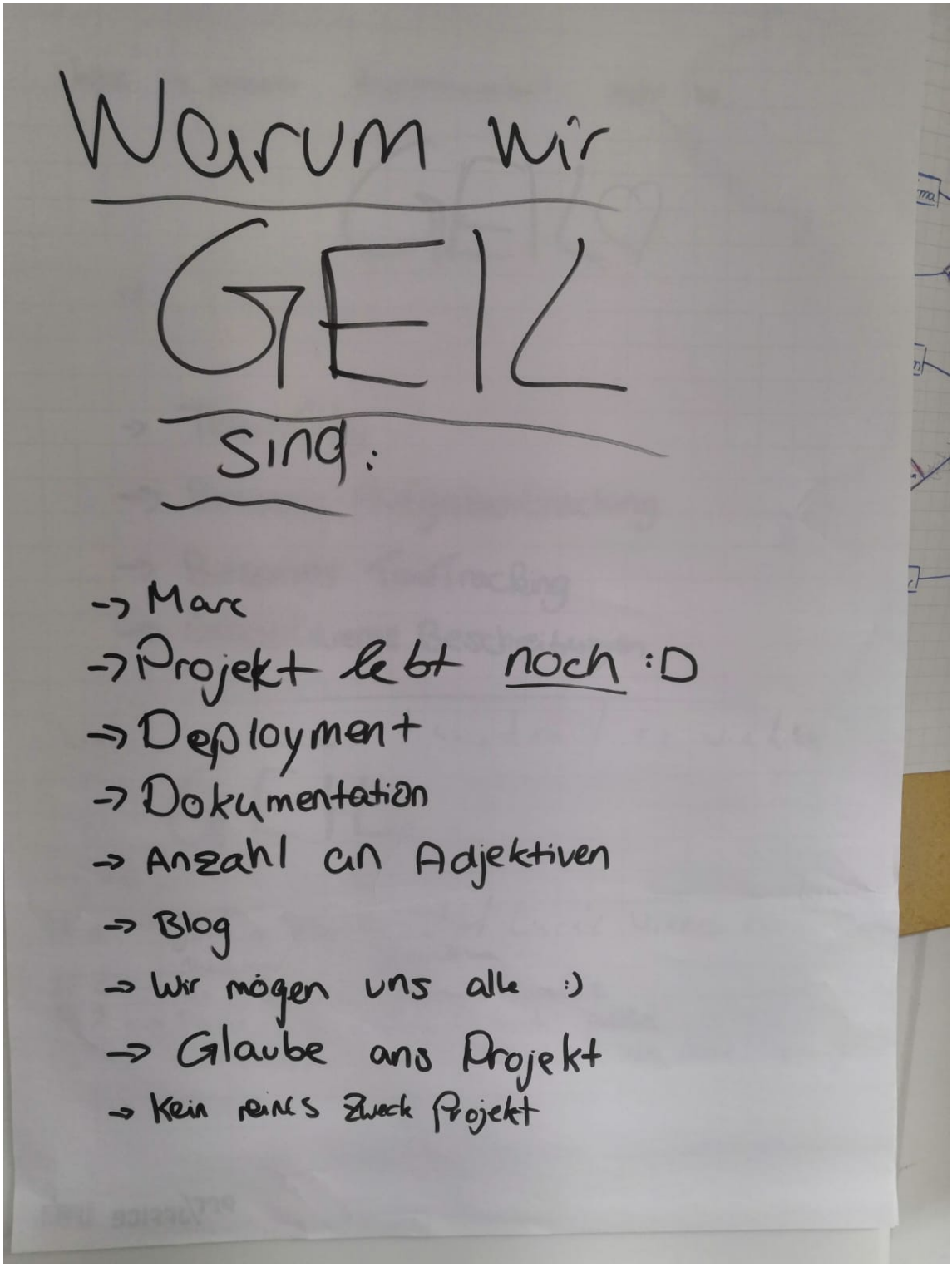
Opening Game

To get a better understanding of the current position and state in our team we had to draw ourselves as a vehicle that represents us the best.

- Duc went with a cargo ship, representing his enormous amount of ideas, which might take a while to arrive.
- Joschua saws himself more like a car with various malfunctions. As soon as the small problems are fixed he can take off with his ideas.
- Dennis draws a rocket, just as a rocket he might need a couple tries for a successful launch, but as soon as the rocket is in the air everything goes way faster.

What worked well for us?

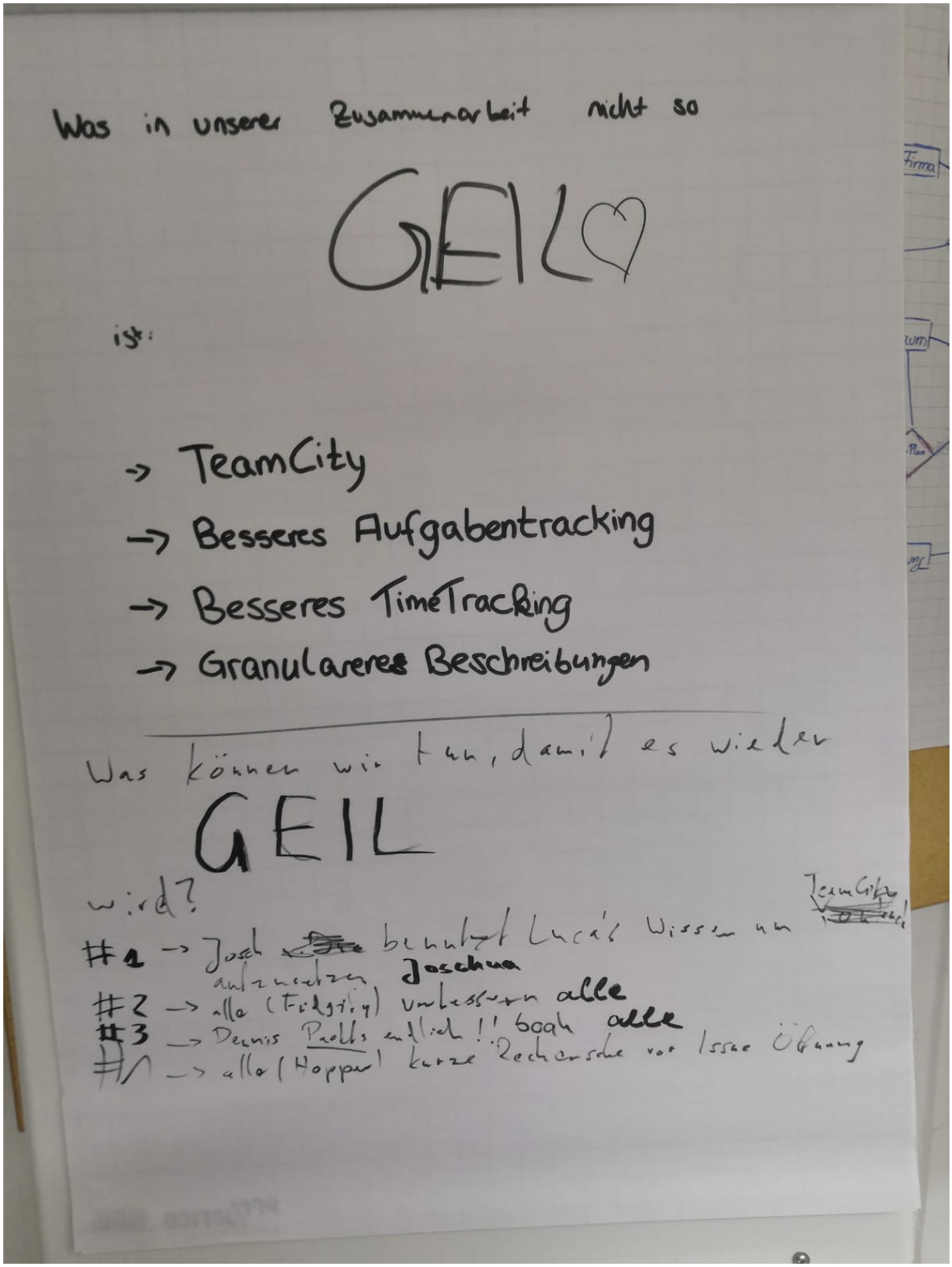
After we were done displaying our artistry, we took a look at the first question. With the help of the Hopper Team, we tried to answer this question for us.



In summary, we can say, multiple points are working well for us. But most importantly, us believing in- and liking the project as well as us, the team, plays the biggest role here.

What did not work well for us & how can we improve it?

Next, we collected the negative parts as well as ways to improve it.



As you can see we still had a couple of problems with the given technology, which were fixed throughout this week.

We are looking forward to even further progress in the next week.
Hopefully, you are as thrilled as we are. Until then…

Stay cool. Stay organized.

— Your **Fridgify Team**

WEEK 8 – CLASS DIAGRAM AND DATABASE STRUCTURE

November 22, 2019

Categories: Allgemein

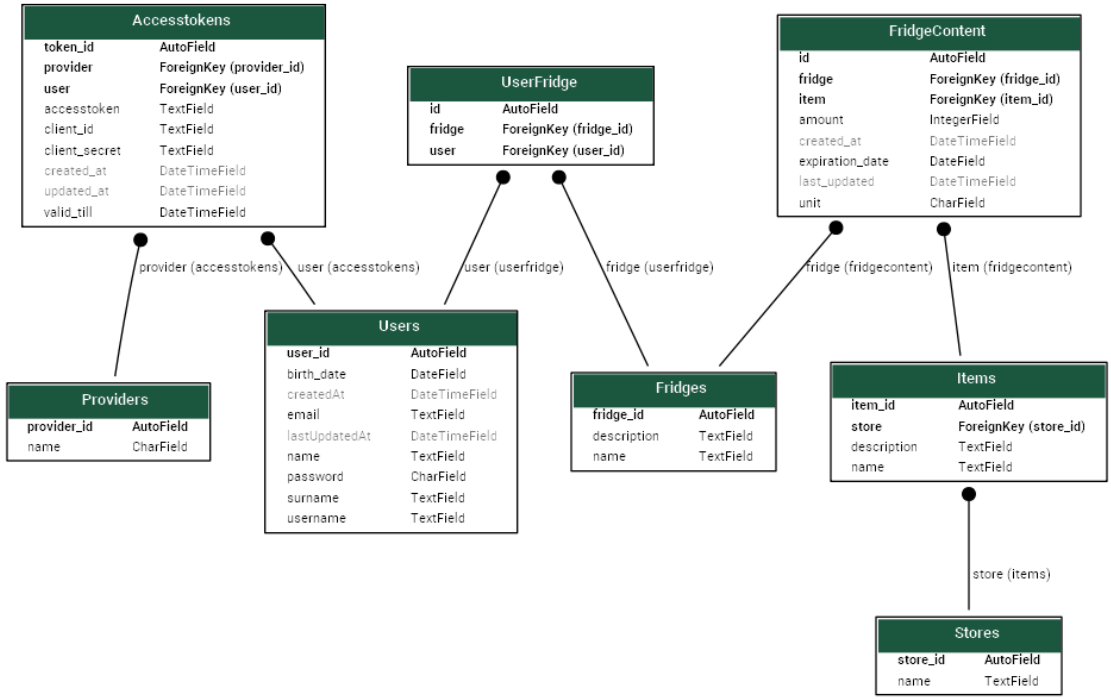
As development keeps on going we want to show you our current class diagrams and database structure. The latter was already mentioned in a previous blog post but we will be further elaborating on that. Our project is structured in backend and frontend, each with their respective technology stack.

Backend

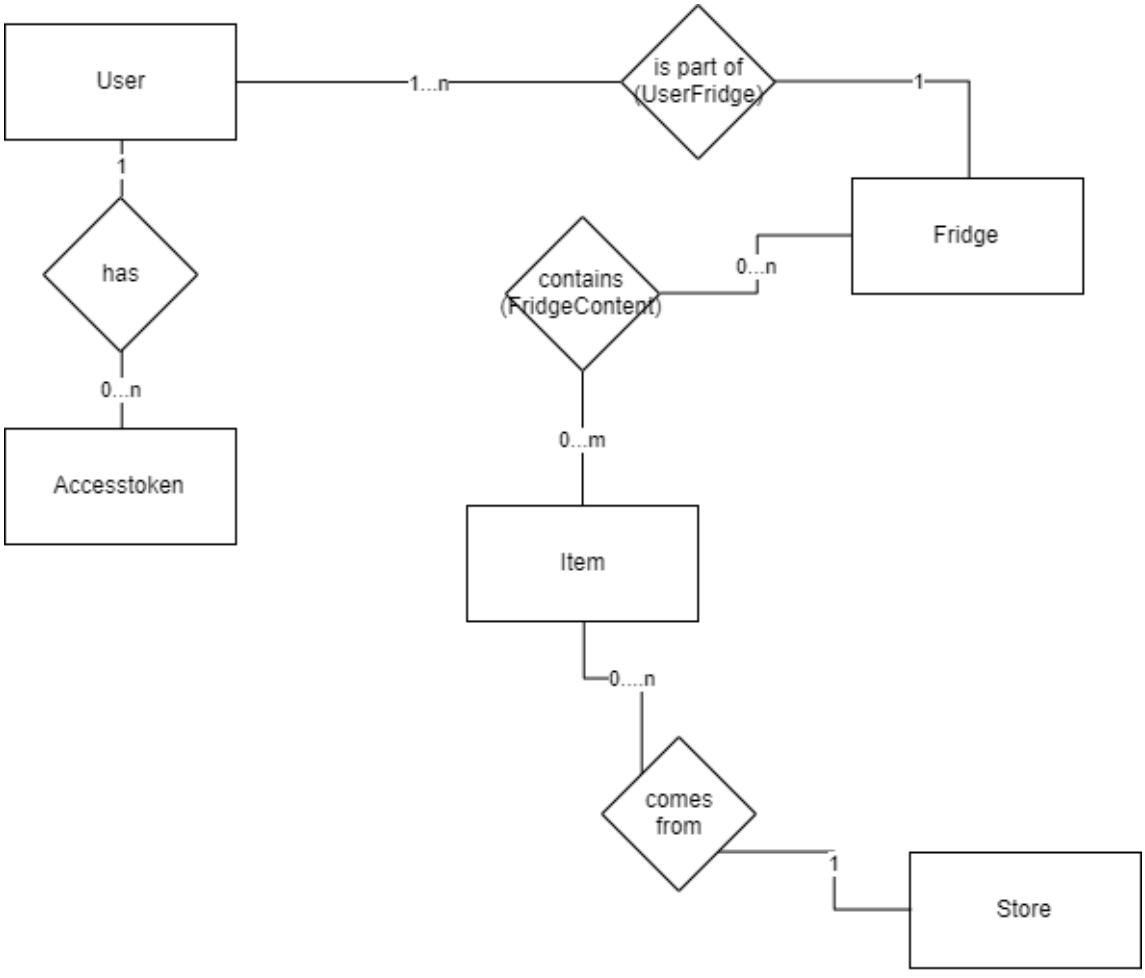
We will start with our backend. We use Python with Django. Each request send to our backend get’s routed to their dedicated controller. These controllers have entrypoints processing requests. Python doesn’t force us to use classes and our backend structure isn´t enforcing it either. We cannot generate a class diagramm because of that. Django however creates a class for each table in our database to access them and gives us the opportunity to create a UML diagram from that.

We take that opportunity and show you our previously constructed ER diagram and of course the generated class diagram.

Generated Class Diagram



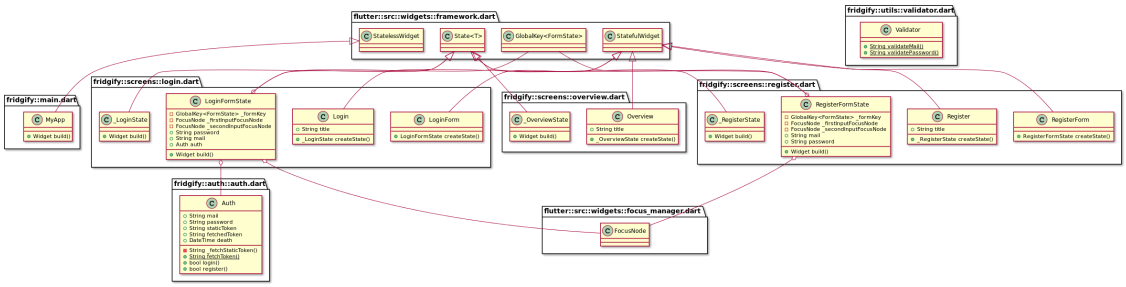
ER Diagram



Frontend

The frontend with flutter wasn't such a hassel and a small plugin helped us creating a class diagram.

Frontend Class Diagram



We are thrilled to hear your thoughts about the current state of the **Fridgify** project.

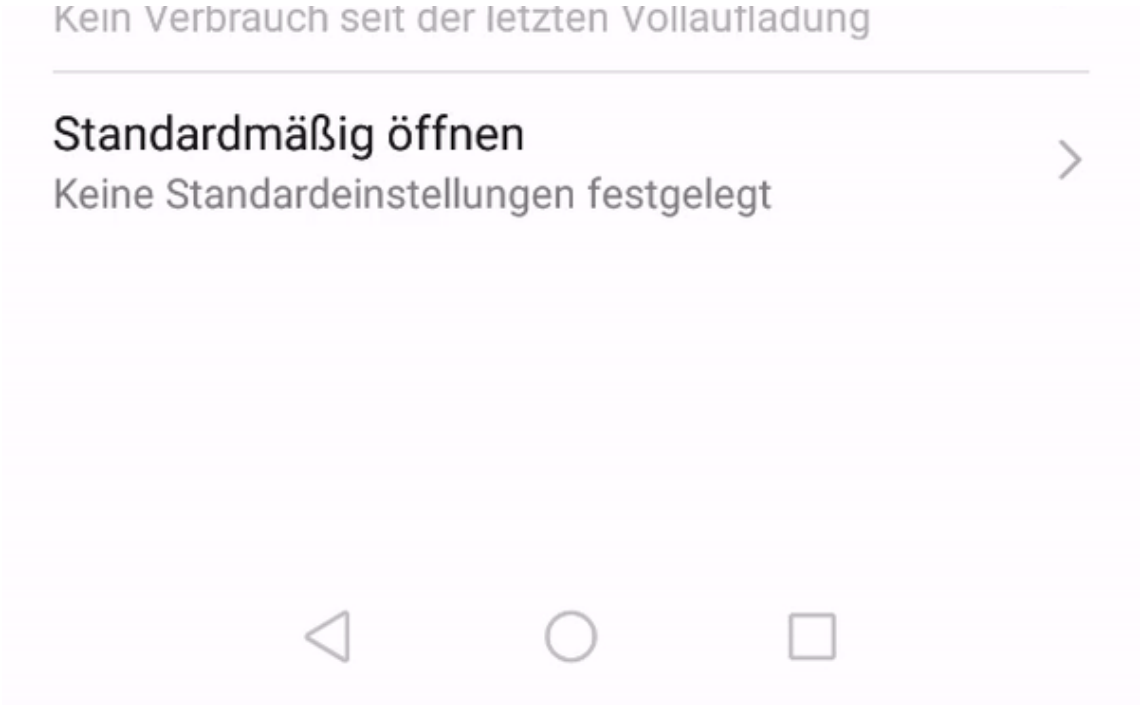
Stay cool. Stay organized.

— Your **Fridgify Team**

WEEK 9 – ARCHITECTURE

November 29, 2019

Categories: Allgemein, Fridgify's Journey



A short disclaimer: This is going to be a pretty long one. So buckle up, take a seat, get some tea and enjoy!

MVC

Now first of all, let's talk about the MVC, short for **M**odel-**V**iew-**C**ontroller as software design pattern. MVC is a popular pattern, due to its easy to understand nature.

Now what exactly happens in MVC?

Let's say a user wants to use our MVC based application. First of, he opens up the app and sees an UI. Every User Interface, everything related to the interface, can be classified as *Views*. Views are in short a visualisation of data and controls.

Now apart from a view, an application deals with lots and lots of data. Data is most of the time stored in databases. To represent data, we use so called *Models*.

To use those models, e.g. adding, updating, showing or deleting data, MVC has *Controllers*. Controllers process data and return it back to the view, where it is shown.

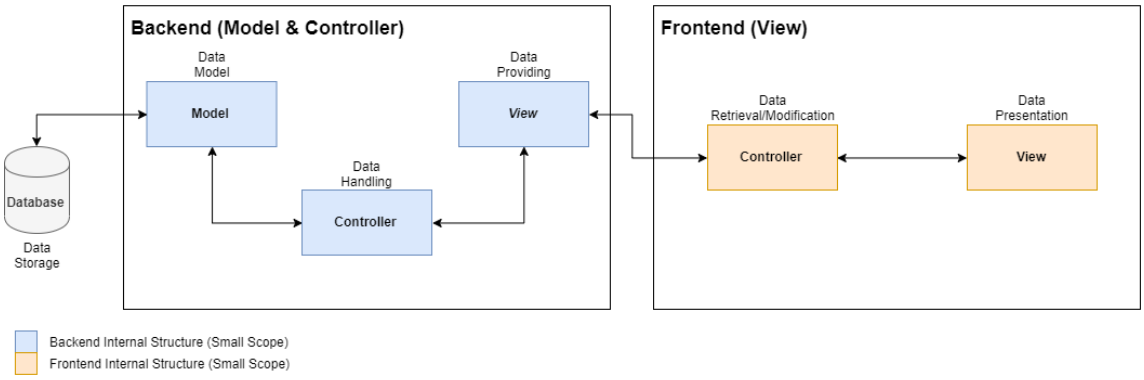
Django as a Framework

Django is a web framework for python, which implements MVC. In Django you have urls, views and models. When making a request, the first interaction point are the urls. Urls are the entry point of each request, directing a request to the correct view.
Views are the actual representation of what the user sees. In Fridgify's case, the views return JSON Data.
Models represent data saved in the database.
Django provides a very nice functionality regarding models: After creating models, database tables are created and updated on runtime accordingly. This allows us developers to not even write a single line of SQL, which is pretty neat.
Now as you may have noticed, we do not have any *controllers*. Data handling can be done in the views, which is not the most clean way to do. This is why we use *handlers*, which shall take over data processing. These handlers are being called by our views and allow us to avoid code redundancy.

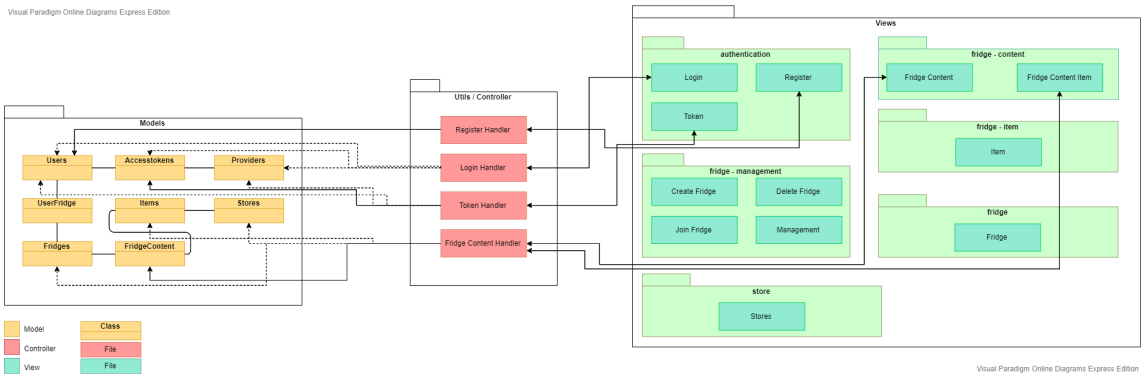
Fridgify's Code Structure

Now why did we just explain a basic concept of programming as well as giving a short overview of a framework?
Well, it is time to explain our idea behind the MVC structure of Fridgify.
As you may know we have two different 'applications': our backend providing an API and our mobile frontend.
While each application has its own structure, we tried to design our software architecture in such a way, that it even complies to MVC in the big scope as well as in the

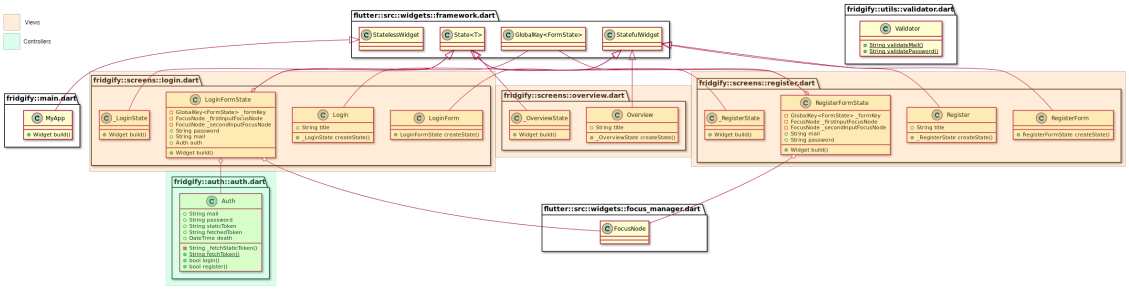
smaller scopes.



Diving deeper into our smaller scopes, you will see the following MVC structure in the backend:



And the following structure, containing only views and controllers, which make requests and control user flows:



As you may see the frontend lacks the *Model* part, because in our bigger scope the model part is provided by the backend. The frontend does not store huge chunks of data, so there is no need for models.

Server/Client Architecture

Now after talking about about code architecture, let's talk about our server/client architecture.

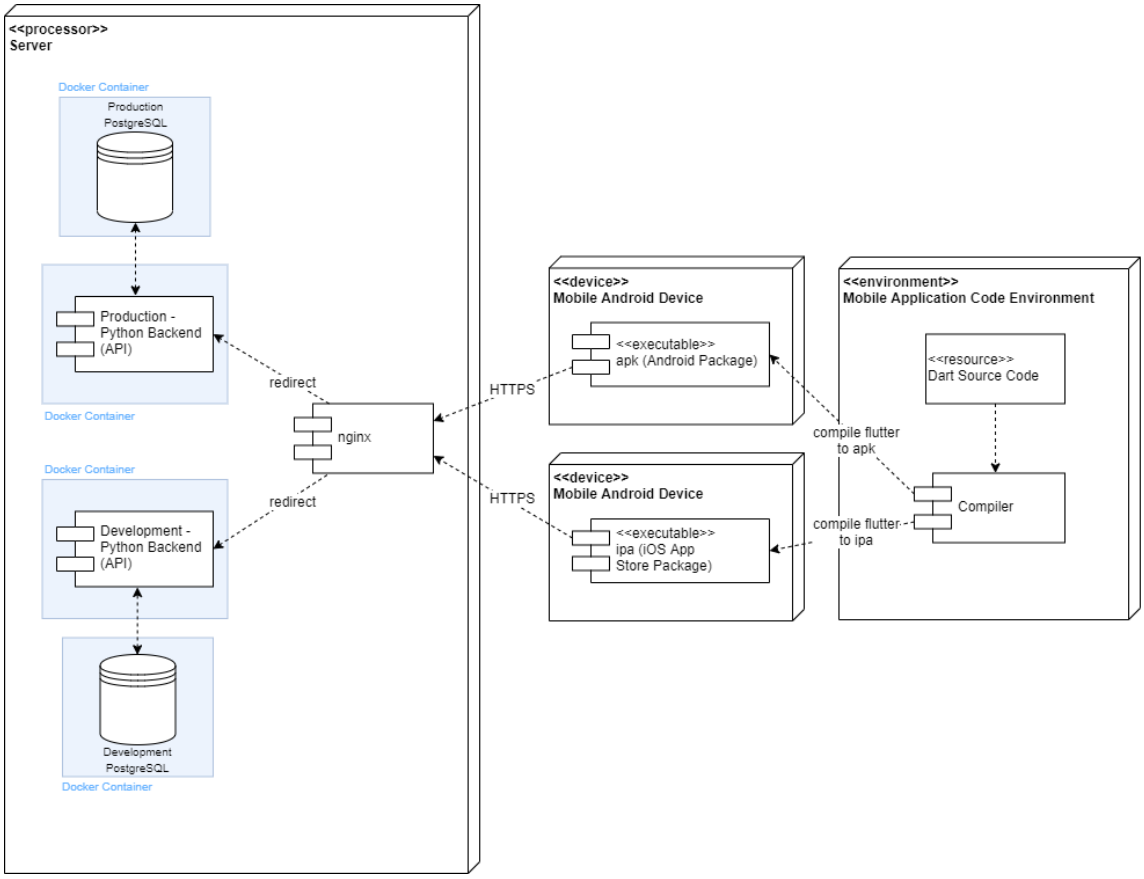
To deploy our backend, we make use of **Docker**. We have two docker containers, one contains our database while the

other one runs our Python application. The application is accessible with **nginx** as our reverse proxy.

Our CI detects changes on our develop and master branch. The develop branch is being tested and taken live for the development API. While a change in our master branch is taken into a new Docker Image and is then taken live on our production server.

Currently the frontend application is compiled manually. This process shall be automated, to ensure the highest quality.

Take a look at our Server/Client Architecture:



Cucumber

Cucumber is up and running. Have a look at our test.

[Fridgify Cucumber Test]

Further Progess

We are almost finished here. Just a few words on our progress:
We have implemented all our specified Use Cases in the backend, which shall be finished till the mid-term. There are still some little tweaks needed, but we are overall very happy with our current status.
Now what's left to do is finish the Use Cases in the frontend part.
We are absolutely thrilled to show you in the next weeks blog our state of development.

Now, here are some last few remarks regarding this blogs content. If you are even more interested in our architecture have a look at our SAD.

We are excited to hear your feedback. Until then…

Stay cool. Stay organized.
— Your **Fridgify Team**

WEEK 11 (YEAH, WE KINDA SKIPPED 10) - SUMMARY

December 13, 2019

Categories: Allgemein, Fridgify's Journey

This few months had been a joyous ride. From humble beginnings to a small prototype, this adventure until now was a blast.
Due to the semester ending soon, we will take a step back with posting weekly blogs. There may still be some updates for you, trying to keep you up to date on our development progress.

If you want to follow our progress even without any blog posts, make sure to have a look at our **important links** list:

* **Update:** Documentation
* YouTrack – Fridgify
* TeamCity – Fridgify
* GitHub – Fridgify
* GitHub – Fridgify Frontend
* GitHub – Dennis Branch
* GitHub – Working Feature Files
* GitHub – Fridgify Backend
* GitHub – Ducs Branch
* GitHub – Joschuas Branch
* Docker Hub – Fridgify
* Fridgify's SRS
* Overall Use Case Diagram
* Fridgify's SAD
* Midterm Presentation
* Burndown
* Gant-Chart
* Cumulative Flow

And if you want to reminisce about our progress, here is an ordered list of our blog posts:

1. Week 1 – Introduction
2. Week 2 – Roles and Technology
3. Week 3 – SRS and OUCD
4. Week 4 – Wireframes, Mockups and Modelling
5. Week 5 – Vegetables, Feature Files and Modeling
6. Week 6 – The Art of Scrum
7. Week 7 – The Art of Scrum II
8. Week 8 – Class Diagram and Database Structure
9. Week 9 – Architecture
10. This Summary 😉

Update:

If you want to have a quick overview of all important documents, do not hesitate to visit our new ***Documentation Repository***.

We are happy you were with us on this journey until now and are grateful for every comment you made.
Now, to wrap everything up, we want to wish all of you happy holidays and a healthy start into the new year.
Hope to see you soon in the next blogs in the new year.
Until then…

Stay cool. Stay organized.

— Your **Fridgify Team**

PHASE 2 WEEK 1 – THE FRIDGE COOLS AGAIN

April 25, 2020

Categories: Allgemein

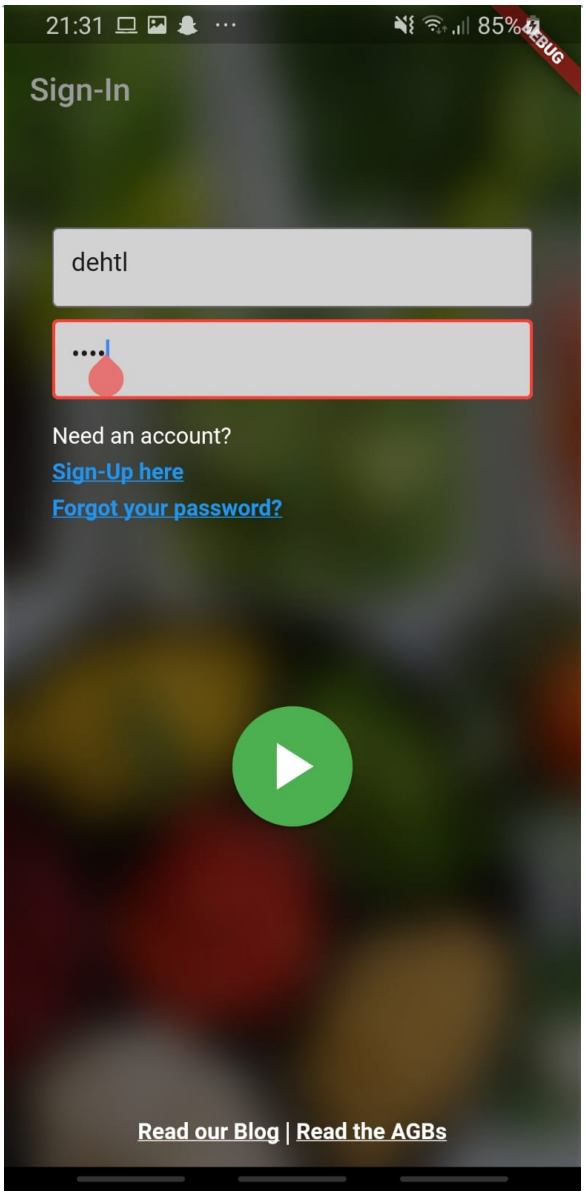
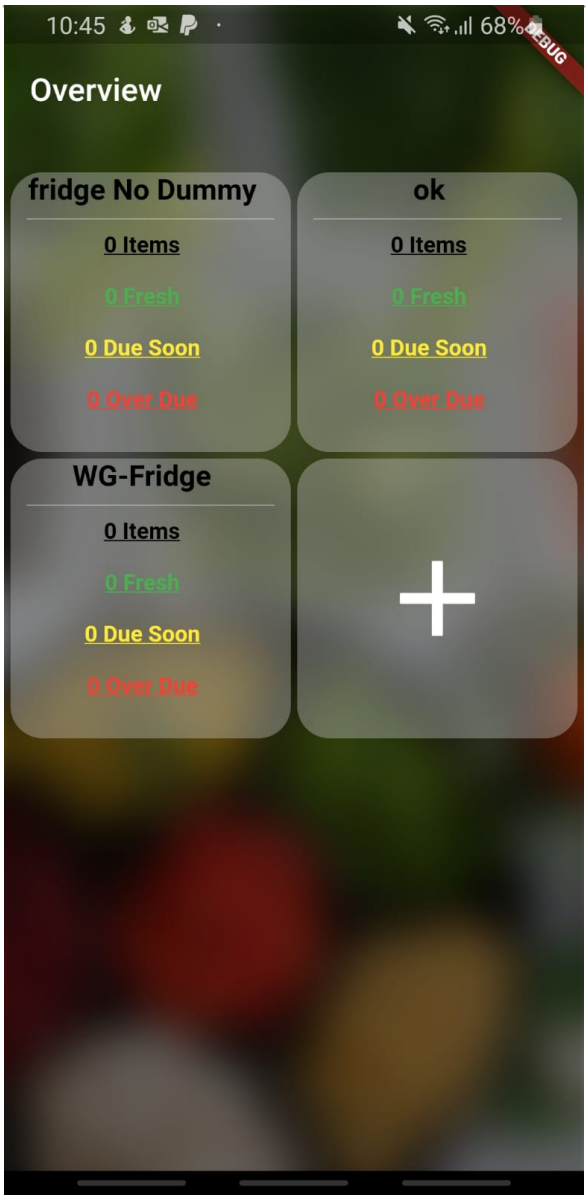
Hello everyone, we, the Fridgify, welcome you all back to our weekly blog entries. we hope everyone here had a tremendous practical phase, Eastern and Christmas break. Today we are eager to talk about what has happened and what is going to occur during the second semester of SE. So lay back and enjoy this entry.

What did happen?

During this long break, we, the Fridgify Team, had much time to think about our current application, about what we did so far, and about what other people could think about it, and concluded that our frontend is inefficient, outdated and does not look as good. The frontend team decided to throw the current state away and rebuild the application from scratch, but still using our mock-ups and the defined technology scope.

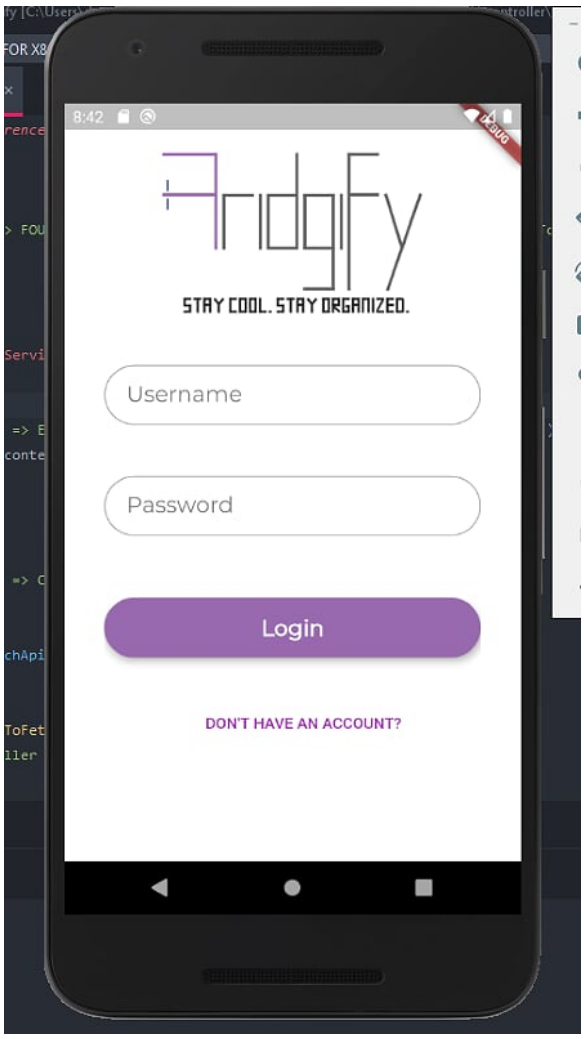
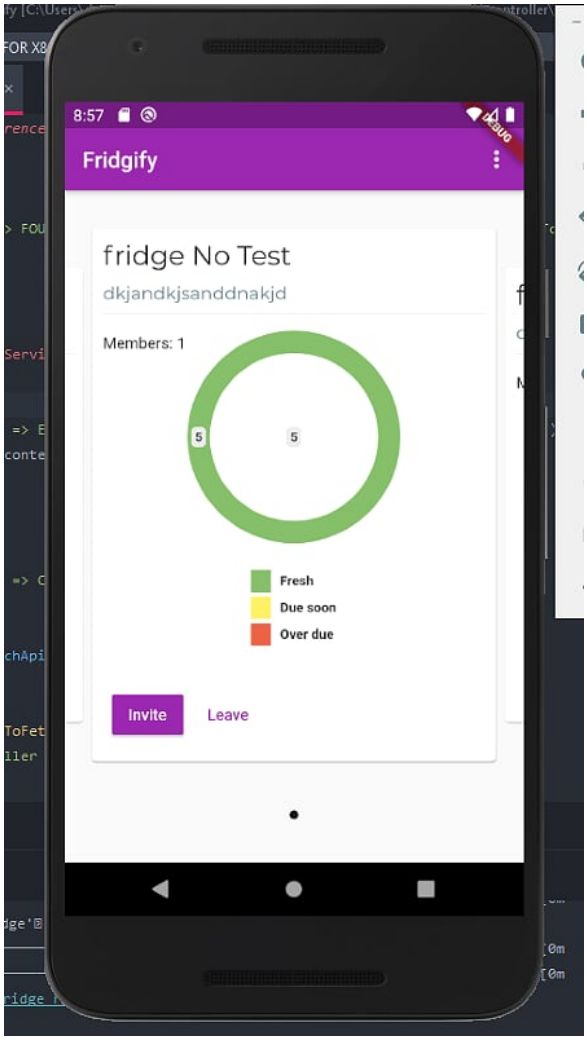
For example, we made out of this:

Login Old	Menu Old

Login Old	Menu Old
	

This:

Login New	Menu New

Login New	Menu New
	

A full demo of the new application can be found here and any kind of feedback about the new design is welcome.

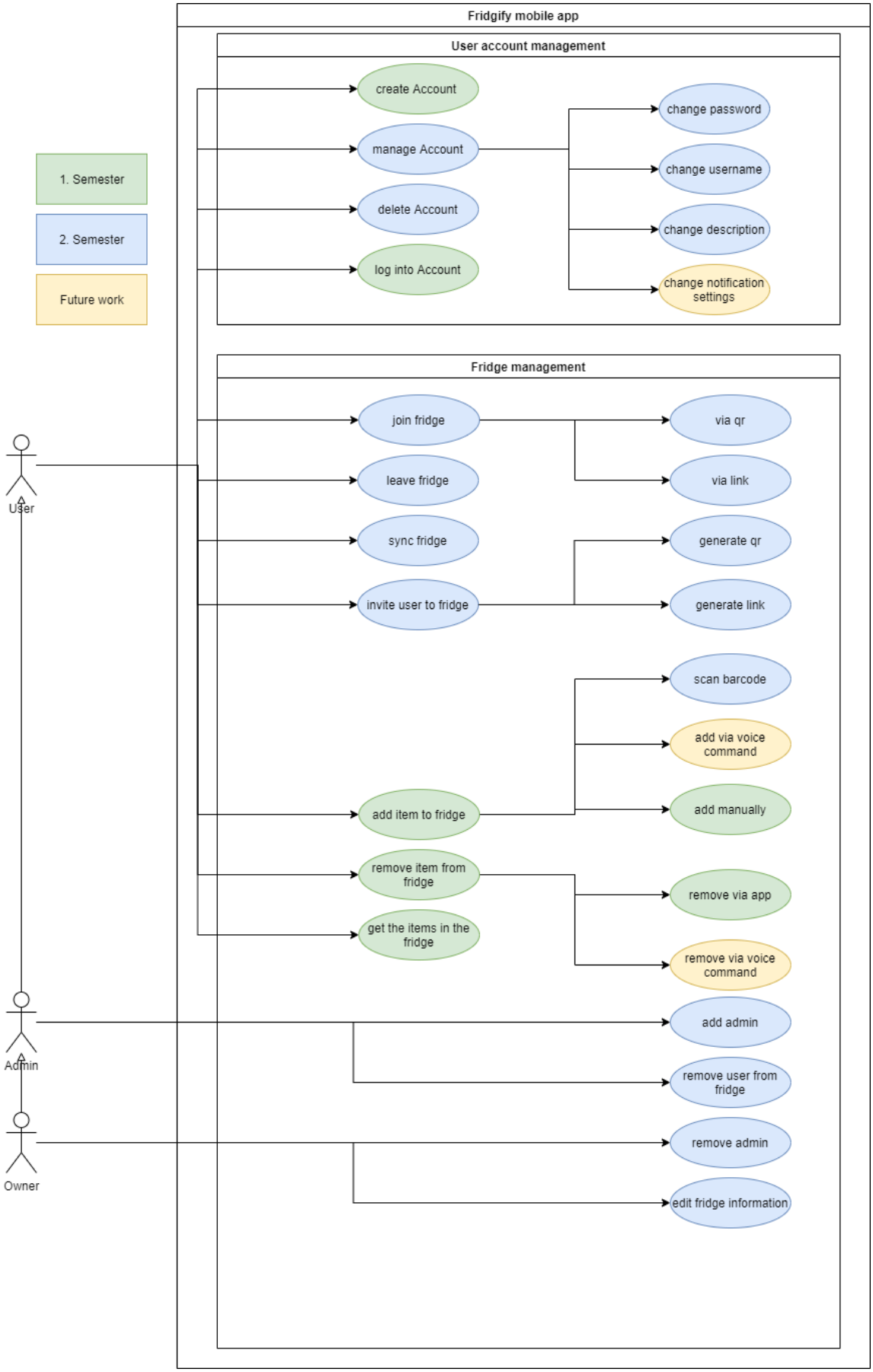
Also we created an overview over the time spent for the use-cases of last semester. Take a look here:

UC	Documentation	Coding	Testing	Warm-Up time	Total	FP
create Account	10	25	10	7	52	
log into account	12	20	10	7	49	
add item to fridge	2	3	2	5	12	

UC	Documentation	Coding	Testing	Warm-Up time	Total	FP
remove item from fridge	2	3	2	3	10	
get the items in the fridge	2	6	4	3	15	

What will happen?

Our goal of the second phase is, of course, finishing the leftover use-cases of last semester as you can see here:



We plan to:

- Add user management, to edit and delete your accounts
- Add fridge sharing, to create QR-codes and links with which other people can join your fridge

- Add a role system, to add admins and manage user in your fridge
- Add a barcode scanner, to easily add new items to a fridge

As you can see, there is quite much to do for us during the next couple of weeks, but we are eager to keep you updated.

Just as for the already finished use cases, we created an "estimated time" table for the new use cases:

UC	Documentation est.	Coding est.	Testing est.	Warm-Up time est.	Total est.	FP
manage account	3	2	2	3	10	
delete account	1	2	3	4	10	
join fridge	1	2	1	4	8	
remove user from fridge	1	2	1	1	5	
sync fridge	2	2	1	4	9	
invite user to fridge	2	1	4	4	11	
add admin	4	1	2	2	9	
remove user from fridge	2	2	2	1	7	
remove admin	2	2	2	2	8	
edit fridge information	4	3	1	2	10	

If you want to read more about our use cases, feel free to visit our freshly introduced documentation repository and find there an in-depth description of the UC.

We are excited to be working on the project again as well as sharing our progress with you. Be prepared for some awesome progress reports. Until then…

Stay cool. Stay organized.

— Your **Fridgify Team**

PHASE 2 WEEK 2 - RISKY STUFF

April 26, 2020

Categories: Fridgify's Journey

Software projects fail all the time. There are multiple reasons for it, rather it is lack of funding, team internal problems or sudden events, a lot can go wrong while working on a project. To prevent nasty surprises it is important to think about possible risks, their impact as well as their probability.
That is why in this weeks edition of the blog, we decided to tackle **Risk Management.**
We thought about risk, which could possibly affect our project as well as endanger the progress of it, and compiled them into a table for you. Here is our result:

Risk Name	Risk Description	Risk Probability of Occurrence	Risk Impact	Risk Factor	Risk Mitigation	Person Charge Track:
Sudden sickness	If one of the developers gets sick and will not be able to continue developing for the project	5%	6	0,3	Constant inclusion of at least one further team member into your codebase	Everyo
Not enough time	Underestimating the time needed for specific tasks	15%	10	1,5	Keeping track of time management	Everyo

Risk Name	Risk Description	Risk Probability of Occurrence	Risk Impact	Risk Factor	Risk Mitigation	Person Charge Track:
A sudden change of software	Flutter is a beta software, it might change without any further notice and break something	50%	7	3,5	Stay uptodate on software blogs and news	Denni Josch
Docker/Server break down	Because of unpredictable issues we might encounter unexpected errors	40%	9	3,6	Always have backups of our images	Duc, Josch
Productivity issues	Procrastination and productivity of certain people might occure	33%	5	1,65	Team members try to motivate and keep an eye on each other	Every
Cyber attacks	People trying to hurt us and our environment	5%	9	0,45	Always have backups of our databases and writing of unhackable software	Every

Risk Name	Risk Description	Risk Probability of Occurrence	Risk Impact	Risk Factor	Risk Mitigation	Person Charge Track:
Dataprivacy	Sudden change of guidelines and policies	50%	1	0,5	Keep an eye on the news and specific other sources	Every

As always, we are excited to hear your feedback. Until then…

Stay cool. Stay organized.

— Your **Fridgify Team**

PHASE 2 WEEK 3 – HOW LONG WILL IT TAKE?

May 1, 2020

Categories: Allgemein

As we jumped back into action last week, we defined a few new use cases, which you have probably read in our last blog. Due to the amount of our use cases, it is important to determine their duration and complexity. This is why this weeks blog is going to tackle the topic: *Function Point Analysis*.

What is a Function Point Analysis?

Function Point Analysis (FPA) is a method of Functional Size Measurement, which was originally developed by Allan Albrecht in the late 1970s at IBM and has been further developed by the **International Function Point Users Group (IFPUG)**. It assesses the functionality delivered to its users, based on the user's external view of the functional requirements. It measures the logical view as compared to measuring the physically implemented view or the internal technical view.
FPA distinguishes between:
- Business transactions (Processes)
- Business data (Data Groups)

Performing the FPA is also referred to as a *Function Point Count* and includes the identification, classification and weighting of each these Process and Data Group components. Once we have a *Function Point Count* we can use the resultant measure of the software product to, in our case, combine it with a time estimation.

Kowalski, analysis

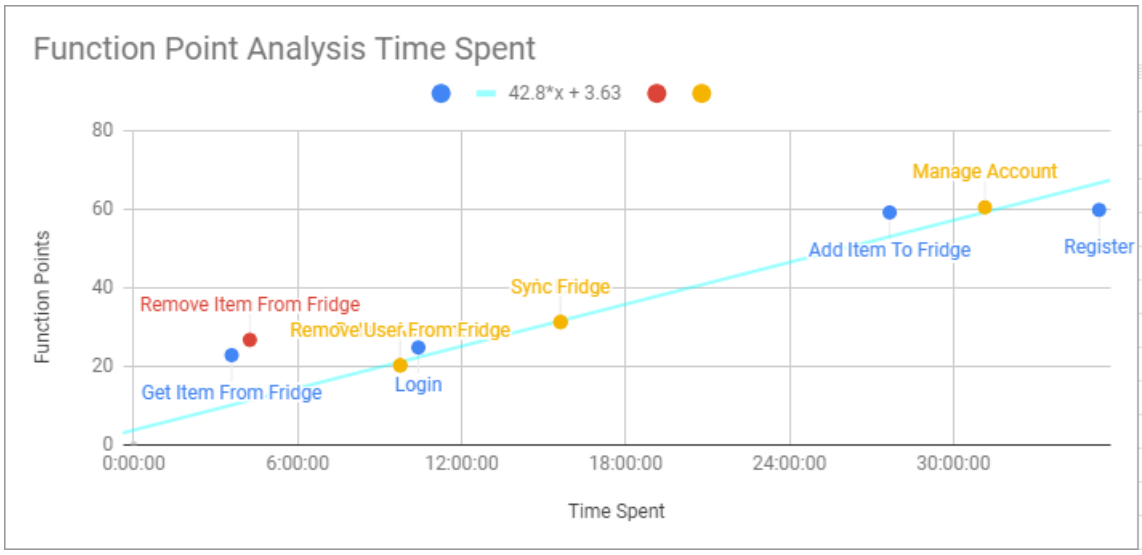
After getting the theory out of the way, let’s dive into the actual analysis part of the FPA for **Fridgify**.

We calculated our function points with the Tiny Tools.

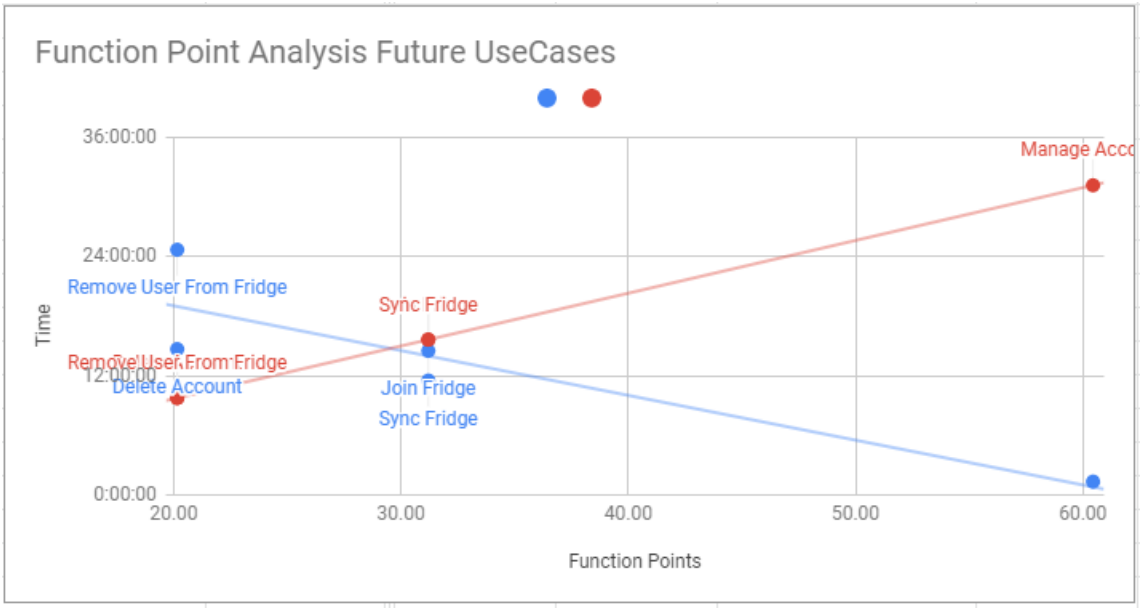
We created a FPA for our old use cases as well as for our newest additions.
If you are interested in our analysis points, you can find our analysis here.

We used the calculated function points to see how we spent our time and have an overview of our estimation.
Here is a diagram showcasing our time spent on all use

cases (estimated time for future use cases):

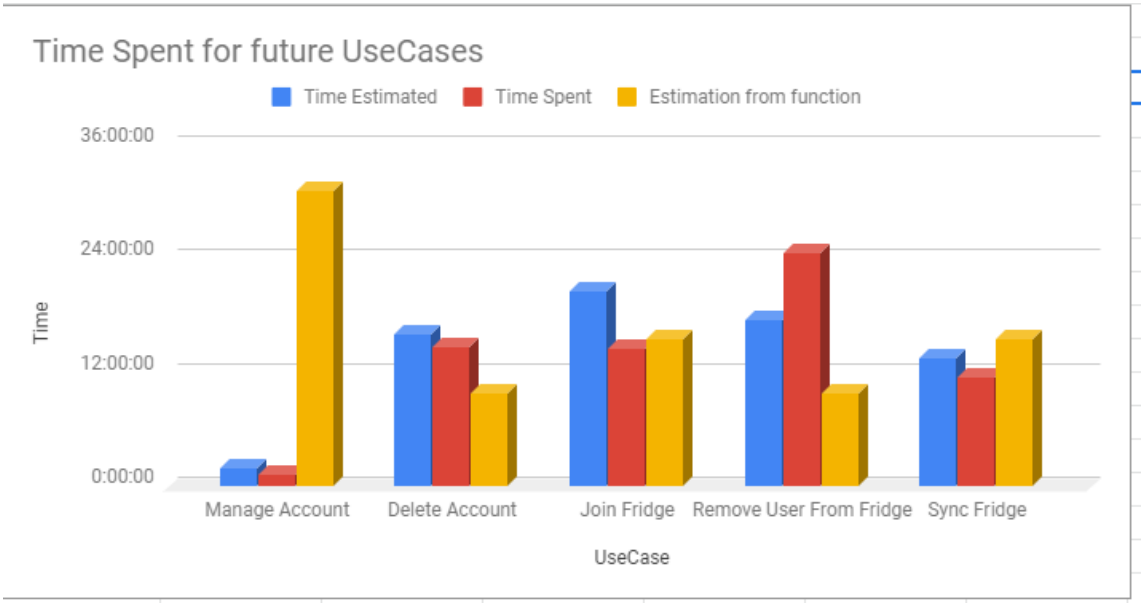


Here is a diagram depicting only our future use cases:



And finally, the following diagram shows the current progress of our future use cases compared to our estimation by our gut feeling as well as the estimation we got from

our analysis:



You can find the whole time estimation analysis here.

With this analysis, we have a better understanding of our main focus points. We are able to adjust our implementation focus accordingly due to now knowing the use cases actual estimated time.

This means we can now fully dive into the implementation phase of our new use cases and try to propel our project forward.
Do you have any thoughts on our analysis? We are thrilled to hear them. If that is not the case, we hope to see you in our next blog. Until then…

Stay cool. Stay organized.
— Your **Fridgify Team**

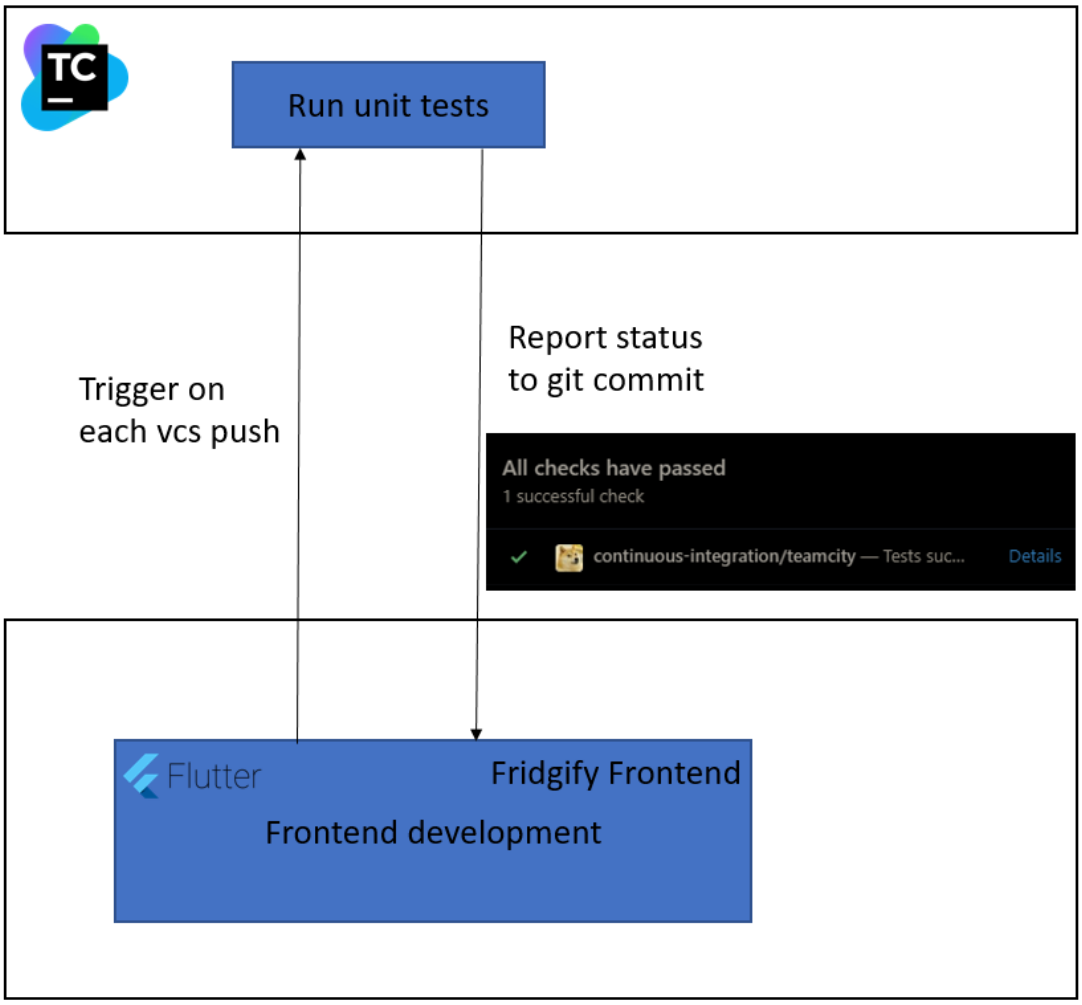
PHASE 2 WEEK 4 – TESTING AND BADGES

May 7, 2020

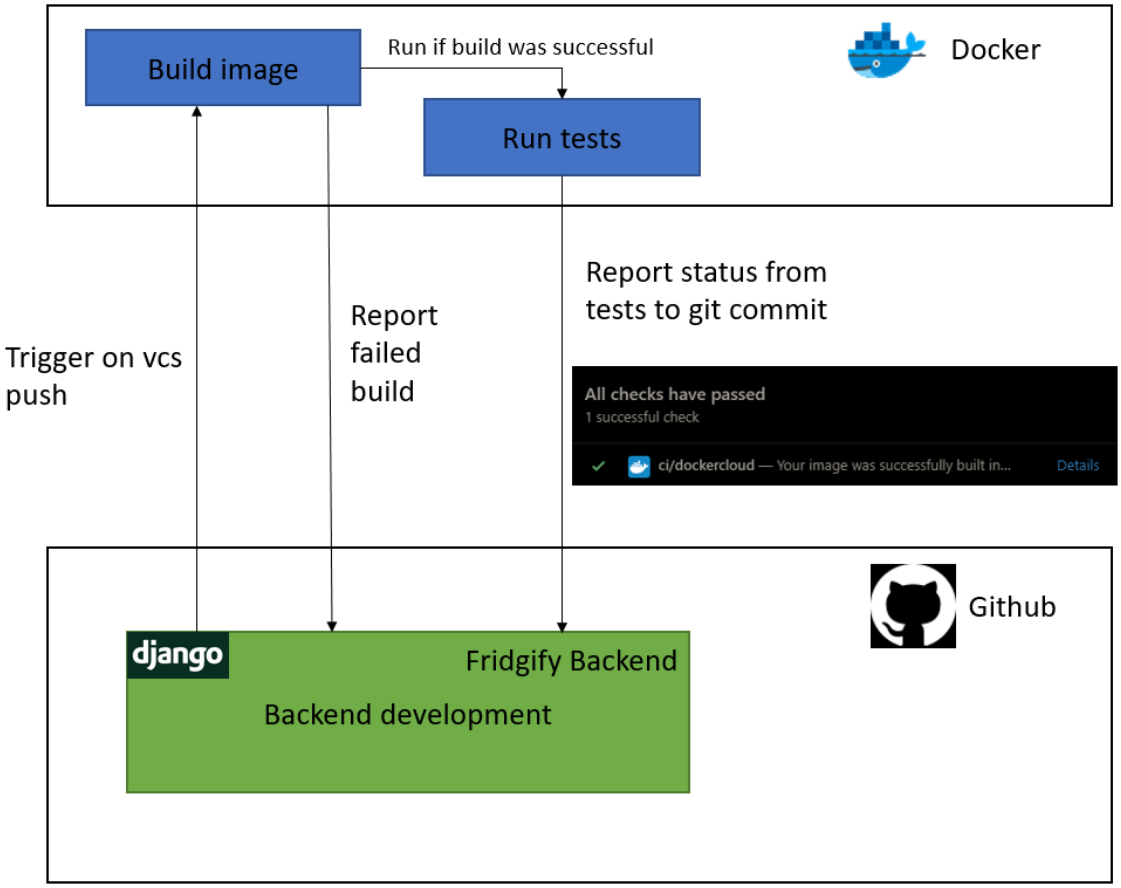
Categories: Allgemein

Another week, another blog. This week is about testing and code coverage. Our goal to write the best code possible forced us to write tests from day one. This benefited us on many occasions because we automated this as soon as we had a small suit of tests to execute. Our task was easy for this week. We only have to present what we already have and create code coverage badges to further illustrate our current test suit. So no more talking, more showing.

Testing



Our frontend tests run on our teamcity server as soon as someone pushes onto any branch in our frontend repository. The result of these tests are displayed as a status of the commit. Flutter offers a great built in test framework that offers a wide range of possibilities to test our app. The tests can be found in the test directory.



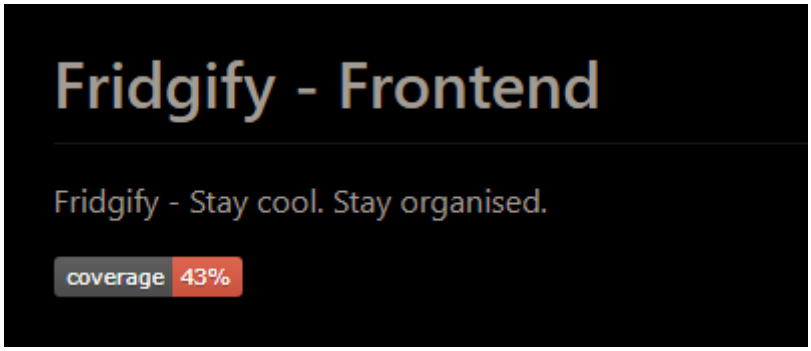
Testing in the backend works a bit different from our frontend. We are using docker to publish our backend api, with our own docker hub organisation Fridgify. We are using the built in image testing of docker hub. This works as follows. Each push on a feature branch triggers an image build on docker hub. Docker hub runs our tests, if the build is successful and reports either passing tests or that the build failed or that a test failed. This is represented in the commit status of git commits. Our tests can be found in the test directory.

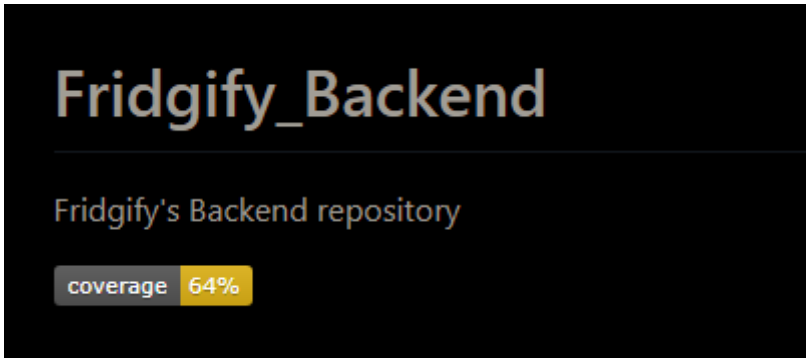
If you want to have an overview of how we want to do our tests, you can have a look at our Testplan.

Badges

Our code coverage badge creation is automated to update the badge each time someone pushes a change. Our badges can be found on the READMEs in the backend and frontend

repository. But I know everyone is to lazy to click on the links so here are pictures.





We used three packages to generate these badges. `flutter_coverage_badge` did the trick with flutter and `django-nose` and `coverage-badge` helped with python and django.

We know that the coverage numbers are low, and we are working to increase them. Expect higher numbers in the following weeks.

If you dont believe that our tests are passing by now, here are screenshots to satisfy the tinfoil heads.

Frontend

```
[Step 3/4] 00:13 +60: /frontend/tests/Fridgify_Frontend/fridgify/test/unit/repository_test.dart: Get token should throw an exeption
[Step 3/4] 00:13 +60: /frontend/tests/Fridgify_Frontend/fridgify/test/unit/repository_test.dart: Get token returns the token
[Step 3/4] 00:13 +61: /frontend/tests/Fridgify_Frontend/fridgify/test/unit/repository_test.dart: Get token returns the token
[Step 3/4] 00:13 +61: /frontend/tests/Fridgify_Frontend/fridgify/test/unit/repository_test.dart: Get headers returns headers
[Step 3/4] 00:13 +62: /frontend/tests/Fridgify_Frontend/fridgify/test/unit/repository_test.dart: Get headers returns headers
[Step 3/4] 00:13 +62: All tests passed!
[Step 3/4] All tests passed
```

Backend

```
[Step 1/1] Ran 50 tests in 2.247s
[Step 1/1]
[Step 1/1] OK
```

Update: User Experience Test

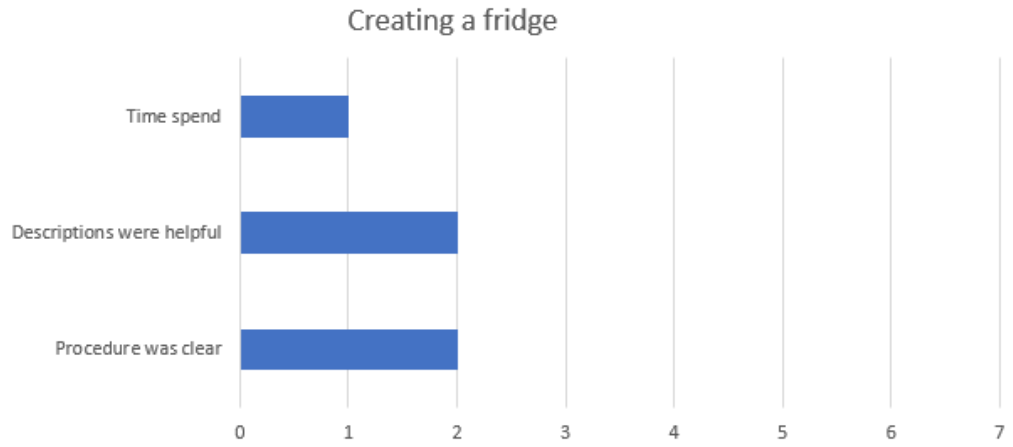
After realeasing our app as an alpha to selected users, we handed out a short questionare about our app. But we know that everyone hates long questionaires. We limited ourselves to three short questions that we apply to different parts of our app. The three questions are:

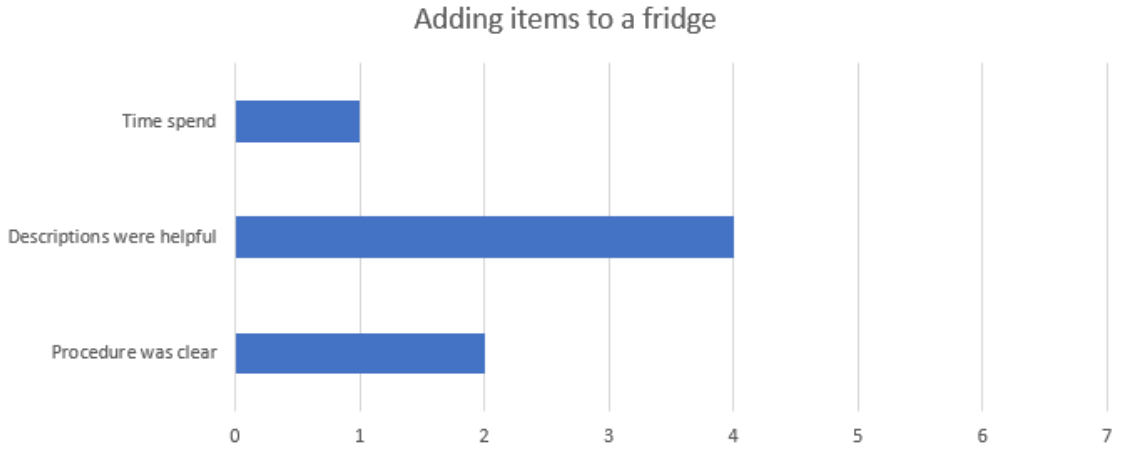
- Procedure was clear
- Descriptions were helpful
- Time spend

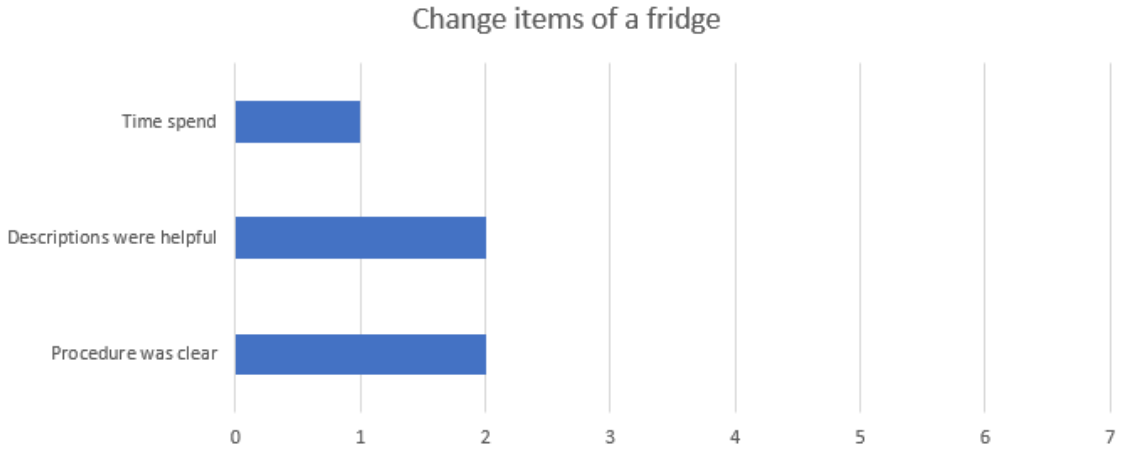
We asked these questions about the following parts of our app.

- Login / Register
- Creating a fridge
- Adding items to a fridge
- Change items of a fridge

These questions can be answerd from 1 to 7. 1 being good, 7 being bad and 4 being neutral.







Excel: Excel

The questionare shows that we achieved a good usability and only need to improve descriptions when adding items to a fridge. We are really happy with the outcome of this questionare and are eager to hear your feedback once we release officialy.

This was all from us this week, see you next week. Until then…

Stay cool. Stay organized.

— Your **Fridgify Team**

PHASE 2 WEEK 5 – CLEANING SOME MESS

May 15, 2020

Categories: Allgemein

Here we are again, welcome everyone to this week's blog entry. Today, we are going to.. *dramatic break* .. touch running systems. Right, we, the Fridgify Team, are going to follow Chapter 1 in Fowler\'s Refactoring, Improving the Design of Existing and try to refactor some existing code.

So, we forked this repository and tried to clean it up.

Here are our approaches:
Joschuas Repository
Dennis's Repository
Ducs Repository

Using IntelliJ and pressing **SHIFT+F6**, we all had the handy feature of easy refactoring and renaming all occurrences of a variable.

Stay tuned for future updates, because just between us, the application is almost done, and might hit the playstore of your choice in the next couple weeks. 😉

Until then…

Stay cool. Stay organized.

— Your Fridgify Team

PHASE 2 WEEK 6 – THEY ARE EVERYWHERE

May 24, 2020

Categories: Allgemein

As a developer you may know this feeling:

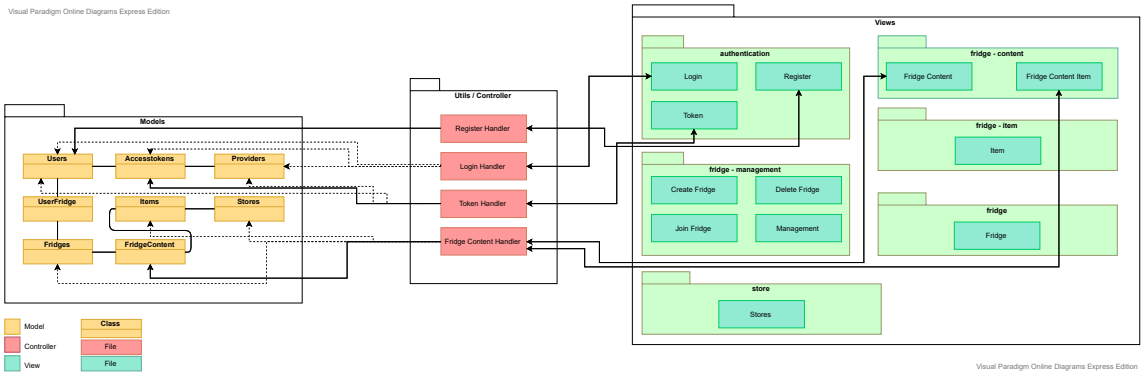
Everybody talks about **Design Patterns**. Every major project

uses some kind of pattern, every experienced software developer recommends them.
In modern software development, structured clean code gets more and more relevant. Because we at **Fridgify** aim for clean and fancy code, this blog is going to delve into a few design patterns we used.

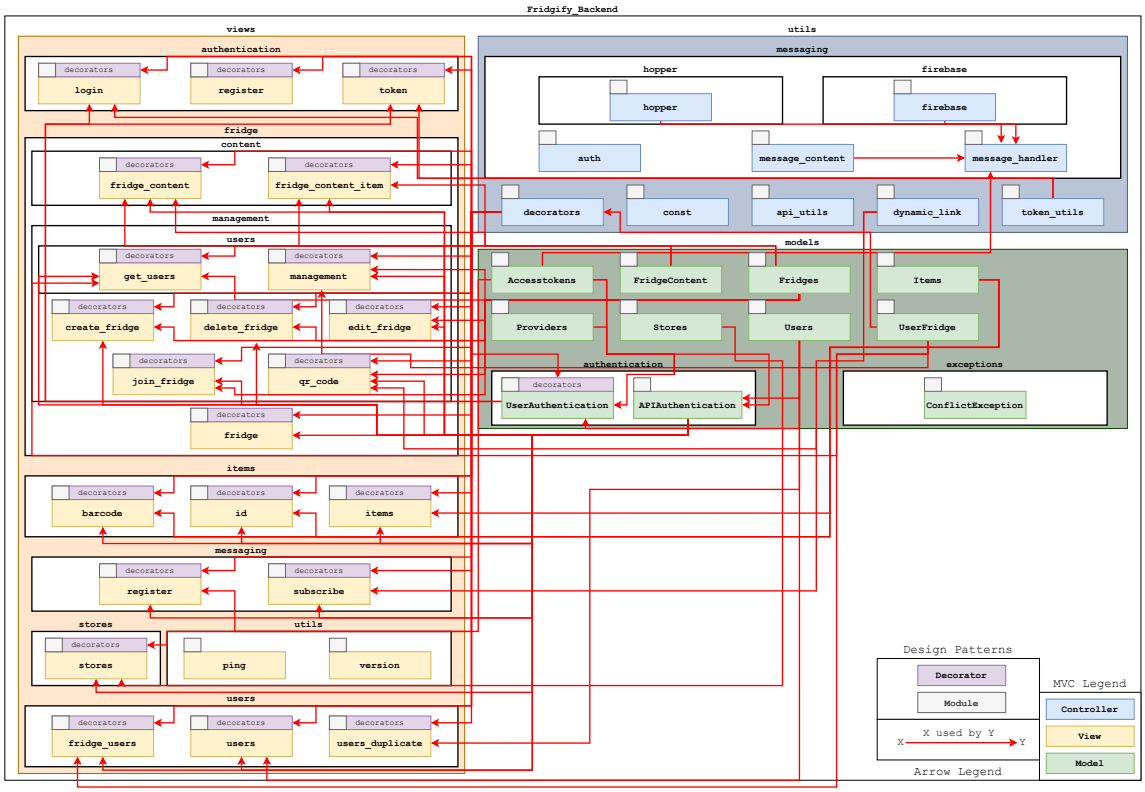
Design Patterns

As we stated in a previous blog, we went through an excessive refactoring phase, in which we changed our project structure. The goal of our work: A more *structured* and more *easy to maintain* code base. Especially the backend went through major change ups.
Just have a look at our UML diagram from before and after our refactoring process:

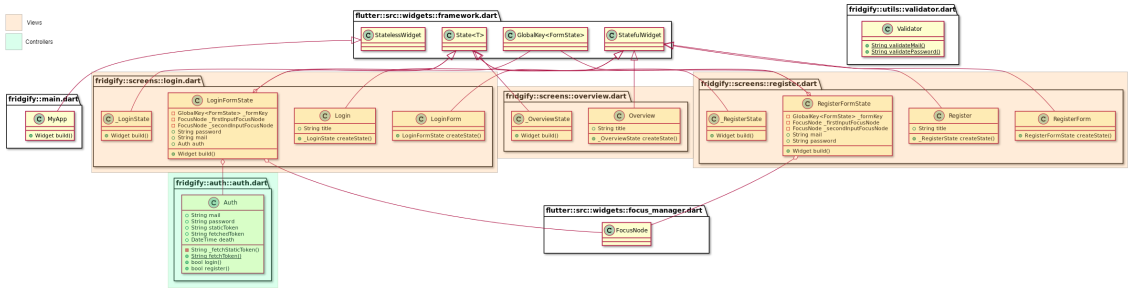
Backend – Before Refactoring:



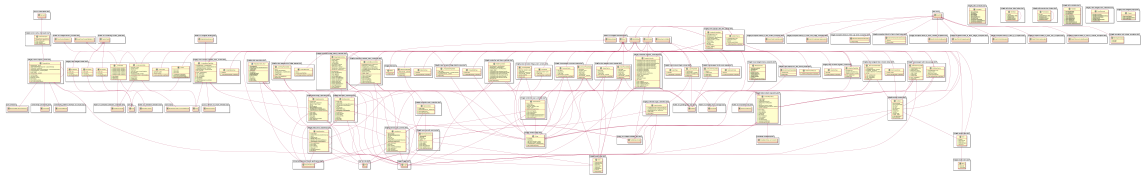
Backend - After Refactoring:



Frontend - Before Refactoring:



Frontend - During Refactoring:



You may have noticed, that the backend complexity increased a little. A lot of it is of course due to us adding new functionalities, but also implementing some new fancy design patterns. So let's talk about **two** design patterns we used in the backend.

Modules

Let's get to the most obvious one: **Modules**. Because we use *Python* as a programming language in our backend, the *module pattern* is already built in! **Modules** are categorized as *structural patterns*. If you are not familiar with this pattern, it essentially groups several related elements into a single conceptual entity. This entity could contain classes, methods or even variables. In *Python*, every .py-file is a module.
We use this pattern to prevent multiple instances of a variable. This is used in our messaging services, as we only want unique application instances of them.

Firebase Messaging Example:

```
import logging

import firebase_admin
from firebase_admin import messaging

app = firebase_admin.initialize_app()
logger = logging.getLogger(__name__)

def send_message(recipients, title, body, **kwargs):
...
```

As you may see, we have multiple variables declared in the module scope. This means they are essentially unique, which helps us only initalizing one application through the whole runtime. If you want to have a deeper look into this code example, you can find it here.

Decorators

Now we will get to the juicy patterns: **Decorators**. The *decorator pattern* allows developers to extend objects and methods dynamically without affecting the actual behavior. Just like the *module pattern*, decorators fall under the category of *structural patterns*

What is our reasoning behind using such a pattern? During our code analysis in our refactoring phase, we saw a lot of redundancy. Especially our authentication process had duplicate code blocks in nearly every functional view. To get rid of them, we could have just created an authentication controller and used that in every method **OR** we would use decorators to create cleaner and more comprehensive code.
Here a short example of how our `@check_body` decorator in action looks like (click here for the big one):

```
@check_body("name")
@permission_classes()
@authentication_classes()
def create_fridge_view(request):
...
```

The `@check_body` decorator checks the request body, if certain keys are inside the body.
And here is the implementation of said decorator (look here for the full implementation):

```
def check_body(*keys):
   def decorator(func):
       @wraps(func)
       def wrapper(request=None, *args, **kwargs):

           return func(request, *args, **kwargs)
       return wrapper
   return decorator
```

As you can see here, we execute our checking logic before executing the actual function. Decorators gave us the freedom to add additional functionalities, before going into the actual view. This way we are able to throw errors, if certain pre-conditions are not met.

In the end, using design patterns helped us a lot in structuring our code in a comprehensive way and make it a

lot easier to read. Besides we were able to remove a lot of redundancy by using them. So, if you are not already using any **design patterns**, consider using some. *Modules* and *Decorators* are just a few patterns, which are very useful in structuring your code.

Do you think *design patterns* deserve their praise? We would love to hear your feedback and opinion in the comments. Hope to see you soon. Until then…

Stay cool. Stay organized.

— Your **Fridgify Team**

PHASE 2 WEEK 7 - METRICS

June 5, 2020

Categories: Allgemein

Let’s talk about **Metrics**.

<!-- Metrics are measures of quantitative assessment commonly used for assessing, comparing, and tracking performance or production. (source: Investopedia)
-->

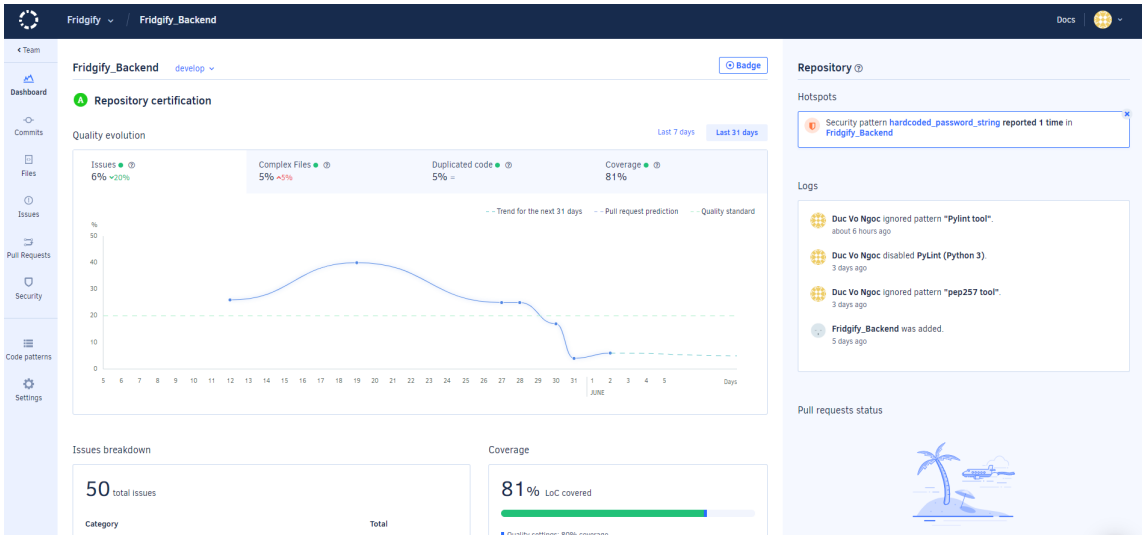
Codacy

Metrics in software development gain more importance each day. Different companies offer different tools to automize **Metrics collection**. **Fridgify** uses *Codacy* for collecting its metrics. *Codacy* offers a lot of metrics and easy to set up. It allows you to have an instant look at the quality of your repository. The following dashboard shows some neat features of *Codacy*.

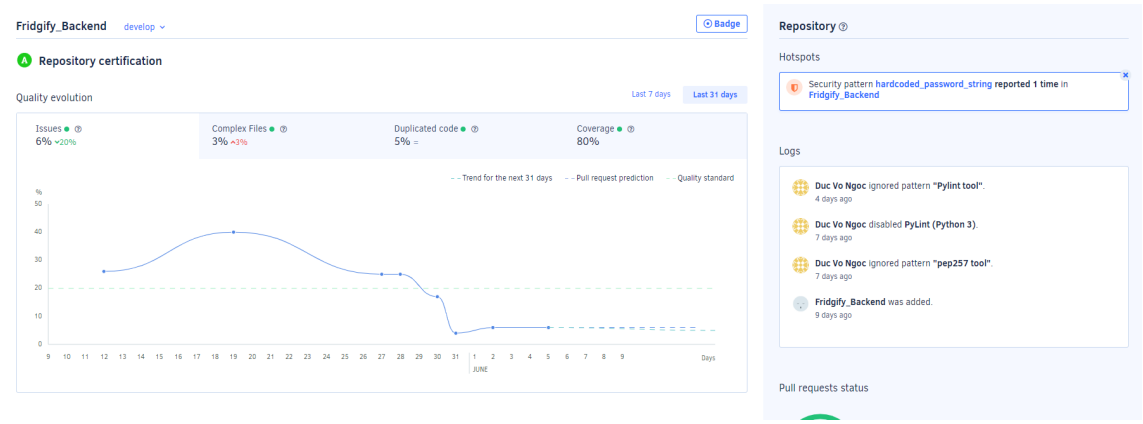
Codacy Dashboard – Backend

Codacy Dashboard – Frontend

Before Refactoring:



After Refactoring:



Cyclomatic Complexity

One measurement we looked into is **cyclomatic complexity**. Developed by Thomas J. McCabe, Sr. in 1976, **cyclomatic complexity** is used to indicate the complexity of a program. It is computed by using the control flow graph of the program.
Basically each control flow is being counted. If the count exceeds a specific number, the file is to complex. Control flows are for example: `if` , `elif` , `except` and more. As Codacy uses *Radon* for complexity in Python, you can have a deeper look here.

As you see in the dashboard, we had a small percentage of files, which are to complex. Our files should only have a complexity of 5.

In the following code example, we had to many control flows. To minimize the count of control flows, we created a decorator, which externalized code, which could be reused as well. Additionally, instead of checking for *username* and *email* duplication seperately, we decided to check it via one Django statement. Moving the initalization of status and detail messages to the beginning, allowed us to remove an `if` -statement too.

Before Refactoring: full code

```
@api_view()
def users_duplicate_view(request):
    """Entry point for duplicate user views"""
    logger.info("Check for duplicate user...")
    try:
        body = json.loads(request.body.decode("utf-8"))
    except json.JSONDecodeError:
        logger.error("Couldn't parse JSON:\n %s", request.body.decode('utf-8'))
        raise ParseError()
    exists = {}
    for key in body.keys():
        if key == "username":
            logger.debug("Check for duplicate username %s...", body)
            if Users.objects.filter(username=body).exists():
                exists = body
        if key == "email":
            logger.debug("Check for duplicate email %s...", body)
            if Users.objects.filter(email=body).exists():
                exists = body
    logger.debug("Existing values: %s", repr(exists))
    if exists:
        return Response(data=exists, status=409)
    return Response(data={"detail": "No duplicates"}, status=200)
```

After Refactoring: full code

```
@api_view()
@required_either_keys("email", "username")
def users_duplicate_view(request):
    """Entry point for duplicate user views"""
    logger.info("Check for duplicate user...")
```

```
    try:
        body = json.loads(request.body.decode("utf-8"))
    except json.JSONDecodeError:
        logger.error("Couldn't parse JSON:\n %s", request.body.decode('utf-8'))
        raise ParseError()

    username = body.get("username")
    email = body.get("email")

    exists = {"detail": "No duplicates"}
    status = 200

    if Users.objects.filter(Q(email=email) | Q(username=username)).exists():
        exists = email
        exists = username
        exists.pop("detail")
        status = 409
    logger.debug("Existing values: %s", repr(exists))

    return Response(data=exists, status=status)
```

Duplicated Code

This code metric should be self-explanatory. Due to us having a very low amount of duplicated code lines, we decided to not look into it. Most of our code duplications are needed, as we use the same structure for our decorators. They have the same inner function names and often times fulfill similar tasks. One decorator for example checks the request body, if it contains all required keys, while another decorator just checks if one of the required keys is inside. If you want to have a look at our decorators, look here.

Conclusion

We think metrics helped us in improving our code quality and making it more maintanable. Especially the code complexity is a very helpful code metric. Keeping code

simple is especially in Python very important, as its goal is to be as easy to read as possible.
What do you think about code metrics? What do you use? We are excited for your opinion! Until then…

Stay cool. Stay organized

— Your **Fridgify Team**

PHASE 2 – WEEK 8 THE ART OF SCRUM III

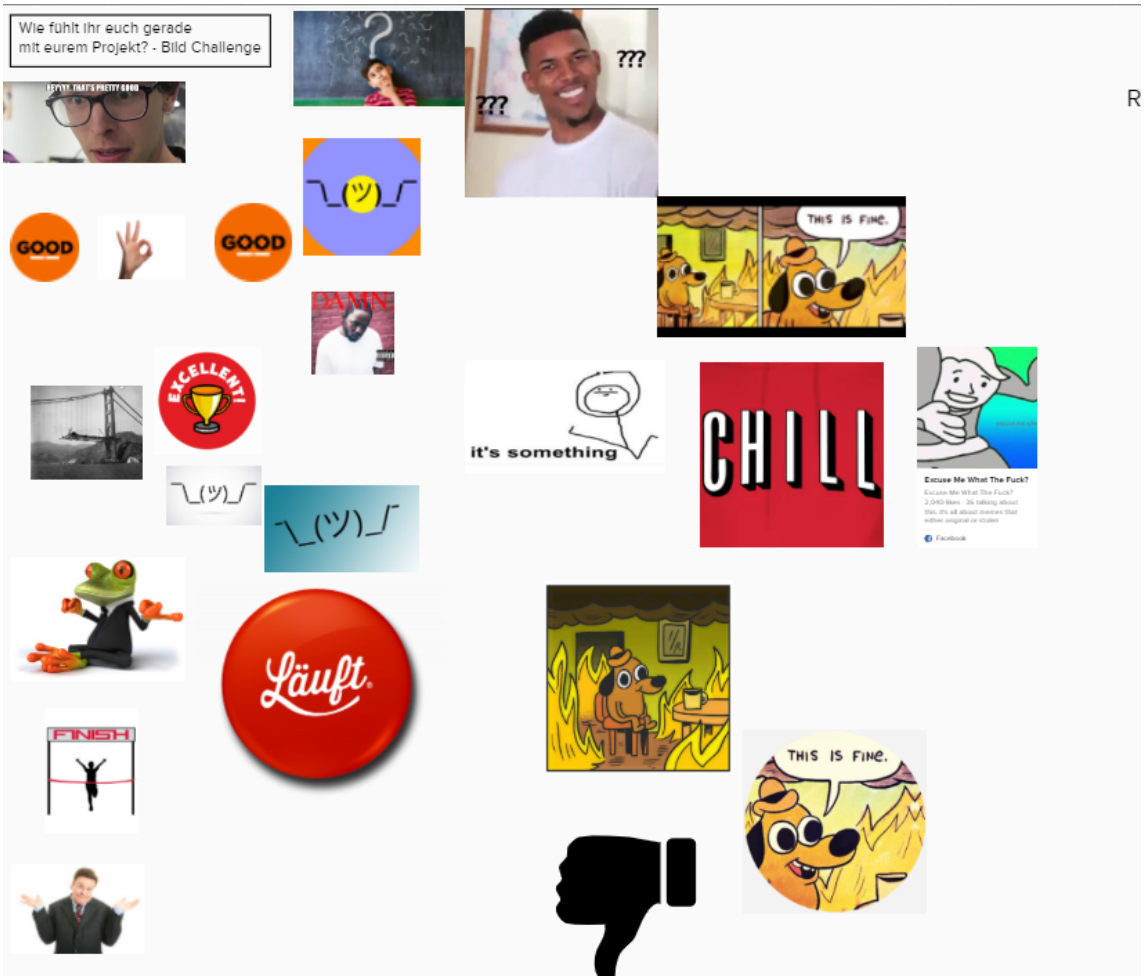
June 7, 2020

Categories: Fridgify's Journey

It has been a while since we talked about Scrum, and I think everyone here knows about our two-part story of (https://blog.fridgify.com/week-7-the-art-of-scrum-ii/ **"The Art of Scrum"**) if not you should check it out. Today, we will talk about the third and last part of **The Art of Scrum**. Again taking a look at our progress, having yet another retrospective, and comparing it with what we have achieved the last time we had one.

Opening Game

Just as the last time we opened the retrospective session with an opening game, other than last time, where we had to draw a picture of ourselves as a vehicle, we had to pick an image with which we identify ourselves and the project.

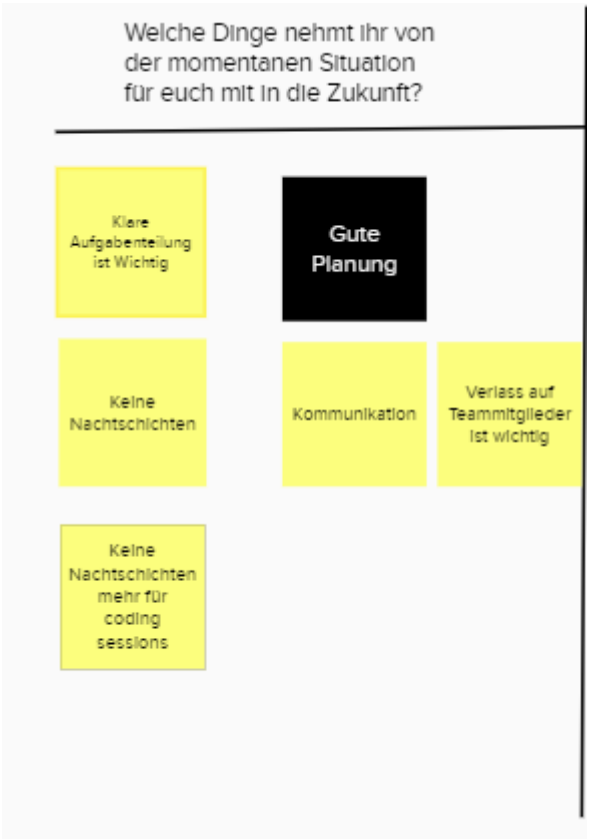


This was the achieved final image.

- Duc went with the "Okay hand sign", representing his overall satisfaction with our current situation.
- Joschua interpreted the question as a "How is the situation overall right now"- question, so he went with the "This is fine" image, showing him being okay, while everything around the world is going down right now.
- Dennis saw himself in the upper left "Hey, that is pretty good" image because he was surprised at how good the project turned out.

What are the things you will take with you for future projects?

Next, we were grouped with Evendo, to answer three questions, the first being the one above.



The summary is, planning, communication, and being able to rely on team members, which all have their essential tasks, are crucial and will accompany us for further projects.

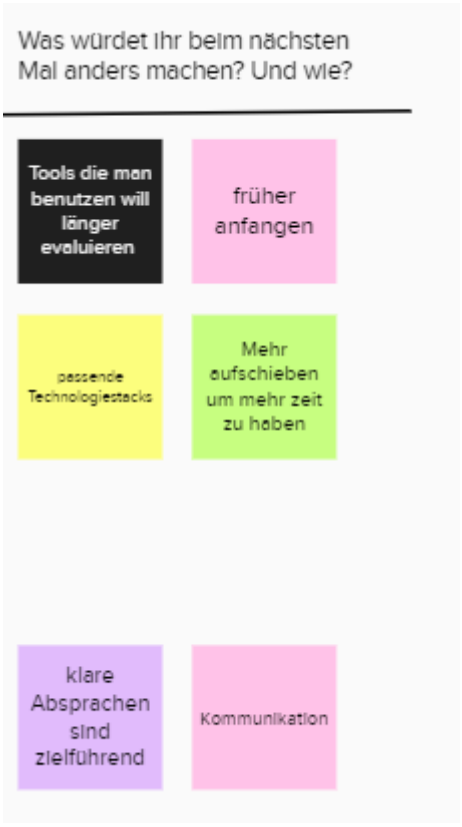
What did you learn?

Then we had to gather the things we learned form this project.

Overall, we learned how to document, program better, and divide tasks better. We determined that if the project's planning gets too complicated, its implementation will not be as easy as well, and selecting the right technology and using the right tools in the project is also essential.

What would you change?

Last but not least, we had to gather the things we would change the next time.



]

The changes are similar to the things we learned. We would evaluate the tools we want to use better, select the right technology stack, and communicate better to achieve our goal quicker.

As you can see, we learned a lot throughout this project, but are still learning.

We are looking forward to even further progress in the next week.
Hopefully, you are as thrilled as we are. Until then…

Stay cool. Stay organized.
— Your Fridgify Team

PHASE 2 WEEK 9 - INSTALLATION

June 15, 2020
Categories: Allgemein

Coming closer to the end we want to give you the chance to check out our progress for yourself. Both our backend and frontend reached a state we are proud to release.

We prepared two tutorials explaining how you can deploy fridgify yourself. It is beneficial to start with the backend. Following up with the frontend be sure to enter your backend url that you set up.

We are thrilled to hear how it went and always there for your questions.
Until then…

Stay cool. Stay organized

— Your **Fridgify Team**

PHASE 2 FINAL – FINAL HAND-IN

June 26, 2020

Categories: Allgemein

Before you start reading this blog, you should play this.
Now we are ready to go…
As you may know, all great things should come to an end. **Fridgify** was a great journey for all of us. We went through hardships as well as a lot of success. We achieved a lot. Ranging from a basic Android application (version 1 of our app) to now having our nearly fully fledged running app, we are sure to be proud of our progress. Let's have a look at our past journey together:

Do you remember the beginning? Our first steps into this big unknown software development world. As every amazing application, it all started with a stupid idea: Week 1 – Introduction

We had to overcome ourselves, taking on roles, which were out of our comfort zone. All of us taking on several leadership roles, was quite a change: Week 2 – Roles and Technology

<!-- A goal without a plan is just a wish
- Antoine de Saint-Exupéry
-->

We just had a small idea, but we lacked a concrete plan. Talking about requirements and modelling use cases as a team, helped us grow together: Week 3 – SRS and OUCD

Although we weren't experts in graphic design, we took it as a challenge. We had to overcome our own shortcomings and approached wireframing and mock upping head on: Week 4 – Wireframes, Mockups and Modelling

It's already been 5 weeks. We converted to vegeterianism and decided to use cucumber. It's been quite a change, as feature files were new to us: Week 5 – Vegetables, Feature Files and Modelling

As all of us worked with Scrum, we were very comfortable. A welcome change of pace: Week 6 – The Art of Scrum

Retrospectives help you grow as a team, as did we. We talked about our mistakes as well as our successes and decided to work towards better team work: Week 7 – The Art of Scrum II

From hour one on, we decided to closely work together. Backend and frontend should work as one single unit, which is why we create diagrams and database structures as a team: Week 8 – Class Diagram and Database Structure

Every project needs a structure. As the saying goes: *'Old is gold' – Mahatma Gandhi (probably, maybe)*, so we sticked

to MVC for our backend **and** frontend: Week 9 – Architecture

We don't talk about week 10.

It's been 11 weeks so far and we were proud to have something to show. Our midterm finally arrived: Week 11 (Yeah, we kinda skipped 10) – Summary

Months went by and we worked in secret on some big things. Everything underwent a refactoring process after our rose-colored glasses came off, everything was kinda…meh…"okay", kinda…meh…"bloody". *Vietnam flashbacks intensify* (don't ask or ask, we are done)

The 4th semester started and we were mentally and physically ready to go. We cooled again and redefined our use cases: Phase 2 Week 1 – The Fridge cools again

Risks are always there. *'Don't be silly, wrap your willy'* – literally every sane human being. Protecting our success from possible risks is important, which is why we sat down and talked about Risk Management: Phase 2 Week 2 – Risky Stuff

It's important to keep an eye out for time. Time passes and each and every second is important. Every minute 60 seconds pass in Africa. We have to use this valuable time, which is why we sat down and worked out an estimate of how much time we will spent by utilizing an insane formula: Phase 2 Week 3 – How long will it take?

Badges are pretty neat, ey. Can we get a badge for having the most badges?!!11: Phase 2 Week 4 – Testing and Badges

So appearently we did not do enough refactoring during the break, so we decided to do even more refactoring. Because `refactoring + refactoring = great stuff` . (To the people we pissed off: You're welcome): Phase 2 Week 5 – Cleaning some mess

Because refactoring + refactoring wasn't refactored enough we decided to refactor our code to include some patterns.

(Author's note: I dream of patterns. I am a pattern. We are all patterns): Phase 2 Week 6 – They are everywhere

No one cared for us, until we set up Codacy *heavy breathing* (r/creepyasteriks): Phase 2 Week 7 – Metrics

We sat down for the last time (or met up online) and talked about our project's progress: Phase 2 Week 8 – The Art of Scrum III

We saw a lot of demos. It seems like projects showed peak performance in procrastination.

It is the 9th week after writing nearly 20000 lines of code. We are proud parents, which will soon part ways with their child. This is heartbreaking…: Phase 2 Week 9 – Installation

It's been a long day without **Fridgify**, my friend. And we'll tell you all about it when we code again. We've come a long way from where we began. We've always stayed cool. We've always stayed organized, my friend.

This blog is our final blog on our journey. It's been a blast. That is only because of you. You supported us all the way through.
Thanks for coming to our TED talk.

TLDR

An actual nice blog overview: Blog Overview

Requirements	
UC	Folder with all UC
SRS	SRS
Test Cases (.feature files)	Folder with feature files

Requirements	
Test Log	Test logs
Test Coverage	Frontend, Backend
Test Plan	Test plan
Functional Test	Folder with feature files
Unit Test	Frontend unit test folder, Backend unit test folder
+ Extra Test	User test

Project Management	
RUP gantt chart	Cumulative Diagram, Gantt, Time
Long-term planning FP-estimation	Folder with FP things
Burn-down chart of sprint	Chart 1, Chart 2
Retrospective Semester 1	Retrospective 1
Retrospective Semester 2	Retrospective 2
FP calculation	Excel with values and charts

Code Areas	
Frontend	Code written by us
Frontend test	Unit
Backend	Code
Backend test	Tests

Installation	
Blog	By hopper

Quality	

Quality	
Architecture SAD	Architectural Representation, Architectural Goals and Constraints, Architecturally Significant Design Packages
Pattern in SAD	Architecturally Significant Design Packages
Metrics SAD	Quality/Metrics
Risk Management	Risk Management Excel
Automated Testing	Blog entry about automation
Functional Test	Folder with feature files
Automatic Deployment	Picture, Development, Production

Sike

Do not worry. We would never give **Fridgify** up. After a short exam break, we will continue development. If you are interested in testing **Fridgify** hit us up and write us an email to get access to our closed Alpha **(support@fridgify.com)** or try out the web app at app.fridgify.com. We will for sure see you again.
Until then…

Stay cool. Stay organized

— Your **Fridgify Team**