

SE Project Documentation

Spontaneous

Semester: Spring 2023



Version: 03.00

Date: 2023-06-08 16:48:03Z

Project Team: Dario Glasl
Jasmin Fitz
Samuel Maissen
Mirio Eggmann
Nick Wallner

Project Advisor: Thomas Kälin

Contents

I	Management Summary	1
II	Product Documentation	3
1	Requirements	4
1.1	Functional Requirements	4
1.1.1	User Story Structure	4
1.1.2	Roles	5
1.1.3	Epics	5
1.2	Non-Functional Requirements	6
1.2.1	Reliability	6
1.2.2	Performance Efficiency	7
1.2.3	Compatibility	7
1.2.4	Usability	8
1.2.5	Security	8
1.2.6	Portability	9
1.2.7	Tracking of the NFRs	10
2	Domain Analysis	16
2.1	Domain model	16
2.1.1	User	17
2.1.2	Visit	17
2.1.3	Activity	17
2.1.4	Preference	17
2.1.5	Filter	17
3	Design	18
3.1	UI Mockups	18
3.1.1	Home Screen	18
3.1.2	Activity Screen	19
3.1.3	Favorites Screen	20
3.1.4	History Screen	20
3.1.5	Account Screen	21
3.2	Color Scheme	22
4	Architecture	23
4.1	Architecture Diagrams C4	24
4.1.1	C4 Context Diagram	24
4.1.2	C4 Container Diagram	25
4.2	Sequence Diagrams	26
4.2.1	Activity Proposal Sequence Diagram	26
4.2.2	Authentication Sequence Diagram	27
4.3	Backend Architecture	28
4.3.1	Architectural Pattern	28

4.3.2	Architectural structure	29
4.4	Frontend Architecture	30
4.5	Pipeline / Deployment Architecture	30
4.6	Infrastructure Architecture	31
4.6.1	Database	31
4.7	Technology Decisions	33
4.7.1	Evaluation Dev Stories	33
4.7.2	Frontend	33
4.7.3	Backend	33
4.7.4	Infrastructure	34
4.7.5	Pipeline	34
4.7.6	Database	34
4.7.7	API's	35
4.7.8	Testing	35
4.7.9	Tools	35
4.8	Tech Stack	36
5	Quality Measures and Project Guidelines	37
5.1	Code	37
5.2	Documentation	37
5.3	Jira	37
5.4	Gitlab	37
5.4.1	CI/CD	37
5.5	Git Workflow	38
5.5.1	Branching	38
5.5.2	Merge requests	38
5.6	Test Strategy	39
5.6.1	Functional Testing	39
5.6.2	Non-Functional Testing	39
III	Project Documentation	42
6	Initial Project Proposal	43
7	Project Plan	45
7.1	Project Management	45
7.1.1	Project Management Method	45
7.1.2	Project Meetings	45
7.1.3	Project Management Roles	46
7.2	Long Term Planning	46
7.2.1	Milestones	47
7.3	Short Term Planning	48
7.3.1	General Workflow	48
7.3.2	First Iteration	48
7.4	Risk Management	49
7.4.1	Updated Risk Matrix - Sprint 2	52
7.4.2	Updated Risk Matrix - Sprint 3	53
7.4.3	Updated Risk Matrix - Sprint 4	54
7.4.4	Updated Risk Matrix - Sprint 5	54
7.4.5	Updated Risk Matrix - Sprint 6	55
7.4.6	Updated Risk Matrix - Sprint 7	55

8	Time Tracking Report	56
8.1	How we Track Time	56
8.2	Time Statistics	56
8.2.1	Overview Current Sprint	57
8.2.2	Planned and Invested Time per Sprint	58
8.2.3	Invested Time per Topic	59
8.2.4	Overall Invested Time per Member	59
8.3	Final Overall Statistic	60
9	Development Environment	61
9.1	Git Project	61
9.2	Frontend	61
9.2.1	Installation	61
9.2.2	Configuration	61
9.2.3	Development	61
9.2.4	Testing	62
9.3	Backend	62
9.3.1	Installation	62
9.3.2	Configuration	62
9.3.3	Development	62
9.3.4	Testing	62
9.4	Database	62
9.4.1	Installation	62
9.4.2	Startup	63
9.4.3	Access	63
9.5	Infrastructure	63
9.6	Automatic Testing	63
10	Personal Reports	64
10.1	Retrospective Meetings	64
10.2	Project Retro	64
10.2.1	Detail Screenshots	65
11	Meeting Minutes	68

Part I

Management Summary

Management Summary

Spontaneous is an innovative web application designed to address the challenge of making decisions quick and easy for individuals and groups. The main problem solved is the difficulty in choosing what activity to engage in or where to eat. The objective was to simplify the decision-making process, encourage people to try new things and step out of their comfort zone.

To solve this problem, a user-friendly interface was developed. It allows users to input basic criteria and receive random proposals for activities. The proposed activities can easily be shared with friends to facilitate group decision-making. Additionally, it is possible to log in with a Google account to save favorite activities and view past proposals.

While the solution is working and the main objectives have successfully been met in terms of functionality, there is still room for improvement. The key aspect that requires attention is the selection of activities, which currently relies solely on the Google Places API as data source. In retrospective, this is not the best source, since it predominantly provides information on restaurants and bars, while offering fewer options for other activities such as sports or cultural events.

The tech stack used in the development of Spontaneous consists of a C# .NET backend, a React and Typescript frontend, as well as a MongoDB for storage. GitLab was used for development collaboration and running the software pipeline, ensuring efficient development processes. Hosting is done over Kubernetes on a cluster in the Google Cloud Platform, providing scalability and reliability for the application.

Throughout the project, we have learned a lot about coordination and the development process in an agile workflow measured by waterfall reviews. We tailored our approach to fit our needs and project requirements, utilizing a combination of tools. Jira served as our central platform for task management, planning, and time tracking. We utilized Confluence for meeting notes and idea drafting. Miro helped us create diagrams and retro boards, while Figma aided in UI mockup creation. These tools streamlined our workflow, improved collaboration, and facilitated efficient development.

The existing solution provides a solid foundation for possible future development and improvements. Potential consecutive projects could include user preferences that could be saved in the account, some statistics about your usage or the possibility to follow your friends and see what they are doing. However, there are no plans to continue the project at this time.

In conclusion, Spontaneous offers a promising solution to the challenge of decision-making, and we enjoyed developing, testing and finishing it together as a team.

Part II

Product Documentation

Chapter 1

Requirements

Our requirements are mainly tracked in [Jira](#).

1.1 Functional Requirements

We use user stories and epics to define functional requirements.

- The user stories are defined in Jira and the epics (if needed) are used to group user stories towards a bigger requirement.
- We define developer stories for non business requirements related stories.
- For tracking the reviews, we create tasks.

1.1.1 User Story Structure

A user story is defined as follows:

- User Story: As a *role* I want to *goal*
- Priority: *lowest, low, medium, high, highest*
- Acceptance criteria
- Technical acceptance criteria

The user stories that are lower in the backlog are defined briefly, the ones that are higher are more detailed. We fully write out the user stories during the sprint planning.

1.1.2 Roles

Our domain knows only **users** of the application Spontaneous. Regarding monetization for Spontaneous, there is a role called **business owner**, but the functionality for it has a low priority, therefore this role is insignificant for now.

The role **user** is the main protagonist in our application. He uses Spontaneous to find new activities. If he wants, he can log in and then view an activity history and save favorites. A more detailed description of the user role can be found in the chapter "Test Strategy" under Target Group.

1.1.3 Epics

These epics are defining our business requirements. Others like developer epics are defined in the [Jira backlog](#) prefixed with "DEV-" or "R1-7" for the reviews. The epic priorities are tracked in Jira but generally they are listed from most important to least important.

User Account [SPON-59](#)

Users can login with their Google accounts for which their data is stored in the database.

Basic Activity Proposal [SPON-108](#)

This epic relates to the mvp of Spontaneous, with its basic functionality where users can get an activity proposed and see their proposed activities.

Advanced Activity Proposal [SPON-148](#)

Refines the proposal mechanism. Users can get an activity proposed based on their history or only being able to get one proposal per day.

Activity Filtering [SPON-149](#)

Users can choose which types of activities they want to get proposed.

Personalization [SPON-109](#)

Users can personalize their experience by setting user preferences for example for filters and choose activity favorites.

Gamification [SPON-102](#)

The application uses gamification to motivate users to use the application. For example: users can progress in ranks for using the application.

Multiplayer [SPON-103](#)

Users can interact with other users. For example: users can invite other users to an activity.

Monetization [SPON-105](#)

The application can be monetized by selling premium features. For example: users can buy a premium account to get more proposals per day. Also business owners can buy boosts for their businesses to be proposed more often.

1.2 Non-Functional Requirements

We use the ISO/IEC 25010 standard to define our NFR's. Please be aware that on purpose not all the categories are defined, since not all are relevant for this project and there is limited time. Our goal is to implement all NFR's with the priorities "high" and the most of the "medium" ones. If there are enough resources, we try to implement the NFR's with the priority "low" as well.

In order to track and plan the NFR's, we created epic's for each NFR. For each epic NFR there are developer stories in order to precise the steps needed for the fulfillment of the NFR. The developer stories were added into the backlog and planned in sprints.

The current status of the completed NFR's can be tracked in the following [Jira Dashboard](#)

1.2.1 Reliability

Table 1.1: Non-Functional Requirements: NFR01

ID	NFR01
Description	The system must be available during 99.9% of the operation.
Requirement	Reliability - Availability
Priority	low
Verification process	Create check in UptimeRobot.
Measures	The app is available during 99.9% of the operation (project timeline)

Table 1.2: Non-Functional Requirements: NFR02

ID	NFR02
Description	App should be able to self-heal in case of major system failures, such as server crashes or data center outages.
Requirement	Reliability - Recoverability
Priority	medium
Verification process	Manually hard power off entire Kubernetes Cluster.
Measures	Application can successfully recover from a simulated disaster scenario within 5 minutes.

Table 1.3: Non-Functional Requirements: NFR03

ID	NFR03
Description	There should be a database data backup in case of total data loss.
Requirement	Reliability - Recoverability
Priority	low
Verification process	Delete all data records and load backup dump into database.
Measures	Daily database data dump is created and dump can be restored to database.

Table 1.4: Non-Functional Requirements: NFR04

ID	NFR04
Description	In case that 1 container fails, the app is still usable.
Requirement	Reliability - Fault tolerance
Priority	high
Verification process	Kill 1 container per layer (frontend, backend, database).
Measures	The app is usable while single containers are killed.

1.2.2 Performance Efficiency

Table 1.5: Non-Functional Requirements: NFR05

ID	NFR05
Description	The query for an activity must not exceed 5 seconds during normal loads (20 Requests/Minute).
Requirement	Performance Efficiency - Time Behavior
Priority	low
Verification process	Test person or automated test performs activity query and evaluates the time elapsed.
Measures	Activity request with at least 10M/bit internet connection takes under 5 seconds.

Table 1.6: Non-Functional Requirements: NFR06

ID	NFR06
Description	The app should be able to handle a minimum of 50 activity requests per minute.
Requirement	Performance Efficiency - Capacity
Priority	low
Verification process	Simulate a load of 50 requests per minute with a stress simulation tool.
Measures	App can handle the load and process the minimum number of 50 activity requests per minute.

1.2.3 Compatibility

Table 1.7: Non-Functional Requirements: NFR07

ID	NFR07
Description	The app should be able to work with the Google IDP to provide login functionality to users.
Requirement	Compatibility - Interoperability
Priority	high
Verification process	Test if able to login with personal Google account.
Measures	The app can integrate with Google IDP and user authentication works.

1.2.4 Usability

Table 1.8: Non-Functional Requirements: NFR08

ID	NFR08
Description	There must be feedback for wrong entered values.
Requirement	Usability - User Error Protection
Priority	medium
Verification process	Users enters or automated test enters wrong value and verifies if feedback is provided.
Measures	Feedback gets provided if wrong values are entered.

Table 1.9: Non-Functional Requirements: NFR09

ID	NFR09
Description	The app should be accessible to users with green-red weakness.
Requirement	Usability - Accessibility
Priority	low
Verification process	Verify layout contrast ratio with Google Chrome lighthouse.
Measures	Layout is compliant with the Google Chrome lighthouse criteria "Background and foreground colors have a sufficient contrast ratio".

Table 1.10: Non-Functional Requirements: NFR10

ID	NFR10
Description	System must be able to be controlled only with keyboard.
Requirement	Usability - Accessibility
Priority	medium
Verification process	Users test if all elements are controllable only by keyboard. This is best effort.
Measures	Acceptance criteria: All elements are controllable with keyboard.

1.2.5 Security

Table 1.11: Non-Functional Requirements: NFR11

ID	NFR11
Description	User traffic to and from the webservice must be encrypted by using TLS.
Requirement	Security - Confidentiality
Priority	high
Verification process	TLS is used for the webservices.
Measures	Acceptance criteria: TLS v1.3 is set up and appropriate cipher suits are used.

Table 1.12: Non-Functional Requirements: NFR12

ID	NFR12
Description	The Webservice need to make use of a TLS Certificate in order to prove its authenticity.
Requirement	Security - Authenticity
Priority	high
Verification process	Check if certificate was installed and is valid for webservice.
Measures	Certificate is installed and valid.

1.2.6 Portability

Table 1.13: Non-Functional Requirements: NFR13

ID	NFR13
Description	The system must provide an REST-API which includes all actions possible in the GUI.
Requirement	Portability - Adaptability
Priority	high
Verification process	Developer checks if actions provided on GUI can be triggered over REST. This is best effort.
Measures	All actions provided on GUI can be triggered over REST.

Table 1.14: Non-Functional Requirements: NFR14

ID	NFR14
Description	The application must be compatible with different browsers and devices.
Requirement	Portability - Installability
Priority	high
Verification process	Using developers tools in Google Chrome test different resolutions and its responsiveness.
Measures	The main functionality (proposing an activity and see its result) must be cleanly executable and displayable. It should work within the following web browsers: Chrome, Firefox and Safari. Screen sizes in which the app should be responsive: 1920x1080px and 360x800px.

Table 1.15: Non-Functional Requirements: NFR15

ID	NFR15
Description	The data stored in the system must be able to be exported using a common data format.
Requirement	Portability - Replaceability
Priority	low
Verification process	Export data from database.
Measures	Data can be exported in common data format such as XML, JSON or CSV.

1.2.7 Tracking of the NFRs

Review NFRs - Sprint 2

Table 1.16: Tracking of the NFRS - Sprint 2

NFR Nr.	Priority	Status
NFR1	low	NOK
NFR2	medium	NOK
NFR3	low	NOK
NFR4	high	NOK
NFR5	low	NOK
NFR6	low	NOK
NFR7	high	NOK
NFR8	medium	NOK
NFR9	low	NOK
NFR10	medium	NOK
NFR11	high	OK: Verified on 21.03.2023. Used tool: Google Chrome Developer Tools (Security) Output: The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM. Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR12	high	NOK
NFR13	high	NOK
NFR14	high	NOK
NFR15	low	NOK

Review NFRs - Sprint 3

Table 1.17: Tracking of the NFRS - Sprint 3

NFR Nr.	Priority	Status
NFR1	low	NOK
NFR2	medium	NOK
NFR3	low	NOK
NFR4	high	NOK
NFR5	low	NOK
NFR6	low	NOK
NFR7	high	NOK
NFR8	medium	NOK
NFR9	low	NOK
NFR10	medium	NOK
NFR11	high	OK: Verified on 04.04.2023. Used tool: Google Chrome Developer Tools (Security) Output: The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM. Additional info: automatic redirect from http to https Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR12	high	OK: Verified on 04.04.2023. Used tool: Google Chrome Certificate Viewer Output: Issued by: Organisation (O) Let's Encrypt Additional info: no certificate warnings Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR13	high	NOK
NFR14	high	NOK
NFR15	low	NOK

Review NFRs - Sprint 4

Table 1.18: Tracking of the NFRS - Sprint 4

NFR Nr.	Priority	Status
NFR1	low	NOK
NFR2	medium	NOK
NFR3	low	NOK
NFR4	high	NOK
NFR5	low	NOK
NFR6	low	NOK
NFR7	high	NOK
NFR8	medium	NOK
NFR9	low	NOK
NFR10	medium	NOK
NFR11	high	OK: Verified on 18.04.2023. Used tool: Google Chrome Developer Tools (Security) Output: The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM. Additional info: automatic redirect from http to https Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR12	high	OK: Verified on 18.04.2023. Used tool: Google Chrome Certificate Viewer Output: Issued by: Organisation (O) Let's Encrypt Additional info: no certificate warnings Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR13	high	NOK
NFR14	high	NOK
NFR15	low	NOK

Review NFRs - Sprint 5

Table 1.19: Tracking of the NFRS - Sprint 5

NFR Nr.	Priority	Status
NFR1	low	NOK
NFR2	medium	NOK
NFR3	low	NOK
NFR4	high	NOK (Task ToDo: SPON-217)
NFR5	low	NOK
NFR6	low	NOK
NFR7	high	In Progress (Task ToDo: SPON-61)
NFR8	medium	NOK
NFR9	low	NOK
NFR10	medium	NOK
NFR11	high	OK: Verified on 02.05.2023. Used tool: Google Chrome Developer Tools (Security) Output: The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM. Additional info: automatic redirect from http to https Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR12	high	OK: Verified on 02.05.2023. Used tool: Google Chrome Certificate Viewer Output: Issued by: Organisation (O) Let's Encrypt Additional info: no certificate warnings Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR13	high	OK: Verified on 02.05.2023. Used tool: SwaggerUI + Visual Studio Output: Valid response in JSON format
NFR14	high	OK: Verified on 02.05.2023. Used tool: Android / Apple Smartphone and Windows Computer (Browsers: Google Chrome, Edge, Brave, Safari) Output: Working app
NFR15	low	NOK

Review NFRs - Sprint 6

Table 1.20: Tracking of the NFRS - Sprint 6

NFR Nr.	Priority	Status
NFR1	low	NOK
NFR2	medium	NOK
NFR3	low	NOK
NFR4	high	OK: Verified on 18.05.2023 A shard can lose one node, a router can lose a node, a config service can lose two nodes and the history still loads but when starting the config node again it needed two nodes up the backend can lose a node (basically the situation in PRD) the frontend can lose a node (basically the situation in PRD)
NFR5	low	NOK
NFR6	low	NOK
NFR7	high	OK: Verified on 18.05.2023. Used tool: Google Chrome + Google Account Output: It is possible to login with a Google Account. Additional info: the Google account mirio.eggmann@ost.ch was used to login.
NFR8	medium	NOK
NFR9	low	NOK
NFR10	medium	NOK
NFR11	high	OK: Verified on 18.05.2023. Used tool: Google Chrome Developer Tools (Security) Output: The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM. Additional info: automatic redirect from http to https Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR12	high	OK: Verified on 18.05.2023. Used tool: Google Chrome Certificate Viewer Output: Issued by: Organisation (O) Let's Encrypt Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR13	high	OK: Verified on 18.05.2023. Used tool: SwaggerUI + Visual Studio Output: Valid response in JSON format
NFR14	high	Mostly OK: Verified on 18.05.2023. Used tool: Android / Apple Smartphone and Mac (Browsers: Google Chrome, Firefox, Safari) Output: Responsive and working app in all screen sizes. Additional info: There is a little display bug in the mobile screen size as reported in issue: SPON-243
NFR15	low	NOK

Review NFRs - Sprint 7

Table 1.21: Tracking of the NFRS - Sprint 7

NFR Nr.	Priority	Status
NFR1	low	NOK
NFR2	medium	OK: Verified on 08.06.2023 Manually powered of the entire Kubernetes cluster in the Google Cloud Console. After three minutes the cluster and application were up and running again.
NFR3	low	NOK
NFR4	high	OK: Verified on 08.06.2023 A shard can lose one node, a router can lose a node, a config service can lose 2 nodes and the history still loads, but when starting the config node again, it needed 2 nodes up the backend can lose a node (basically the situation in PRD) the frontend can lose a node (basically the situation in PRD)
NFR5	low	NOK
NFR6	low	NOK
NFR7	high	OK: Verified on 08.06.2023. Used tool: Google Chrome + Google Account Output: It is possible to login with a Google Account. Additional info: the Google account mirio.eggmann@ost.ch was used to login.
NFR8	medium	OK: Verified on 08.06.2023. Tried to enter a wrong value for the radius in the frontend. User feedback given: Radius (3343) invalid! Must be between 1 and 1000!
NFR9	low	NOK
NFR10	medium	NOK
NFR11	high	OK: Verified on 08.06.2023. Used tool: Google Chrome Developer Tools (Security) Output: The connection to this site is encrypted and authenticated using TLS 1.3, X25519, and AES_128_GCM. Additional info: automatic redirect from http to https Tested sites: - spontaneous.ch - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR12	high	OK: Verified on 08.06.2023. Used tool: Google Chrome Certificate Viewer Output: Issued by: Organisation (O) Let's Encrypt Tested sites: - dev.spontaneous.ch - api.dev.spontaneous.ch
NFR13	high	OK: Verified on 08.06.2023. Used tool: SwaggerUI + Visual Studio Output: Valid response in JSON format
NFR14	high	OK: Verified on 08.06.2023. Used tool: Android / Apple Smartphone and Mac (Browsers: Google Chrome, Firefox, Safari) Output: Responsive and working app in all screen sizes.
NFR15	low	NOK

Chapter 2

Domain Analysis

2.1 Domain model

Very low priority stories are not displayed in the domain model. If they later rise in priority, the domain model is being updated.

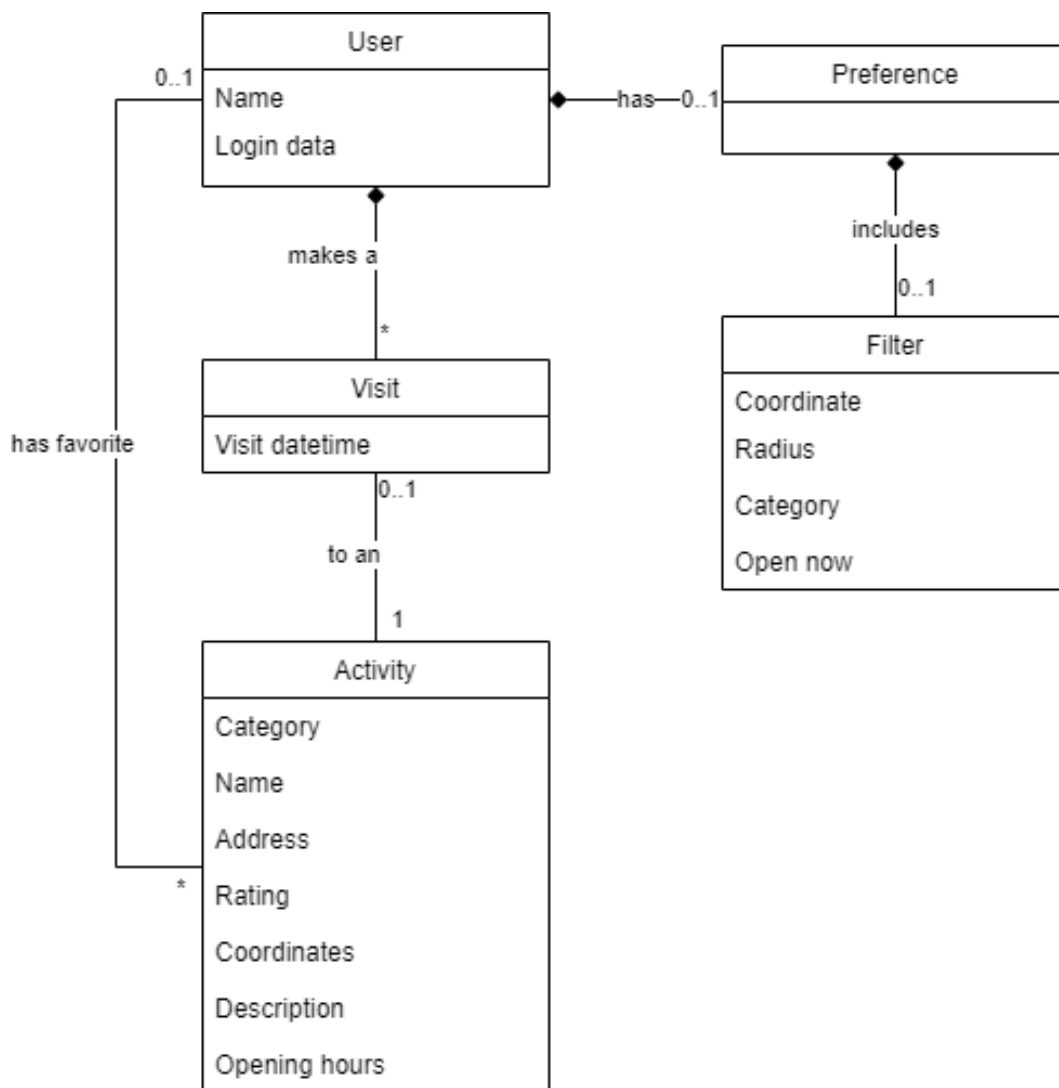


Figure 2.1: Domain model

2.1.1 User

A user can mark several activities as favorites.

2.1.2 Visit

A visit is made by a user at a specific time. This represents the execution of an activity.

2.1.3 Activity

An activity is always generated anew and is the result when using the Spontaneous app. The reason for this is that activities are sourced from an external entity. However, we want to freeze the data from the activity at the time it gets proposed so that it always stays the same and doesn't change if the external source changes it. Because of that (even if this activity has already been generated once), there can only be one visit to an activity.

It does not have to be marked as a favorite. Furthermore, an activity can also be suggested without a user being logged in. This results in the fact that no visit can be made either.

2.1.4 Preference

A user's preferences include information that is necessary to provide a specific user with individual setting options, such as preset filter options for generating activities.

2.1.5 Filter

Filters are used to better find activities. They are saved per user as their preference.

Chapter 3

Design

3.1 UI Mockups

To visualize the arrangement of the most important parts of our application, interface sketches were designed using the tool [Figma](#). The sketches were designed before the implementation of the application and are therefore not 100% accurate.

If a user is logged in, the "Login" and "Sign Up" button in the navigation on the right will disappear. Instead, his username will be shown. An idea would be to remove the "Account" button and integrate the navigation to the user account page with the displayed username.

3.1.1 Home Screen

This screen represents the landing page. A user can set filters and then click the "Go" button to get an activity proposed.

The mockup shows the home screen of the 'BE SPONTANEOUS' application. At the top is a navigation bar with the logo, 'Favorites', 'History', 'Account', 'Help', 'Login', and a 'Sign Up' button. The main heading is 'Are you Spontaneous?'. Below this are four filter boxes: 'Category' (with a dropdown set to 'Sport'), 'Price Range' (with four red dollar signs), 'Location' (with a 'Current Location' toggle and a 'Search near' input), and 'Radius' (with a 'km' input). There are also 'Opening Hours' (with an 'Open Now' toggle and an 'Open at' input) and 'Lorem Ipsum' (with two checkboxes). A large black 'GO' button is centered below the filters. The footer contains the 'SPONTANEOUS' logo, copyright information '© 2023 - 2023', and the 'OST Ostbayerische Fachhochschule' logo.

Figure 3.1: Home Screen Mockup

3.1.2 Activity Screen

This is the result page where a proposed activity gets displayed. The relevant info is displayed in boxes, e.g. the opening hours or the location of the place.

On the top left, an icon for the chosen category is shown. On the top right, there are different action buttons: A "Did it" button one can use to mark the activity as done, an "Add to Favorites" button and a sharing button to send a link of the activity to a friend.

Optionally, the place also gets marked on a map. Also optional is the displaying of the "match" box, where the previously set filters are compared with the result.

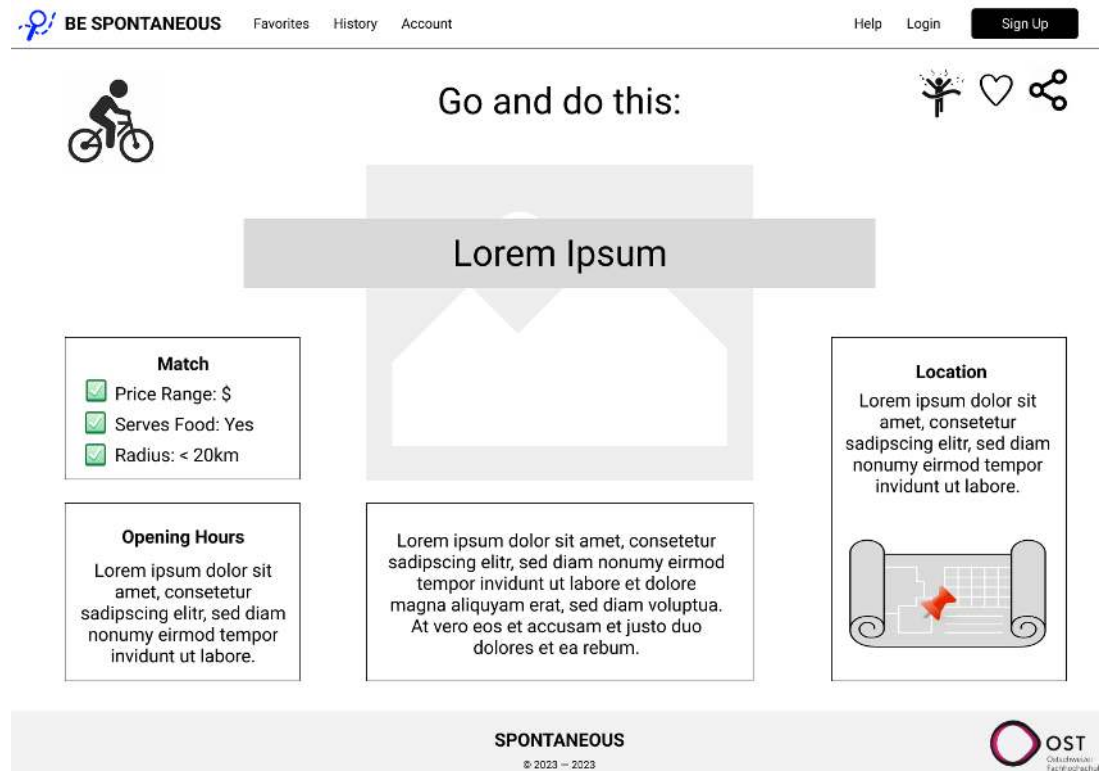


Figure 3.2: Activity Screen Mockup

3.1.3 Favorites Screen

Activities marked as favorites are displayed here, optionally also marked on a map. With a click on the card one gets routed to the corresponding activity screen.

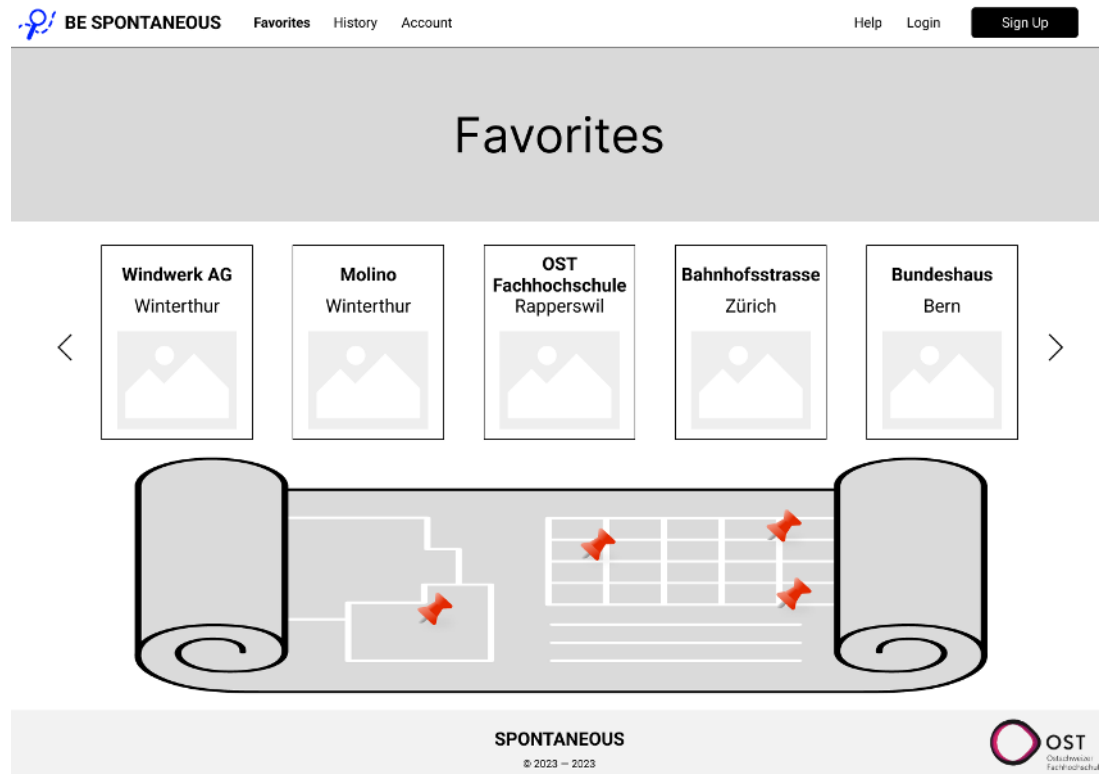


Figure 3.3: Favorites Screen Mockup

3.1.4 History Screen

On the history screen, a user can see all past proposed activities he got when logged in. With a click on an entry in the table he gets routed to the corresponding activity screen.

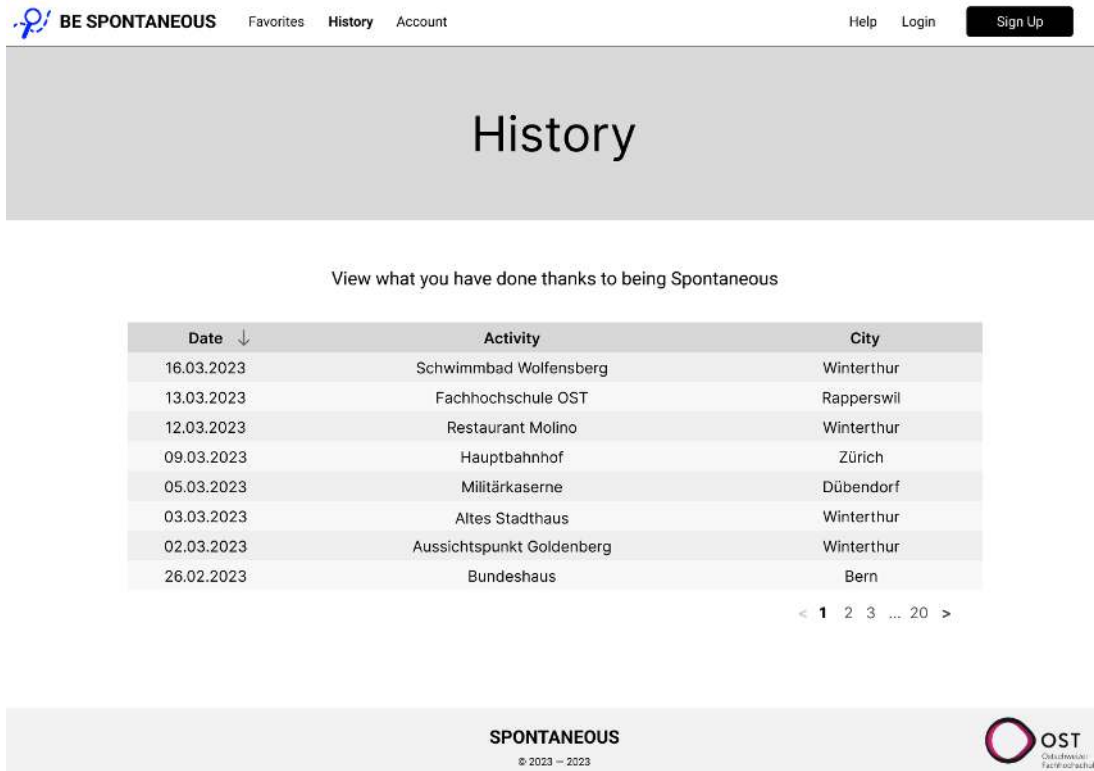


Figure 3.4: History Screen Mockup

3.1.5 Account Screen

A user can set preferences here that are automatically applied on every request. Additionally, infos like the email address of the user are displayed.

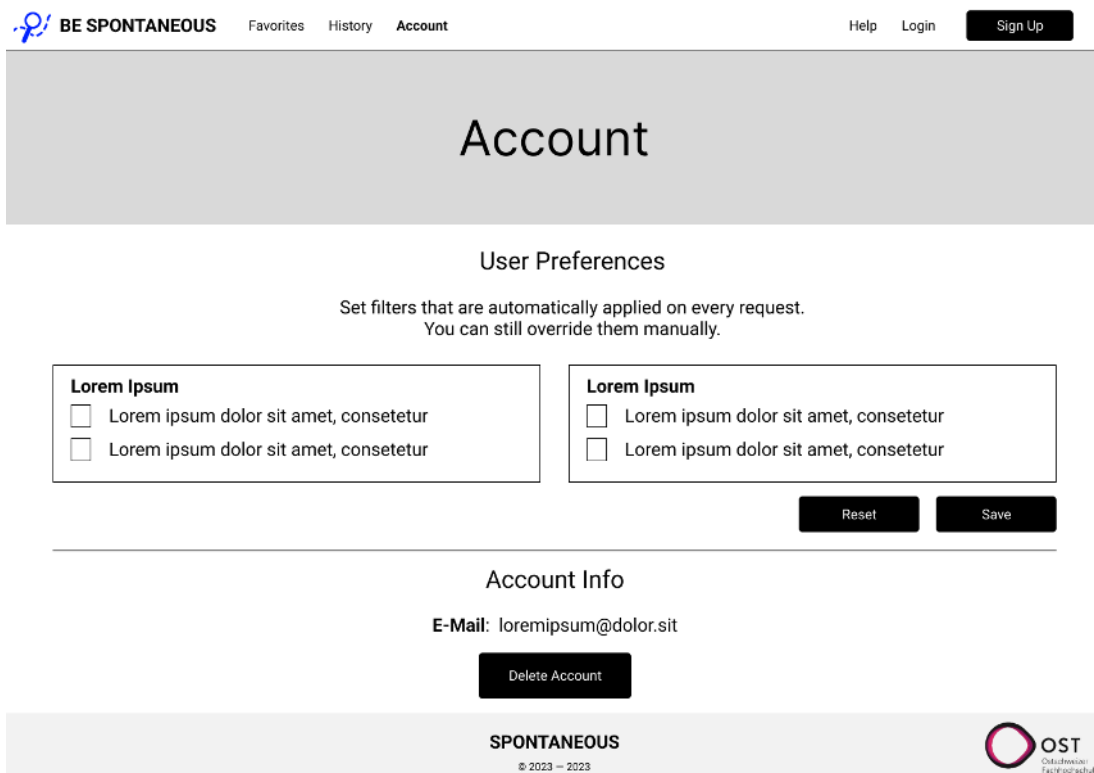


Figure 3.5: Account Screen Mockup

3.2 Color Scheme

We use these colors: [Coolor Scheme](#)

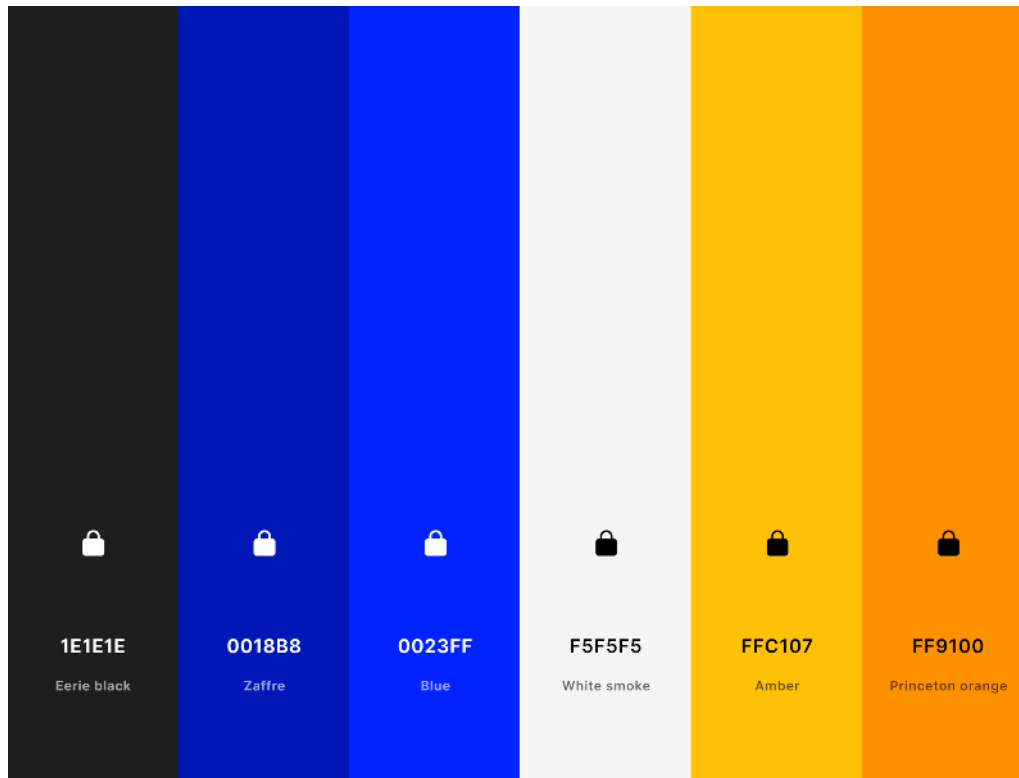


Figure 3.6: Color Scheme

Eerie black: 1e1e1e, Zaffre: 0018b8, Blue: 0023FF, White smoke: f5f5f5, Amber: ffc107, Princeton orange: ff9100

Logo



Figure 3.7: Logo

Chapter 4

Architecture

Our [Miro board](#) contains all architecture diagrams.

4.1 Architecture Diagrams C4

The architecture diagrams are based on the C4 standard. However, we only defined the context and container diagram, because the components and code are too volatile and better described by the architecture pattern above.

4.1.1 C4 Context Diagram

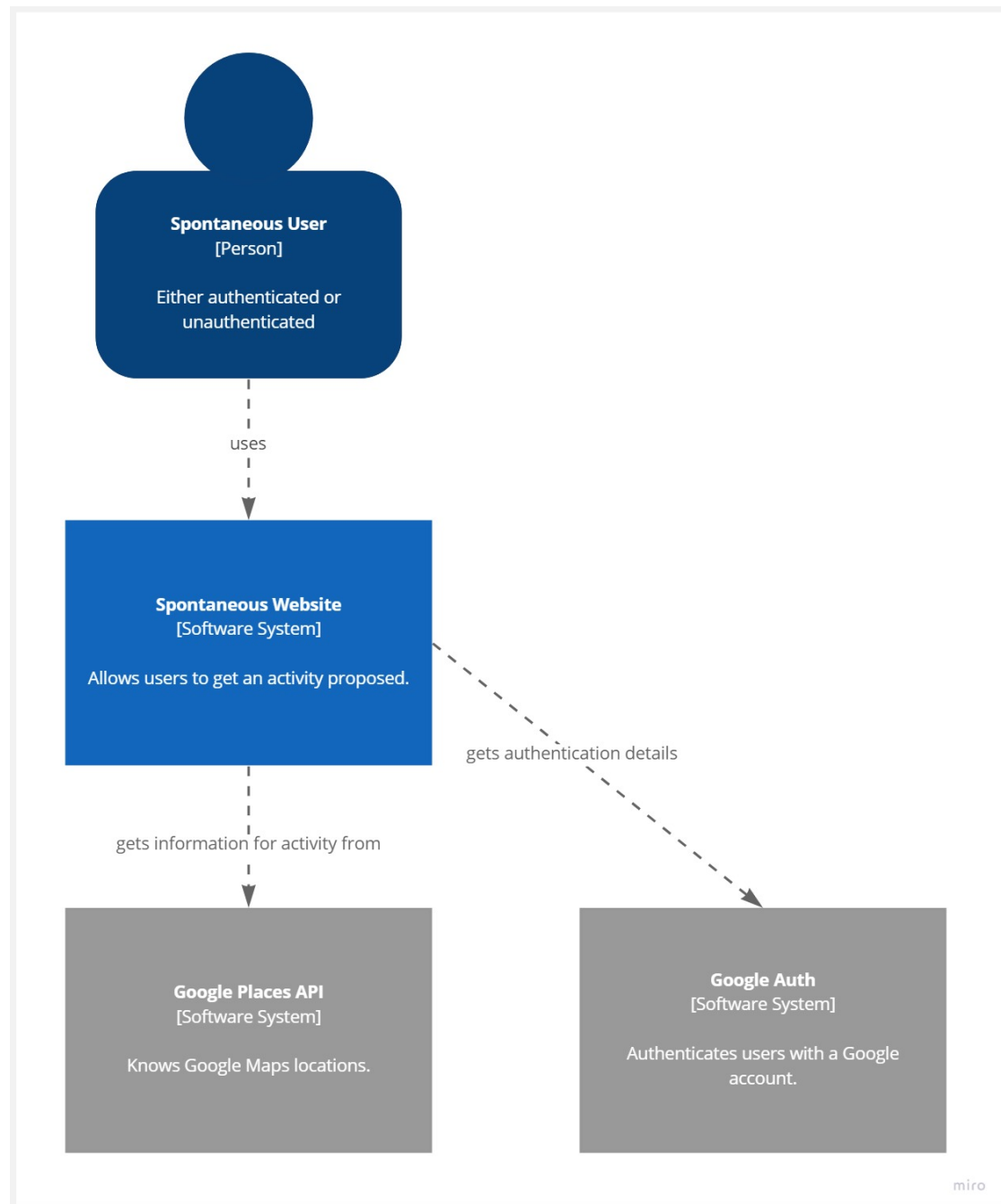


Figure 4.1: C4 Context Diagram

The user doesn't have a direct line to the Google auth software system, because it isn't necessary for our context.

4.1.2 C4 Container Diagram

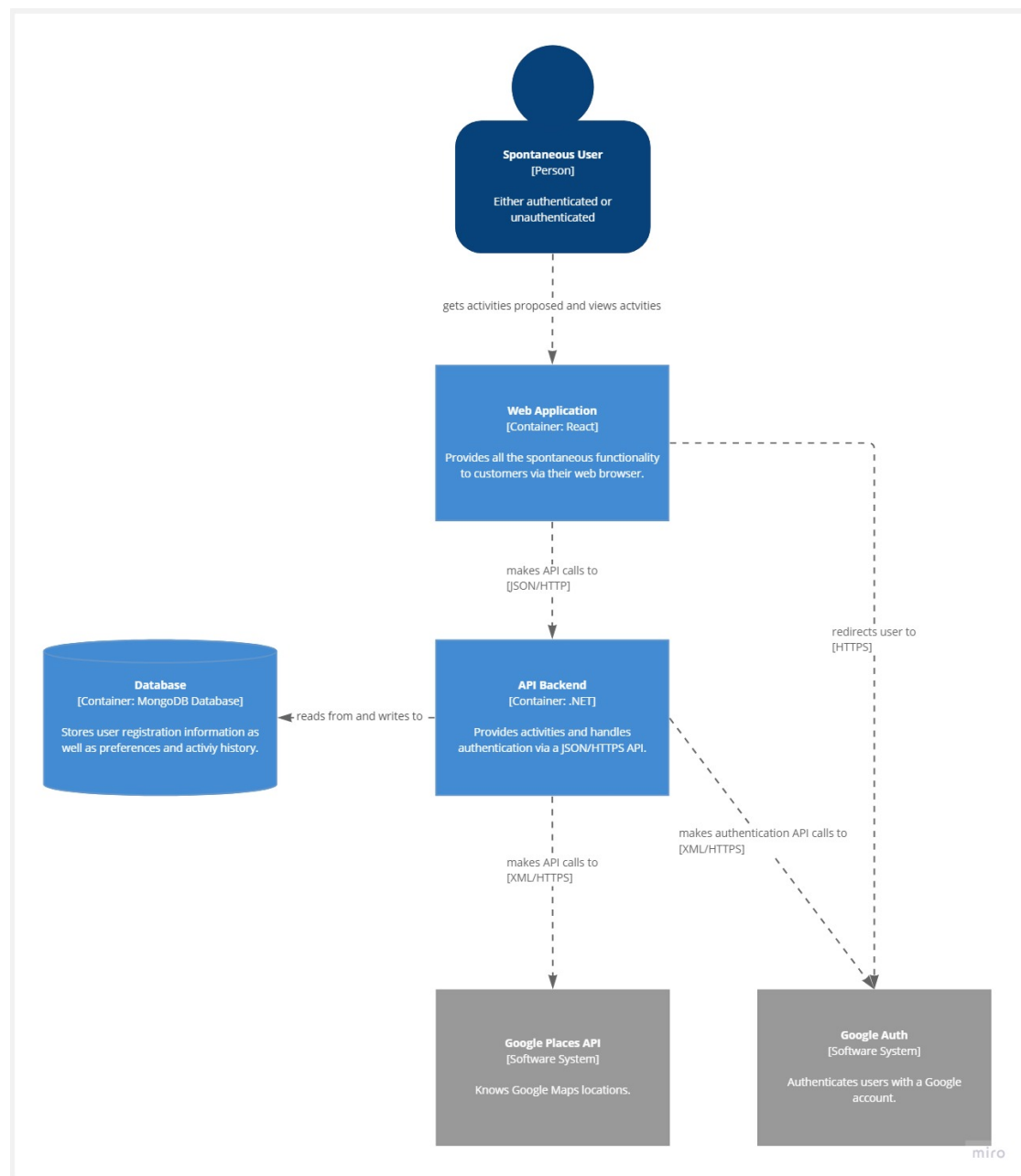


Figure 4.2: C4 Container Diagram

4.2 Sequence Diagrams

The following diagrams show a dynamic view to the most important features.

4.2.1 Activity Proposal Sequence Diagram

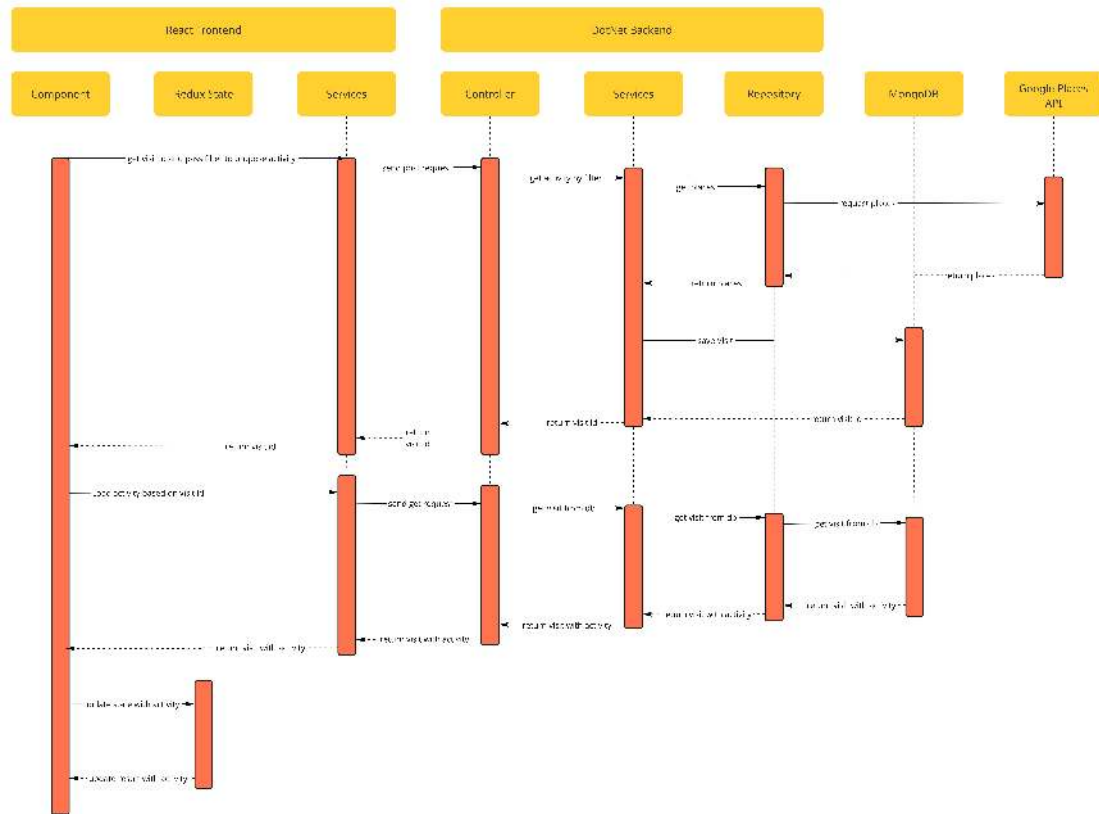


Figure 4.3: Activity Proposal Sequence Diagram

Loading places from Google via service and repository is actually a bit more complex than shown here. However, this diagram is not intended to explain the algorithm but to show roughly which layers the request goes through.

4.2.2 Authentication Sequence Diagram

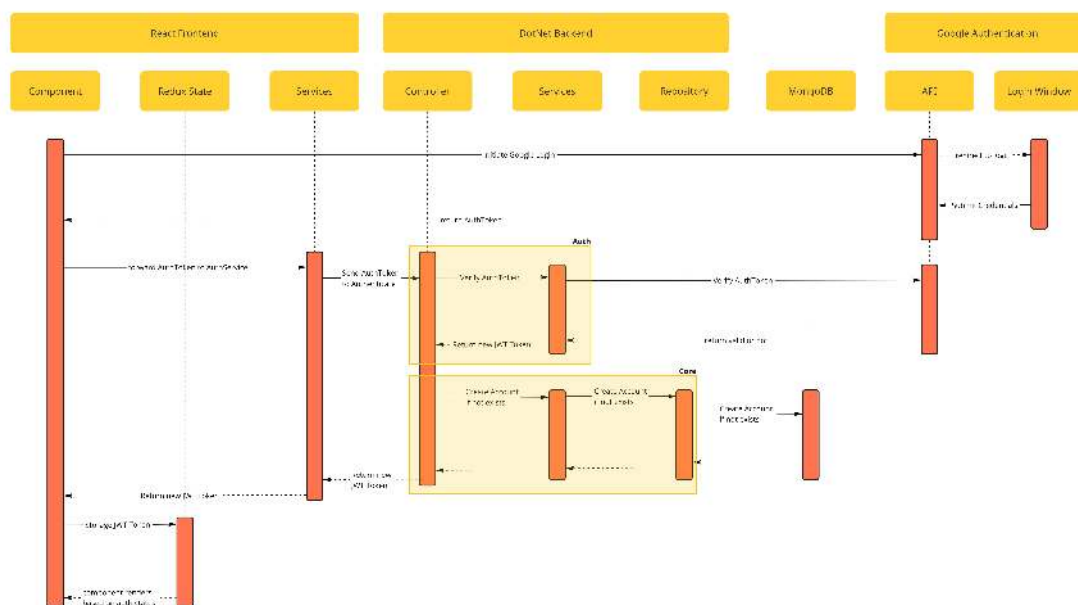


Figure 4.4: Authentication Sequence Diagram

4.3 Backend Architecture

4.3.1 Architectural Pattern

The architecture is built based on the Clean Architecture pattern. In contrast to the [original structure of Robert C. Martin](#), we have merged the entities and use cases layers, since the size of the project does not require such a separation of enterprise and application.

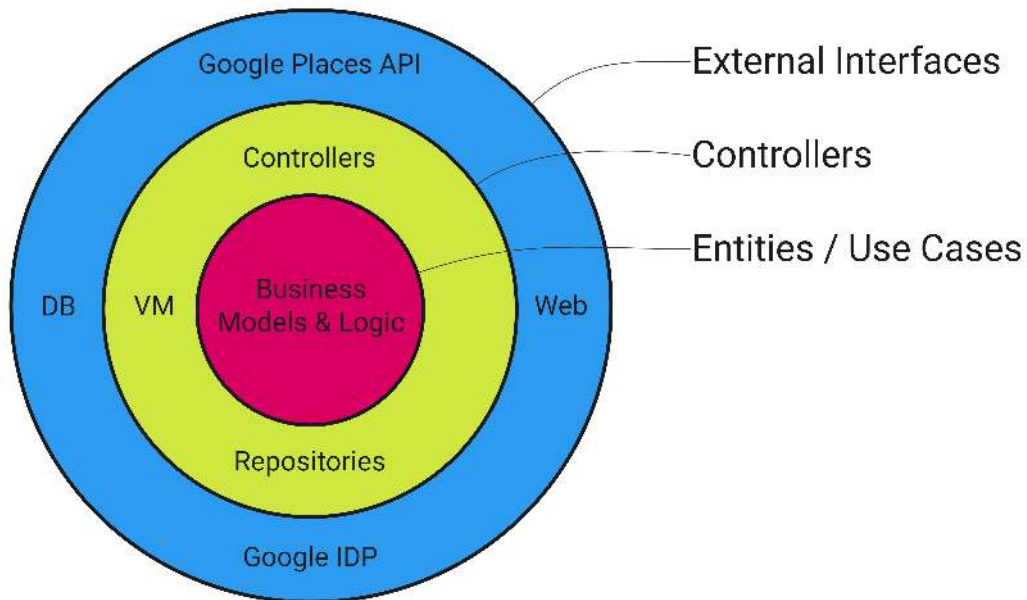


Figure 4.5: The Clean Architecture Approach

The abbreviation VM stands for View Models.

4.3.2 Architectural structure

The backend is developed with the programming language C# in the version .NET 7. It can be found in the "Spontaneous.sln" solution in the "Backend" folder. The solution is divided into different projects. This makes the individual components easier to change and replace with newer technologies. In this way, the dependencies are clearly defined.

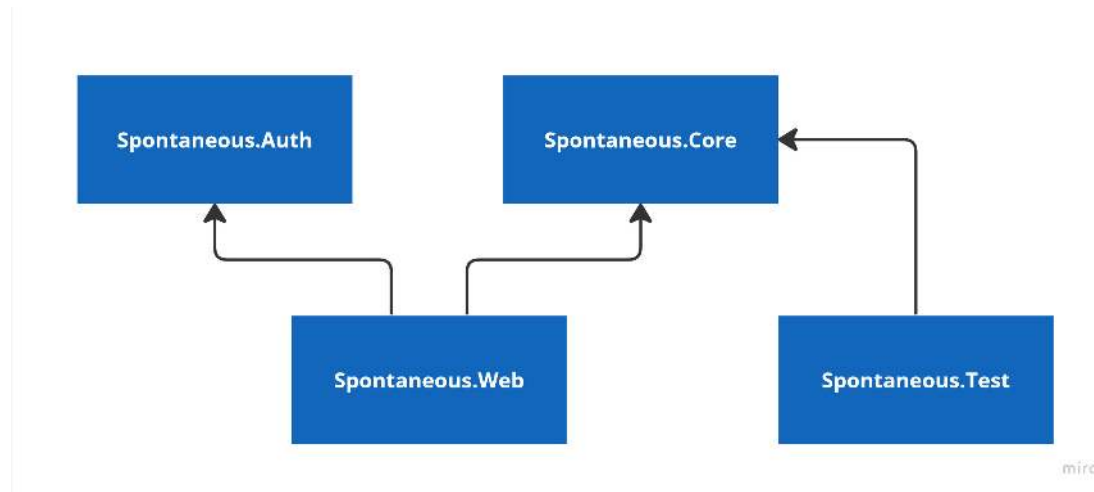


Figure 4.6: Backend Project Structure

Core

The core project contains most of the code. This includes the essential logic to make the application work, which is handled in so-called "services". Additionally, database access and requests to the Google Places API are found here. These are managed in so-called "repositories". Repositories differ from services in the way that they are only responsible for fetching data in its original form. The processing then takes place in the services. Business objects are modeled here.

We deliberately decided against another project with only database accesses, as this would have added more overhead than it would have benefited in the context of our SE project. That's why database accesses are integrated in the core project in the form of repositories.

Auth

The Auth project includes all the authorization and authentication mechanisms.

Web

The API which communicates with the frontend is in the project "Web". This contains only little code which is responsible for the mentioned communication. Business objects are converted into view models here.

Test

In the "Test" project, XUnit is used to cover important elements of the "Core" project using unit tests.

4.4 Frontend Architecture

In the frontend, we use React.js. This diagram show the general structure of the frontend. We simplified the state because Redux manages the state, see this [Redux Architecture](#). By state we mean the data that a component uses and updates through hooks.

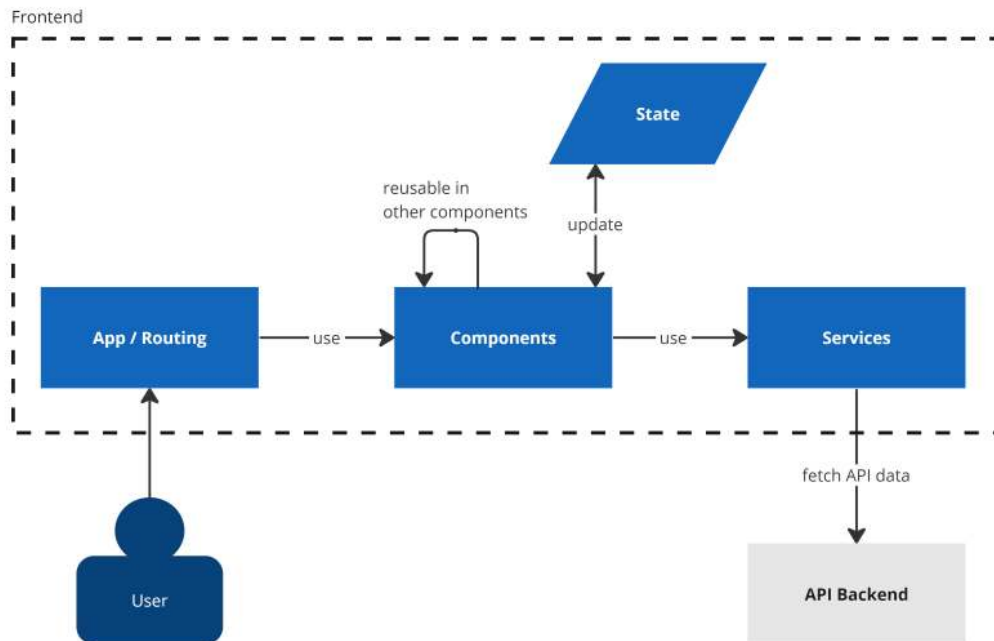


Figure 4.7: Frontend Architecture

We don't have a clean architecture diagram for the frontend because we are bound to the technology architecture.

4.5 Pipeline / Deployment Architecture

This diagram only shows the jobs required for the deployment. Other jobs for example testing and linting are not concluded in this diagram.

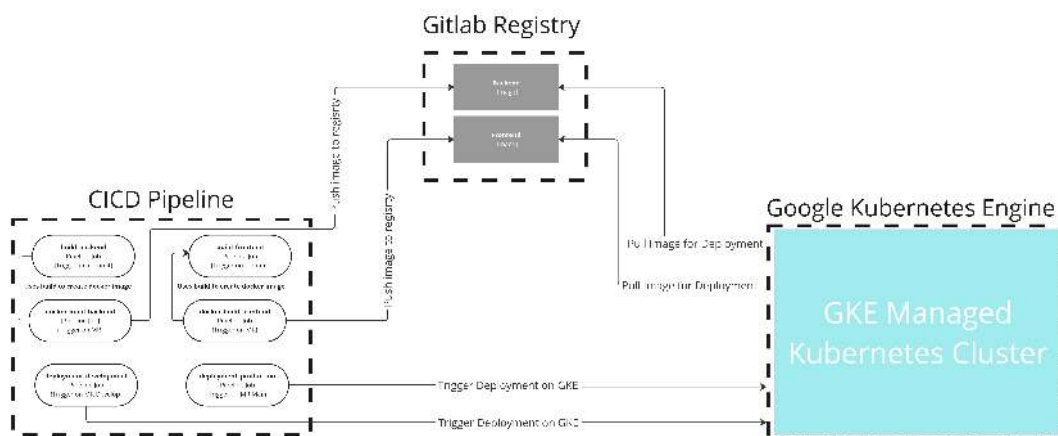


Figure 4.8: Pipeline and Deployment Architecture

4.6 Infrastructure Architecture

Following, the Kubernetes deployment is visualized. It concludes all important components which are necessary for our application. The Kubernetes deployment runs on the Google Kubernetes Engine. Since it's a managed cluster, the architecture of the cluster itself is not relevant and as well not visualized further.

4.6.1 Database

The database deployment is quite complex, since it's a shared database which is scalable and high available. Since drawing all the instances on the diagram would overfill it, we pulled together the same instance types and name it with the count-suffix "01-03". The whole sharded database consists of following instances:

- 2x database router: Responsible for receiving requests and redirecting it to right shard.
- 3x config servers: Store the information about which shard is stored on which nodes.
- 3x nodes for shard01: Store the actual data for the specific shard01.
- 3x nodes for shard02: Store the actual data for the specific shard02.

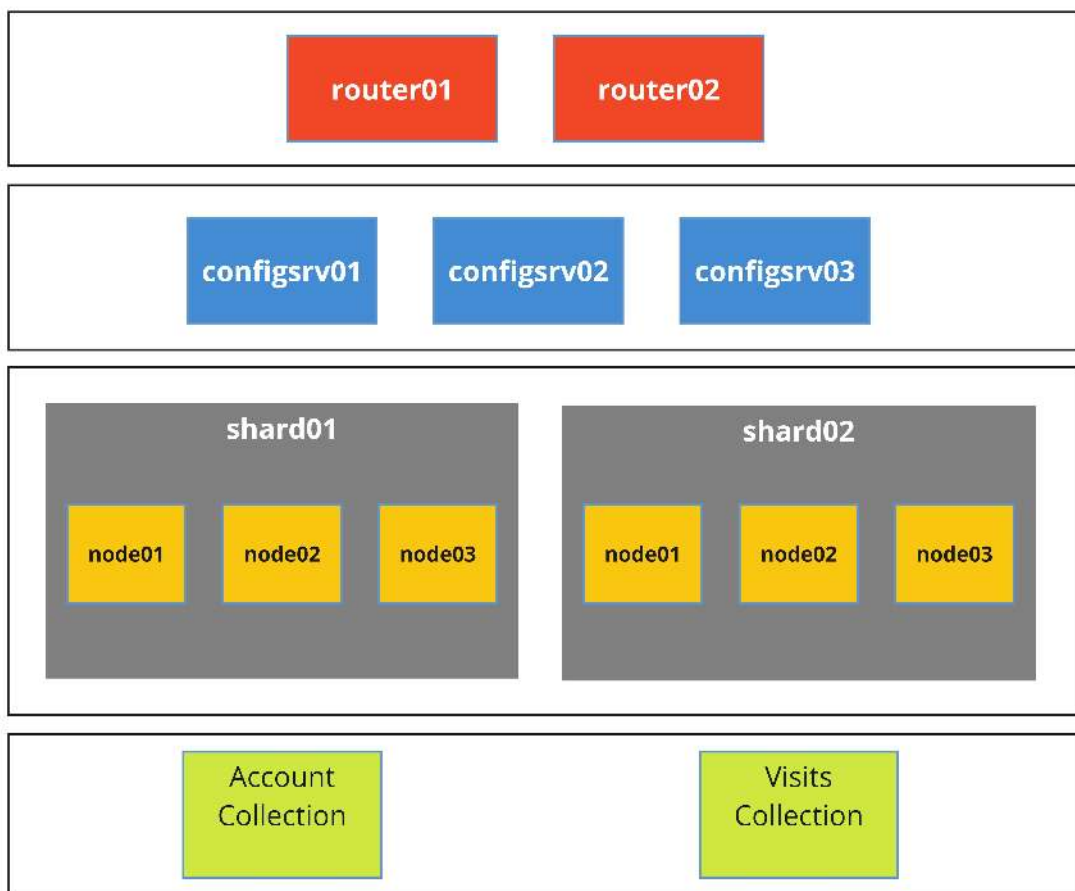


Figure 4.9: Sharded MongoDB Setup

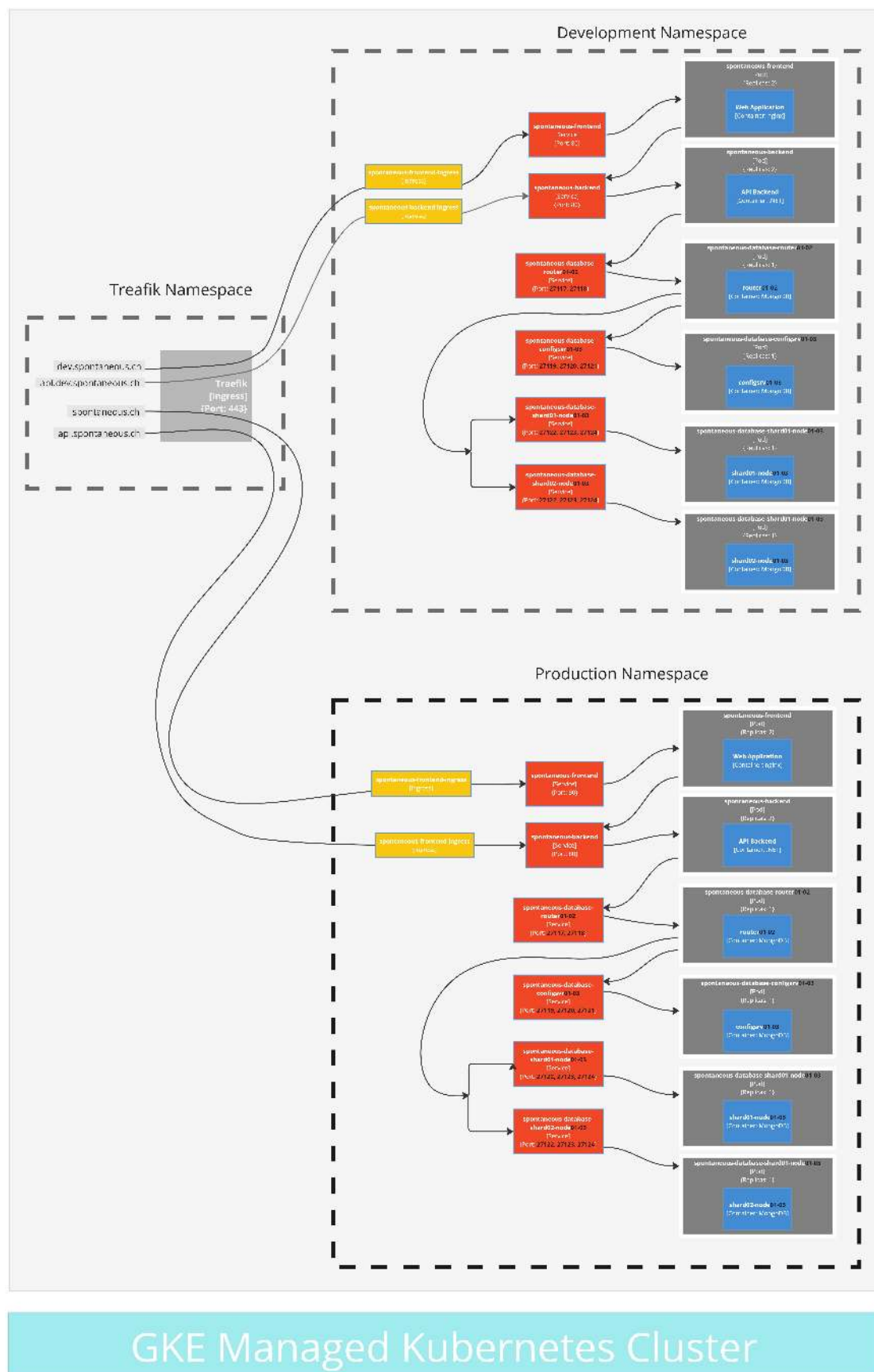


Figure 4.10: Infrastructure View

4.7 Technology Decisions

Most technologies were chosen because we had the most experience with them and they were the most popular and therefore best documented.

- We use a lot of different technologies, so we don't have to reinvent the wheel and can focus on the core features of our application.

4.7.1 Evaluation Dev Stories

For the most important decisions that weren't already decided in the project proposal we did evaluation user stories under the epic: [Dev-Evaluate-Technology](#). Those stories evaluate if planned technologies really deliver our needed functionality and can be integrated with spontaneous.

- [SPON-50 Evaluate Database](#)
- [SPON-51 Evaluate Maps API](#)
- [SPON-52 Evaluate Test Strategy](#)
- [SPON-53 Evaluate Google Identity Provider and JWT](#)

4.7.2 Frontend

- **Typescript**, we simply wanted types and use JavaScript in a more structured and clean way.

- **React**, best known by us and most popular framework.

Alternatives: Angular, VueJS, .Net MAUI, Blazor

- **Sass**, we wanted to use a CSS preprocessor for better code reuse and better readability.

- **Material UI**, we wanted to use a UI framework for faster development and better design and responsiveness.

Alternative: Bootstrap

- **Webpack**, used for bundling and minifying the frontend files.
- **NPM**, used for managing frontend dependencies.
- **Axios**, used for making HTTP requests to the backend.
- **Redux**, used to store state used in different components.
- **Analytics**, if we would go live we would use Piwik or Google analytics for analytics and improving usability.

4.7.3 Backend

- **.Net 7**, ASP .NET Core Web API, we wanted to use a framework which is well documented and has a lot of support.

Alternatives: Java, NodeJS

- **Swagger**, is really useful for testing and documenting the API.
- **MongoDB lib** (mongodb.bson), nuget package for accessing MongoDB.
- **Google API lib**, nuget package for Google API models.

4.7.4 Infrastructure

- **Traefik**, we use Traefik for load balancing and proxying. We didn't use Nginx because we wanted to use HTTP/3 which isn't supported by Nginx yet.

Alternatives: Nginx, Caddy, HAProxy, Envoy

- **Nginx**, used as a web server for hosting the static frontend files.
- **Kubernetes**, used for container orchestration and scaling.
- **Helm**, used in Kubernetes for managing applications.
- **Cert Manager**, used for managing certificates for HTTPS in our cluster.
- **Let's Encrypt**, used for getting free certificates for HTTPS.
- **Google Cloud**, was the most convenient and cheapest option for hosting our services and we knew it best.

Alternatives would have been: (AWS, Azure).

- **Cloudflare**, used as our DNS provider.
- **Hostpoint**, is where we bought our domain.
- **Docker**, used for containerization and for local development.

Possible technologies for future use:

- **Terraform**, we might use Terraform for infrastructure deployments if we need to change the platform often.
- **Redis**, could be used later for better scaling and performance by using caching.
- **Kafka**, if we would do microservices or use Postgres we would use Kafka for communication between services and writing to the sharded database.

4.7.5 Pipeline

- **Gitlab CodeQuality**, used for checking backend code quality (running in the pipeline). Not used for the frontend because it doesn't work with our ESLint config.

Alternative: SonarQube, but it is not free and a big effort to integrate with a private repository, but we can use SonarLint locally in the IDE.

- **ESLint**, used for checking frontend code quality in the pipeline and locally. We had the most experience with it compared to others.

Alternatives: StyleCop, Prettier

4.7.6 Database

- **MongoDB**, easiest to use on multiple nodes and offers everything we need.

Alternatives: CosmosDB, Postgres, Redis. More complicated on multiple nodes.

4.7.7 API's

- **Google Places API**, Google has the best data and we fit in their free tier.

Alternative: OpenStreetMap

- **Google Identity**, Google Auth is secure, needs less coding and is easier for the user.

Alternative: Basic Auth with JWT (we would need to implement it ourselves)

4.7.8 Testing

- **Cypress**, easiest to use on multiple nodes and offers everything we need.

Alternative: Selenium.

- **XUnit tests**, used for testing the backend.
- **Mock**, used for mocking in the backend.

4.7.9 Tools

- **Jira**, we choose Jira because of it's popularity and because more of us know it rather than YouTrack.

Alternatives: YouTrack, Trello, GitLab (was too function limited)

- **Confluence**, because we use Jira we can reference Jira tickets in Confluence and vice versa.
- **Miro**, free and perfect for creating diagrams together.
- **Figma**, de facto standard for UI design.
- **GitHub Copilot**, optionally used for quicker coding.
- **Overleaf**, optionally used for writing LaTeX quicker.
- Various **source control tools** like: Git Bash, GitKraken, SourceTree

4.8 Tech Stack

This is a visualisation of our used technologies.

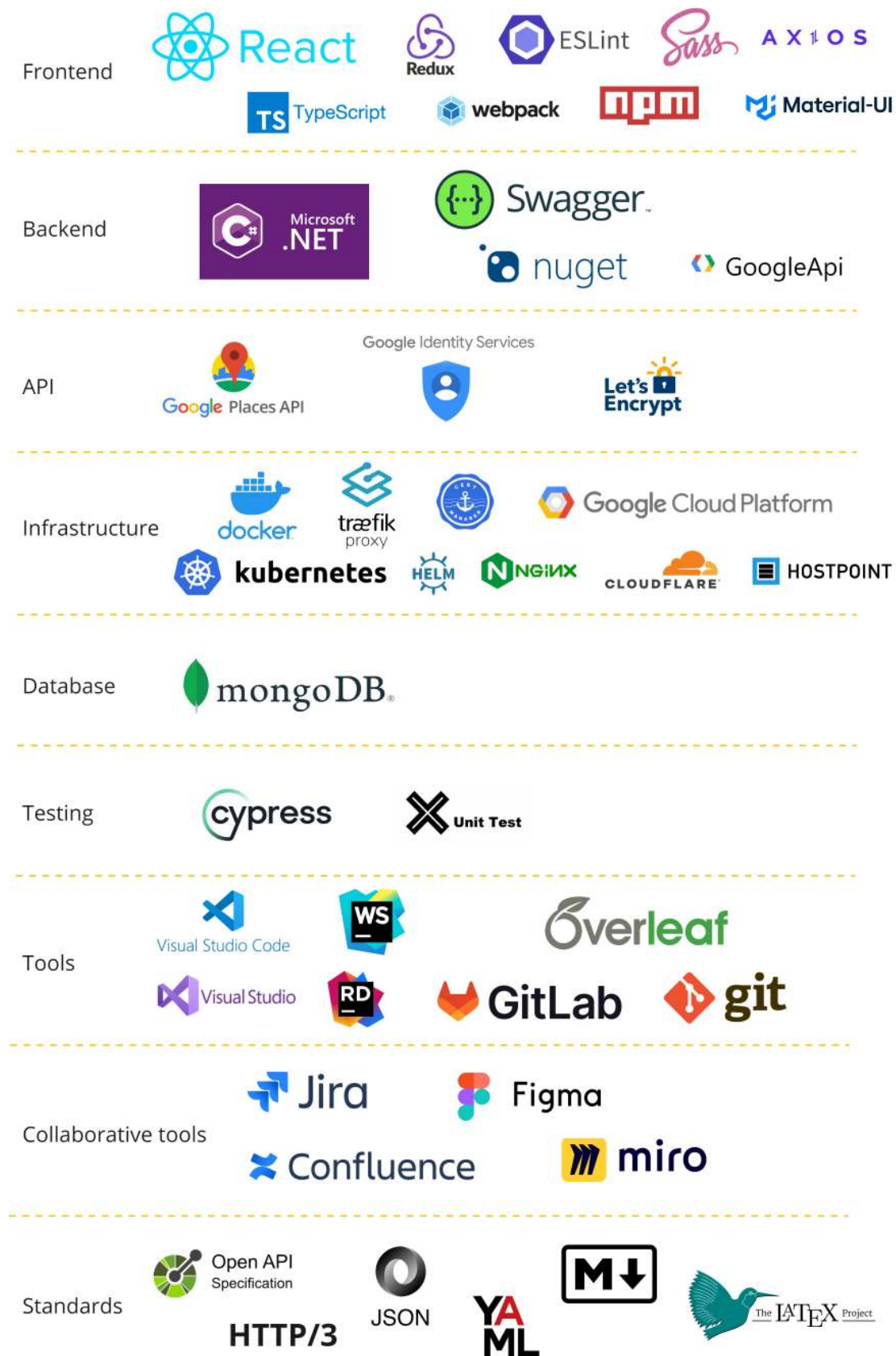


Figure 4.11: Tech Stack

Chapter 5

Quality Measures and Project Guidelines

In order to keep quality to our software we use the following measures.

5.1 Code

- To enforce clean and proper formatted code, we use linter: SonarLint in the backend and eslint in the frontend. These should be integrated in the IDE by the developer.
- Eslint is additionally executed in the pipeline.
- The linters ensure the standard recommended code metrics.
- We don't use SonarCube, since the overhead would not be in proportion to the project size and project importance.

5.2 Documentation

- Keep files with a readable amount of lines and split files only where it makes sense.
- Use LaTeX whenever possible, instead of using pictures.

5.3 Jira

We track all our time on to Jira issues, see the time tracking report [56](#).

- Keep the fields and references in issues up to date, so that everybody sees the current state of progress.
- Never book time on epics.
- Definition of done means no open branches, formatted for hand in, no more work logs.

5.4 Gitlab

[Spontaneous GitLab](#) is where we keep our repository versioned and run our pipeline.

5.4.1 CI/CD

- Never update the pipeline on develop, always use a branch to test it.
- The pipeline is meant for building, testing and deploying the application.

5.5 Git Workflow

This section describes how to work with git in our project.

- Timed by the reviews we merge from develop to master.
- Keep temporary local files and build files out of the repository via .gitignore.
- Adhere to the branch naming.
- Use the Jira issue id's whenever possible.

5.5.1 Branching

- Always work on branches, branched by default from the develop branch.
- Use the Jira issue id in the name to link it.

Name your branches in the following ways:

- feature/SPON-40-update-UI
- documentation/SPON-12-add-docu
- bugfix/SPON-50-fix-typo

Follow the branching from this diagram:

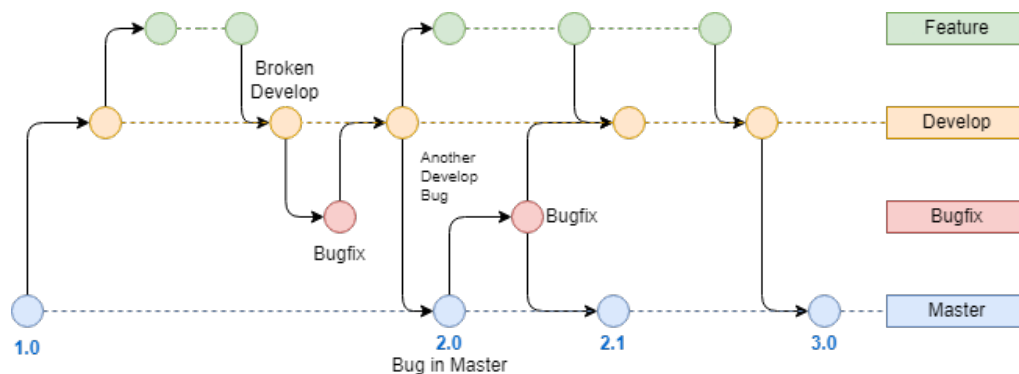


Figure 5.1: Git Branching Workflow

5.5.2 Merge requests

- Small changes in configs, code or documentation can be merged without a reviewer.
- Other changes should be reviewed by one other developer from the team.
- Regularly merge new changes from the develop branch into your branch, to avoid having one big merge conflict upon the merge request.
- Merge your branch frequently to develop so others have a up to date state and so that the merge requests are quicker and easier reviewed.
- Only create merges from develop to master if develop is stable (as if we would hand in our software and documentation).

5.6 Test Strategy

5.6.1 Functional Testing

The functionality of the software is tested on two different levels: Unit tests for specific code logic and system tests for the interaction between the different software components.

There is no test coverage goal, but every important part should be covered by an appropriate test. The test should be created before, in parallel or shortly after the development of the corresponding feature, but certainly before opening a merge request. Since this project isn't critical and it wouldn't have any impact if the deployed version did have errors, the focus doesn't lie on picky testing.

The tests are automatically executed by the pipeline on every merge request to GitLab, except for Cypress tests. Each developer should verify that all tests are passed before opening a merge request.

Unit Testing

Unit testing is done with the open source tool [xUnit.net](#). When the tests are executed in the pipeline, a report is generated. It can be found as an artifact.

System Testing

System tests are performed using the open source web application [cypress](#). Since cypress tests against a deployed instance, the tests are only executed once a merge request is actually merged. It lies in the responsibility of the developer to check if the pipeline succeeds after merging.

5.6.2 Non-Functional Testing

Usability Testing

To test the usability of our app, two usability tests were performed. The goal is to get feedback and inputs in regards to the usability and gain an understanding of how intuitive the UI is to new users.

Both tests did take place with four different customers from our target group. This consists of rather young people from age 15-35, that are open-minded and willing to try new things. A member from the Spontaneous team performed the test with each person individually. The exact procedure can be taken from the following table. The person conducting the test wrote down the answers / remarks / feedback from the user and the results were analyzed in the following week.

The two tests were performed in the following weeks:

- calendar week 17
- calendar week 19

The performed tests are documented in the [wiki](#).

Test Protocol Usability Test 1

This test took place in calendar week 17 after the second draft was finished. In the table, S stands for the Spontaneous team member that conducts the test, and U for the test user.

Table 5.1: Test Protocol Usability Test #1

#	Who	What	Description
1	S	Setup	Give a laptop to the user with an open instance of Spontaneous, explain general procedure of the test
2	U	Landing Page	Take a look at the landing page. - Is it clear what the site is about? - Are you excited to continue being on this site? - What would you do next if you were on this site on your own?
3	S	Explanation	Explain what the goal of the Spontaneous application is and what functionality it has - Get random activity proposed according to your filters - Favorites - History - Login with Google
4	U	Activity Proposal	Get an activity proposed according to the following criterion: (don't show the user where to click) - Category: Sport - Opening Hours: Now open - Location: Rapperswil - Is this intuitive? - How long did it take the user to complete the task? - Did he make any mistakes?
5	U	Result Page	Take a look at the result page. - Is it clear what the site is about? - Do you find all information necessary? - Do you need more information? - What would you do next if you were on this site on your own?
6	U	Information	Tell your testing advisor the following information: - What is the name of the activity? - Where is it located? - When is it open? - What category does it belong to?

Test Protocol Usability Test 2

This test took place in calendar week 19 near the end of the construction phase.

Table 5.2: Test Protocol Usability Test #2

#	Who	What	Description
1	S	Setup	Give a laptop to the user with an open instance of Spontaneous, explain general procedure of the test
2	S	Explanation	Explain how the login with Google works in general and provide credentials for the user
3	U	Login	Login with Google with the credentials previously provided - Is this intuitive? - How long did it take the user to complete the task? - Did he make any mistakes? - General remarks about the login process?
4	S/U	Test 1	Repeat steps 2 - 6 as described in the testing protocol for the usability test 1
5	U	Buttons	Perform the following steps: (don't show the user where to click) - Add the shown activity to your favorites - Share the link to this activity with a friend - Is this intuitive? - How long did it take the user to complete the task? - Did he make any mistakes?
6	U	Favorites	Perform the following steps: (don't show the user where to click) - View all your favorite activities - How many activities are in your favorites? - What are the opening hours of one of your favorite activities? - Is this intuitive? - How long did it take the user to complete the task? - Did he make any mistakes? - General remarks about the favorites page?
7	U	History	Perform the following steps: (don't show the user where to click) - View the history of activities proposed to you - How many activities were proposed to you? - Show a picture of the activity of your last proposal. - Is this intuitive? - How long did it take the user to complete the task? - Did he make any mistakes? - General remarks about the history page?

Part III

Project Documentation

Chapter 6

Initial Project Proposal

Project name: Spontaneous

Team Members

1. Dario Glasl (dario.glasl@ost.ch)
2. Jasmin Fitz (jasmin.fitz@ost.ch)
3. Samuel Maissen (samuel.maissen@ost.ch)
4. Mirio Eggman (mirio.eggmann@ost.ch)
5. Nick Wallner (nick.wallner@ost.ch)

Availabilities

Time slot	Mon	Tue	Wed	Thu	Fri
08h00-09h00	-	(XO)	(XO)	-	(XO)
09h00-10h00	-	(XO)	(XO)	-	(XO)
10h00-11h00	-	-	(XO)	-	(XO)
11h00-12h00	-	-	(XO)	-	(XO)
12h00-13h00	(XO)	(XO)	(XO)	XR	(XO)
13h00-14h00	-	-	-	-	XO
14h00-15h00	-	-	-	-	XO
15h00-16h00	-	-	XO	XR	XO
16h00-17h00	-	-	XO	XR	XO
17h00-18h00	XO	-	XO	(XR)	XO
18h00-19h00	XO	-	XO	(XR)	XO

ALL time slots during which every team member can attend review meetings.

XO	Slot available online
XR	Slot available in Rapperswil
(XO)	Slot available online, but not optimal
(XR)	Slot available in Rapperswil, but not optimal
-	Slot not available

Having a slot available on campus automatically implies that this slot is also available online. The more slots that are available, the more likely it is that you will get an optimal slot and coach assignment. You are required to have a minimum of 5 slots available during regular working hours (08h-12h, 13h-17h) to be eligible to form a team.

Vision

The app is called "Spontaneous" and it helps users find fun things to do and nice restaurants on the fly. With a simple interface some basic criterias can be choosen, in order to get a random propose for an activity or a place to eat. The app should make it easy to try something new with the condition the user aswell using it as considered.

Proposed Realisation

We plan to create a web application with a C# .NET backend and a TypeScript (+ React library) frontend. For source control and CI/CD we use Gitlab (CI). The backend of the web application should be running on Kubernetes - optimally on a cluster in GCP/AWS or if not allowed maybe on a small bare-metal-cluster that we setup on a provided virtual server of the vkiosk.

Chapter 7

Project Plan

7.1 Project Management

7.1.1 Project Management Method

We decided to choose SCRUM+ as our project management method since it suits the best for our use-case as well as it is recommended by our lecturer. SCRUM+ is a combination of SCRUM with Rational Unified Process (RUP). RUP is used for the long-term planning, whereas SCRUM is used for short-term planning in iterations.

7.1.2 Project Meetings

Table 7.1: Project Meetings

Type	Description	Periodicity
Weekly	Is held every Thursday between 15:00 and 17:00 (15 minutes) to discuss about the current state of the project and brainstorm together about problems and fundamental decisions.	Weekly
Retro / Planning / Review	Discussion what went well or badly, why that was and what could have been done better in the past sprint. Define which issues to take from the backlog for the next sprint. Takes place every second Thursday between 15:00 and 17:00 (2 hours) at iteration end.	every second Thursday
Meeting with advisor	Fixed meetings with advisor to verify on state of project.	09.03.2023 23.03.2023 06.04.2023 20.04.2023 04.05.2023
Knowledge Exchange	Planned meeting in order to exchange knowledge about work done. These were held on the week-end.	after every product increment

7.1.3 Project Management Roles

Table 7.2: Project Management Roles

Role	Description	Name
Project Manager	Updates project plan and risks (every end of iteration), sends documents to Thomas 2 days in advance and controls time tracking & analysis.	Samuel Maissen
Scrum Master	Lead of sprint planning and retrospective meetings. Keeps track of deadlines and ensures communication within the team.	Samuel Maissen
Product Owner	Creates stories, manages backlog, prioritizes features.	Nick Wallner
Development Team	Completes the tasks in a Scrum sprint. Helps in sprint planning and goal setting.	Nick Wallner Samuel Maissen Mirio Eggmann Dario Glasl Jasmin Fitz
Responsible infrastructure	Technical responsibility and architect for infrastructure.	Mirio Eggmann
Responsible frontend	Technical responsibility and architect for frontend	Jasmin Fitz
Responsible backend	Technical responsibility and architect for backend	Dario Glasl
Responsible graphics	Responsible for all graphics used in the project	Nick Wallner
Responsible recordkeeping	Creates notes from meetings, documents the results in the project plan	Jasmin Fitz
Supporting architect	Coordinates the architecture of frontend, backend and infrastructure and maintains the big picture of it. After each iteration, the supporting architect also updates which NFRs are already fulfilled.	Mirio Eggmann

7.2 Long Term Planning

As described, we are using SCRUM for short-term planning in iterations. Because of the timing of the SE project and the RUP phases, we're having different iteration sizes, in contrast to the original SCRUM method.

	KW 7	KW 8	KW 9	KW 10	KW 11	KW 12	KW 13	KW 14
Start of the week	13/02/2023	20/02/2023	27/02/2023	06/03/2023	13/03/2023	20/03/2023	27/03/2023	03/04/2023
RUP	Inception	Elaboration						
Iterations	Iteration 0	Iteration 1			Iteration 2		Iteration 3	
Iteration start		23/02/2023			09/03/2023		23/03/2023	
Milestones	Project idea					First draft		
Reviews				09/03/2023		23/03/2023		06/04/2023
Increments	Program Increment 1						Program In	

Figure 7.1: Long Term Plan #1

KW 15	KW 16	KW 17	KW 18	KW 19	KW 20	KW 21	KW 22	KW 23
10/04/2023	17/04/2023	24/04/2023	01/05/2023	08/05/2023	15/05/2023	22/05/2023	29/05/2023	05/06/2023
Construction						Transition		
Iteration 4		Iteration 5		Iteration 6		Iteration 7		
06/04/2023		20/04/2023		04/05/2023		18/05/2023		
	Second draft				Final product			Project end
	20/04/2023		04/05/2023					
Program Increment 2		Program Increment 3				Program Increment 4		

Figure 7.2: Long Term Plan #2

7.2.1 Milestones

Project idea

A project idea is defined.

First draft

Most important requirements are identified. Risk analysis is done. Core architecture is implemented. Git repository, issue and time tracking and documentation is set up. Infrastructure is set up.

A first draft for the application is implemented. This includes:

- At least one request from the web frontend to backend
- At least one database call from the backend
- At least one response from the backend to the frontend

On the infrastructure side, this means the following:

- Frontend and backend services are runnable as Docker containers
- The database MongoDB is runnable as a Docker container
- An initial Kubernetes cluster deployment is created for staging (Development / Test), whereupon a new version is deployed on each pull request merge into the develop branch.

Second draft

Core requirements for the application are implemented. Core requirements include the following:

- Users can get activities proposed.
- Location based activities.
- A result page with for a proposed activity.
- Errors are handled via UI notifications.

Infrastructure is fail proof, meaning that the servers / services run on more than one node, so when one breaks, another can take over. Automatic production deployment is created when merged into the master branch. In our MongoDB database, sharding is activated.

Final product

The final product includes:

- Core requirements (filtering, result page, history, favorites and activity proposal)
- User account (login and current state of the users data like their own history and favorites)
- Most important findings from usability tests are implemented
- All features described in stories which are planned in sprints

Project end

Application works without GitLab pipeline errors. Application is deployed. Documentation is finished. Advisor has access to documentation and final application.

7.3 Short Term Planning

7.3.1 General Workflow

As described in section project management method, we are using SCRUM for short term planning. During the sprint planning meeting, we look at the backlog, create storys and brainstorm about new ideas in the team. For every story, we create a Jira ticket with a time estimate as described in section time tracking. Furthermore, we add labels and criterias to the story to make it easier to work with it later. Depending on the content, we also directly assign a responsible person. During the sprint, which lasts 2 weeks, we work on the storys previously discussed.

7.3.2 First Iteration

During the first three weeks, we didn't work exactly according to SCRUM, since we were only learning about it in the lecture. Nevertheless, we have scheduled these first three weeks as an iteration. We decided to already use Jira to track our tasks, but not in the form of user stories. The goal of the first iteration was to get everything up and running, as well as further define our project idea and decide on the tools and technologies. On Thursday, 09.03, our first official sprint started.

7.4 Risk Management

Table 7.3: Risk Matrix

Probability	Severity			
	Negligible	Marginal	Critical	Catastrophic
Certain				
Likely		R01		
Possible		R11, R06	R04, R07	
Unlikely		R05		R10
Rare	R08		R03, R09	R02
Eliminated				

Table 7.4: Risk Matrix Legend

Color	Meaning
	Eliminated
	Low
	Medium
	High
	Very high

Table 7.5: Risk List

Risik Nr.	Risik	Description	Mitigations
R01	Team member gets sick	In winter time it is likely that one of the team members will get sick. Since we are 5 people, this should not be a big problem. The other people will just have to invest a little more time than usual. It could be a bit of a problem if the person who got sick worked on a larger task completely alone.	We have to ensure that all persons of the team try to maintain a healthy lifestyle that prevents getting sick as much as possible. Furthermore, we have to ensure that we create enough small work tasks in the Sprint so that no single person works on a too large task which would block out other team members if the person working on this task gets sick. We also have to try to transfer knowledge across at least two team members (for example for react) so that we have no single point of failure.
R02	OST Gitlab is down	Our source code, artifacts, container-images and deployment are stored in a OST Gitlab repository. So it would be pretty bad if the service would not be available anymore.	We would still have a local copy of the Git-Repo on our computers on which we can work on our tasks. But if we would have to merge code or trigger our pipelines for the deployment on our production it would be pretty bad. Worst case we would be able to import the project on another Gitlab Server (for example gitlab.com) to still be able to work together if the OST Gitlab would have a major outage that takes days to fix (which is very unlikely).

R03	Cloud Jira is down	If Jira, our issue and time-tracker, is offline and so we would not be able to check our Sprint Board or Backlog.	We would still have our short and long-term plan where we can check what we had planned. Further with 5 persons we would be able to remember what we had as tasks in the current sprint. To track the time we could temporarily switch to a text file and book it later on the tasks when Jira is available again.
R04	Learning React takes more time than expected	Since only one person of the team already worked a little with react it could be possible that the other team members will have to invest more time than expected in learning and understanding how to use it.	Create prototypes to check if the learned knowledge can be applied.
R05	Conflict between team members	In a team with 5 members it is possible that conflicts will arise. But since we already know each other quite well it is unlikely except when we will get stressed at the end of the project.	We already know each other quite well, so this is quite useful to know how to prevent conflicts between each other. Furthermore we will use team building techniques (like eating lunch) to mitigate this problem.
R06	Misjudgment of the time required	Since we have never worked together in this team constellation and also with some of the technologies (React and MongoDB) it is quite hard to estimate the time required.	Create prototypes to have a feeling of the time required for the new technologies and refine the times for the tasks in the backlog refinements with the new knowledge of the past sprint. Since we work with scrum, we can always react flexibly to incorrect time estimates. For example, by reducing the scope.
R07	Learning LaTeX takes more time than expected	Since nobody of the team has ever really worked on a project with LaTeX to document the stuff it is possible that we will have to invest more time than expected.	Use the setup of the SEP1 lessons and maybe use something like Copilot to assist with LaTeX keywords in the IDE.
R08	There is yet another pandemic	We didn't expect Covid to happen so it is also possible that we will see yet another pandemic.	Since we already planned on working together partly remote we have everything setup to deal with a (worstcase) lockdown situation. However there will probably be a little work performance impact which we are not able to mitigate with preventive measures.

R09	Google Cloud Platform is down	If Google Cloud Platform would go down (which is pretty unlikely) we would not be able to deploy our project and we would also not be able to access an already deployed instance since we run our production on a Kubernetes cluster on google cloud.	We accept the risk that it is possible that GCP goes down because it is not highly critical that we can deploy and access our web application during the project phase. Only when we have the presentation it would be a bit cumbersome but also there we would be able to make a demo on a local machine.
R10	The chosen technologies are not working well together	Since nobody of the team has already worked with the exact tech-stack that we selected we are not 100% sure that they will work well together.	Create a small prototype very early which shows that the technologies are working together (Frontend (React), Backend (.NET), Database (MongoDB), Deployment (Kubernetes).
R11	Code does not meet quality requirements	It could be possible that the code does not meet the requirements we defined for the project.	We use a linter like eslint to be able to correct code smells and formatting errors directly in the IDE and also use a code checker (like SonarQube or again a linter like eslint) in the Pipeline. Furthermore we use PR reviews to ensure that another pair of eyes checked if the code meets the requirements.
R12	Knowledge transfer in between team members is insufficient	If team members do not effectively transfer it's knowledge about important parts they did in the application, it can lead to misunderstandings and conflicts which has in the end impact on the quality and time invested into the application.	Plan regularly meeting in order to exchange in between team members and take up the key points about main parts done in the application.

7.4.1 Updated Risk Matrix - Sprint 2

Table 7.6: Risk Matrix Sprint 2

Probability	Severity			
	Negligible	Marginal	Critical	Catastrophic
Certain				
Likely			R12	
Possible		R11, R06	R04	
Unlikely		R05, R01		
Rare	R08		R03, R09, R07	R02, R10
Eliminated				

R12 (NEW)

Table 7.7: Risk Sprint 2 - R12 (NEW)

Brief description	Knowledge transfer in between team members is insufficient
Old position	-
New position	Likely-Critical
Explanation	While working on the project each team member was focused on his assigned part for the project. As recognized there is not enough knowledge exchange happening what could lead to problems if not clear how something was supposed to be.

R10

Table 7.8: Risk Sprint 2 - R10

Brief description	The chosen technologies are not working well together
Old position	Unlikely-Catastrophic
New position	Rare-Catastrophic
Explanation	Since we were not 100 percent sure if all the technologies work well together, we planned the following functionality with the first draft of the application: request from frontend to backend, a database call from backend and a response from backend to frontend. Fortunately, everything worked as expected and there were no bigger issues. As well with the first draft, we defined to have the backend, frontend and database running as Docker containers. Furthermore, an initial Kubernetes deployment was planned and created. We are now able to run all the components in the Kubernetes deployment. Therefore, the risk of something not working with the technologies could successfully be mitigated.

R07

Table 7.9: Risk Sprint 2 - R07

Brief description	Learning LaTeX takes more time than expected
Old position	Possible-Critical
New position	Rare-Critical
Explanation	Time has shown that all team-members get along great with LaTeX and that the documentation is in great quality. Furthermore, the time invested to document is within the normal range. Nonetheless, some things possibly could require more time than expected, that's why we left the probability to "Rare" and didn't set it to "Eliminated".

R01

Table 7.10: Risk Sprint 2 - R01

Brief description	Team member gets sick
Old position	Likely-Marginal
New position	Unlikely-Marginal
Explanation	As 4 of 5 team-members got sick last week and already recovered from it, it is quite uncertain that they will get sick again. They should now be more resistant to the annual flu.

7.4.2 Updated Risk Matrix - Sprint 3

Table 7.11: Risk Matrix Sprint 3

Probability	Severity			
	Negligible	Marginal	Critical	Catastrophic
Certain				
Likely				
Possible		R11, R06	R04	
Unlikely		R05, R01		
Rare	R08		R03, R09, R07	R02, R10
Eliminated			R12	

R12

Table 7.12: Risk Sprint 3 - R12

Brief description	Knowledge transfer in between team members is insufficient
Old position	Likely-Critical
New position	Likely-Eliminated
Explanation	An initial 2 hour meeting took place in order to exchange knowledge about the current version of our application. Further after every product increment there is a knowledge-exchange planned.

7.4.3 Updated Risk Matrix - Sprint 4

Table 7.13: Risk Matrix Sprint 4

Probability	Severity			
	Negligible	Marginal	Critical	Catastrophic
Certain				
Likely				
Possible		R06	R04	
Unlikely		R05, R01		
Rare	R08	R11	R03, R09, R07	R02, R10
Eliminated			R12	

R11

Table 7.14: Risk Sprint 4 - R11

Brief description	Code does not meet quality requirements
Old position	Possible-Marginal
New position	Rare-Marginal
Explanation	Linters are implemented and metrics meet the quality requirements. Further, the course of the work has shown that the PR Reviews were taken seriously.

7.4.4 Updated Risk Matrix - Sprint 5

Table 7.15: Risk Matrix Sprint 5

Probability	Severity			
	Negligible	Marginal	Critical	Catastrophic
Certain				
Likely				
Possible		R06		
Unlikely		R05, R01	R04	
Rare	R08	R11	R03, R09, R07	R02, R10
Eliminated			R12	

R04

Table 7.16: Risk Sprint 5 - R04

Brief description	Learning React takes more time than expected
Old position	Possible-Critical
New position	Unlikely-Critical
Explanation	All team members did now work with React and got to know it better. The invested time is in an acceptable range.

7.4.5 Updated Risk Matrix - Sprint 6

Table 7.17: Risk Matrix Sprint 6

Probability	Severity			
	Negligible	Marginal	Critical	Catastrophic
Certain				
Likely				
Possible				
Unlikely		R05, R01, R06	R04	
Rare	R08	R11	R03, R09, R07	R02, R10
Eliminated			R12	

R06

Table 7.18: Risk Sprint 6 - R06

Brief description	Misjudgment of the time required
Old position	Possible-Marginal
New position	Unlikely-Marginal
Explanation	While already working on the project quite some time all team members do now have a good feeling with the new technologies and how much time is required to implement certain features. Furthermore all team members strengthen the cooperation and team spirit through the whole project and therefore work could be done more efficient.

7.4.6 Updated Risk Matrix - Sprint 7

Table 7.19: Risk Matrix Sprint 7

Probability	Severity			
	Negligible	Marginal	Critical	Catastrophic
Certain				
Likely				
Possible				
Unlikely		R05, R01, R06		
Rare	R08	R11	R03, R09, R07	R02, R10
Eliminated			R12, R04	

R04

Table 7.20: Risk Sprint 6 - R04

Brief description	Learning React takes more time than expected
Old position	Unlikely-Critical
New position	Eliminated
Explanation	Learning React even better does no longer represent a risk, since we got to know it quite good for the range we needed to. Furthermore, almost all features are implemented and no big feature which would require some distinct React skills is planned.

Chapter 8

Time Tracking Report

8.1 How we Track Time

Following the regulations we defined for the time tracking:

- All time is tracked in issues created in our [Jira Project](#).
- We track our time in 30 min blocks.
- Per sprint as a team we have around **80h** of capacity.
- For every sprint there is a separate coordination ticket to track the time for meetings and other unplanned coordination tasks.
- During sprint planning, we are estimating together the time used for the specific stories.
- The estimations are done in hours and is in form of the Fibonacci sequence with a maximum of 13h per story.
- Does a story exceed 13h it needs to be broken down in more parts.

8.2 Time Statistics

In order to have a nice overview about the current status and time tracking of the project we created a [dashboard](#) in Jira. It covers the most important key numbers and answers following questions. Please be aware that the screenshots made of the dashboard are not up to date and only for documentation purpose. The latest status can always be found online on the [dashboard](#).

8.2.1 Overview Current Sprint

In order to analyze the status of the current sprint, there are multiple charts. The chart "Current sprint key values" shows how much of the sprint backlog is already done, in progress and still to do, which provides a simple and clear overview:

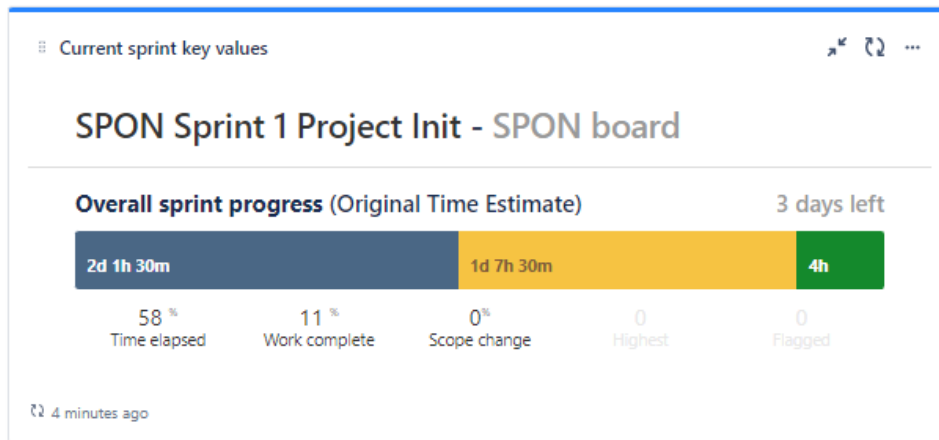


Figure 8.1: Current Sprint Overview Time Tracking

Further there is a chart "Sprint time management for issues" which shows the issues in the current sprint with it's statistics:

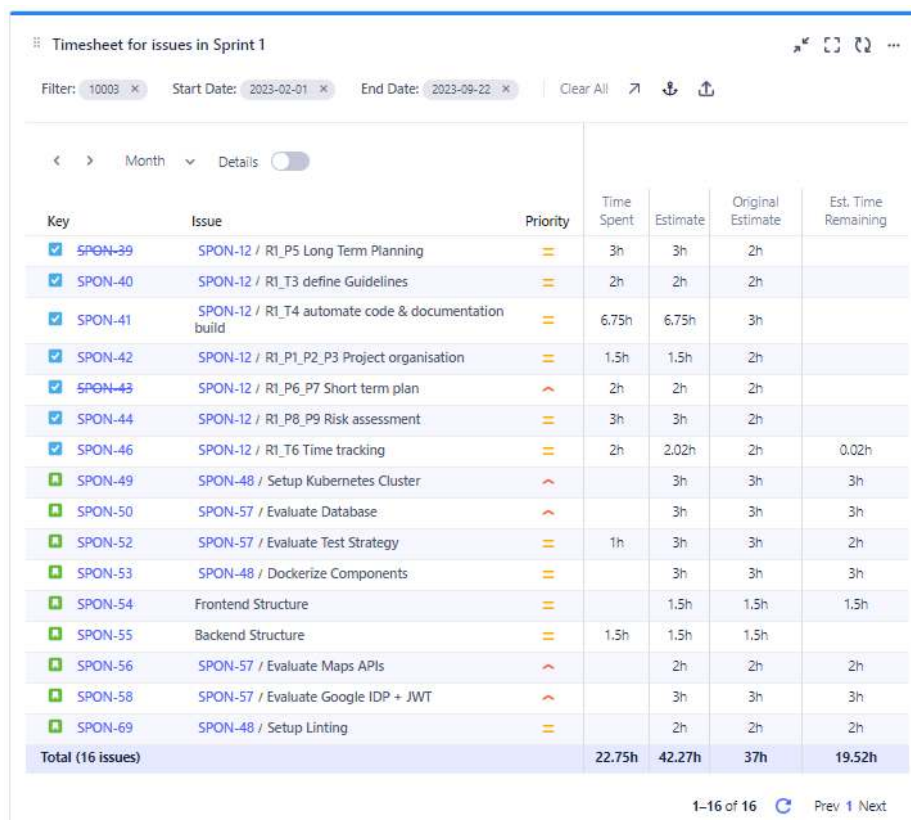


Figure 8.2: Sprint Time Management for Issues Time Tracking

As well there is a chart "Sprint time management for team" which shows current sprint's workload per team-member:

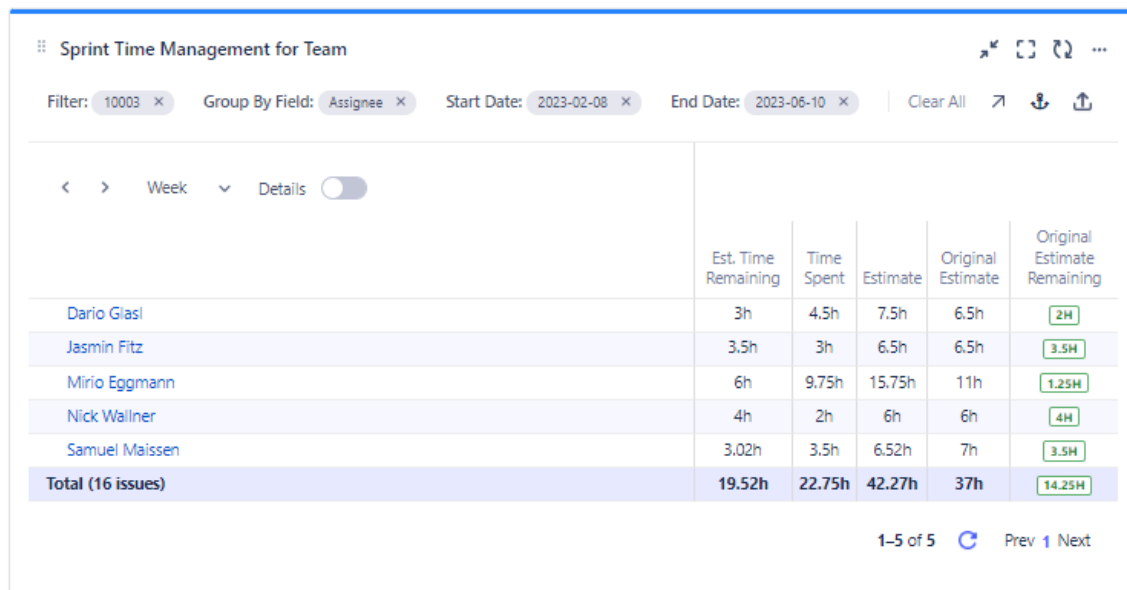


Figure 8.3: Sprint Time Management for Team Time Tracking

8.2.2 Planned and Invested Time per Sprint

This can be found on in the chart "time planned and spend for the sprints". The chart lists each sprint with its time spent and estimated:

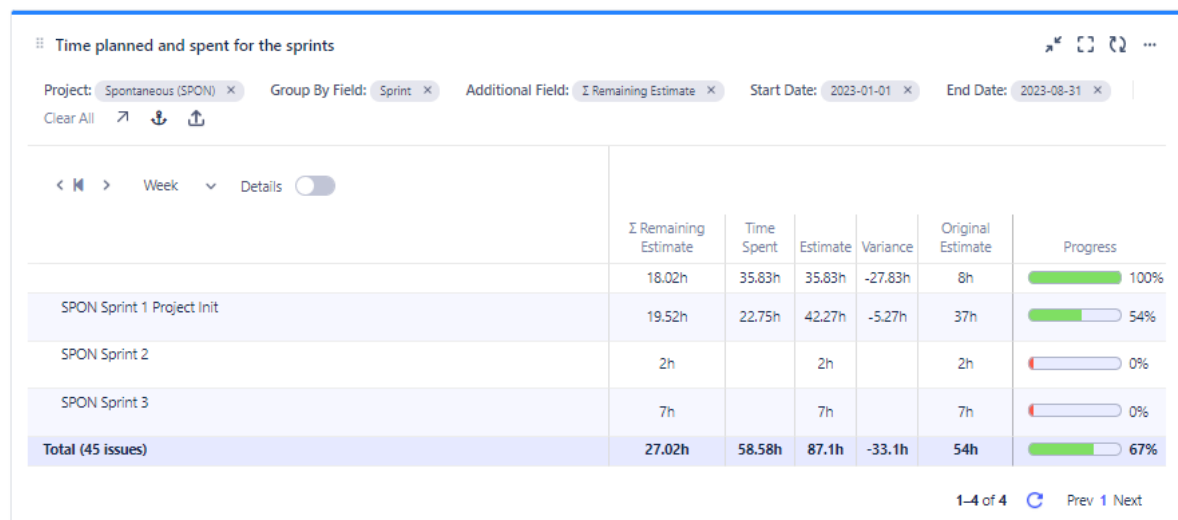


Figure 8.4: Time Planned and Spend for the Sprints Time Tracking

8.2.3 Invested Time per Topic

This can be found on in the chart "Overall time balance per issue". It shows all issues ever created in the project with their estimation and time spent. Additionally, the total time spent can be seen in the last row on the very right.

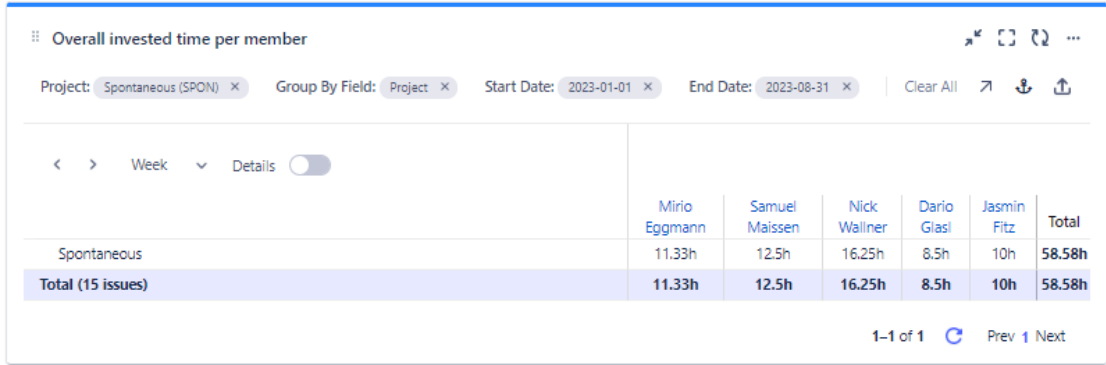


Key	Issue	Priority	Time Spent	Estimate	Variance	Original Estimate	Progress
SPON-9	Coordination	=	0.83h	0.83h	-0.83h		100%
SPON-10	SPON-9 / Meeting-23022023-KickOff	=	16.5h	16.5h	-16.5h		100%
SPON-11	Projekt Planning	=	3.5h	3.5h	-3.5h		100%
SPON-12	R1-InitialProjectSetup	=					
SPON-33	R2-Requirements	=					
SPON-34	R3-EndofElaboration	=					
SPON-35	R4-Quality	=					
SPON-36	R5-Architecture	=					
SPON-37	R6-ProjectPresentation	=					
SPON-38	SPON-9 / Meeting-25022023-ProjectCoordination	=	2.5h	2.5h	-2.5h		100%
SPON-39	SPON-12 / R1_P5 Long Term Planning	=	3h	3h	-1h	2h	100%
SPON-40	SPON-12 / R1_T3 define Guidelines	=	2h	2h		2h	100%

Figure 8.5: Overall Time Balance per Issue Time Tracking

8.2.4 Overall Invested Time per Member

This can be found on in the chart "Overall invested time per member". The total time per member can be analyzed, as well as the total time invested in the project on the very right:

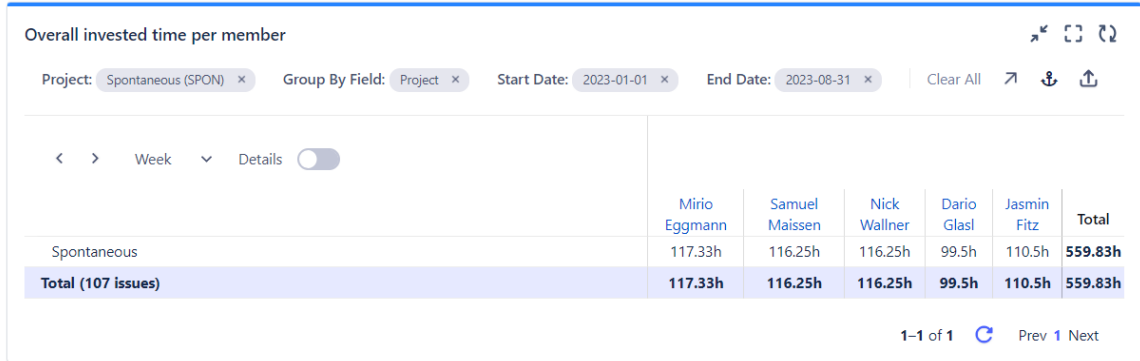


Project	Group By Field	Start Date	End Date						
Spontaneous (SPON)	Project	2023-01-01	2023-08-31						
				Mirio Eggmann	Samuel Maissen	Nick Wallner	Dario Glasl	Jasmin Fitz	Total
Spontaneous				11.33h	12.5h	16.25h	8.5h	10h	58.58h
Total (15 issues)				11.33h	12.5h	16.25h	8.5h	10h	58.58h

Figure 8.6: Overall Invested Time per Member Time Tracking

8.3 Final Overall Statistic

While the previous section described the single charts, this section represents the final time tracking report for the time invested during the whole project. In total we invested nearly 560 hours into our SE-Project. The work was distributed quite evenly through all project members.



Project: Spontaneous (SPON) ×	Group By Field: Project ×	Start Date: 2023-01-01 ×	End Date: 2023-08-31 ×	Clear All	↗	📌	🔄
< > Week ▾	Details ☐						
	Mirio Eggmann	Samuel Maissen	Nick Wallner	Dario Glasl	Jasmin Fitz	Total	
Spontaneous	117.33h	116.25h	116.25h	99.5h	110.5h	559.83h	
Total (107 issues)	117.33h	116.25h	116.25h	99.5h	110.5h	559.83h	
							1-1 of 1 🔄 Prev 1 Next

Figure 8.7: Overall Time Tracking

Chapter 9

Development Environment

This chapter describes how to set up the different tools to start developing on the project.

9.1 Git Project

Clone the [git repository](#) with a tool of your choice, e.g. using the [git bash](#) and executing the `git clone` command as described [here](#).

9.2 Frontend

9.2.1 Installation

Follow these steps to set up the environment for frontend development:

1. Install a suitable IDE that supports React and TypeScript. The configuration steps in the next section are explained for [Webstorm](#) by JetBrains.
2. Install npm according to these [instructions](#).
3. Install dependencies through the command `npm install` executed in a terminal in the `/Frontend` folder.

9.2.2 Configuration

To unify development in the team, following configurations should be set up:

1. Install the plugin [EditorConfig](#) in the Webstorm IDE.
2. Be sure the plugin is enabled. You can change this under Settings > Editor > Code Style > General > Enable EditorConfig support.
3. Best practice is to enable automatic reformatting on save. You can set this up under Settings > Tools > Actions on Save > Reformat Code.

9.2.3 Development

To start developing, follow these steps:

1. With the command `npm start` executed in a terminal in the `/Frontend` folder, a local server is started up.
2. Open a browser and go to URL `localhost:8080` to view the running application (you may need to adjust the port).

9.2.4 Testing

To perform static code analysis / linting, execute the command **npm run check:eslint** in a terminal in the /Frontend folder.

9.3 Backend

9.3.1 Installation

Follow these steps to set up the environment for backend development:

1. Install .NET 7 according to these [instructions](#)
2. Install a suitable IDE that supports .NET. The configuration steps in the next section are explained for [Visual Studio](#)

9.3.2 Configuration

To unify development in the team, following configurations should be set up:

1. Install the extension [Resharper](#) in Visual Studio.
2. Setup Resharper to use your preferred shortcuts.
3. Add the Google API key to your user secrets in the **Spontaneous.Web** project. Since this information is not publicly disclosed and not in version control, please ask a team member for more specific information.

9.3.3 Development

To start developing, follow these steps:

1. Start the application using the command **dotnet run** executed in a terminal in the /Backend/Spontaneous.Web folder.
2. Open a browser and go to URL **localhost:5153/swagger/index.html** to view the swagger endpoint (you may need to adjust the port).

9.3.4 Testing

- A specific tool for linting is not necessary since the tool Resharper automatically fixes any issues in this direction.
- To execute the specified unit tests, either use the command **dotnet test** executed in a terminal in the /Backend folder or use the test explorer in Visual Studio.

9.4 Database

9.4.1 Installation

Follow these steps to set up the environment for database access and development:

1. Install [Docker Desktop](#).
2. Install a suitable IDE that supports MongoDB. The configuration steps in the next section are explained for [DataGrip](#) by JetBrains.

9.4.2 Startup

To be able to start up the database, follow these steps:

1. Start the Docker Desktop application
2. Start the docker container using the command **docker-compose up** executed in a terminal in the /Database folder.
3. Currently, you have to manually create the database after starting the container. Please refer to the next section "Access" to learn about how to accomplish this (steps 1-5 using the CLI). In the future, this might be done by a script.

9.4.3 Access

There are different ways to access the database once it is started up.

- Add a connection to the database in DataGrip following these [instructions](#).
- Or connect to the database using a CLI.
 1. Execute the command **docker ps** in a new terminal to view a list of all running docker containers.
 2. Execute the command **docker exec -it CONTAINER_HASH bash** in a terminal where CONTAINER_HASH is replaced with the correct container id visible in the output from the command before.
 3. Execute the command **mongosh --username admin** in the terminal. Please ask a team member for the secret password to gain access.
 4. You can then access the database using [MongoDB Shell](#).
 5. If this is the first time using the database, execute the following commands (case sensitive!) to create the database and a first collection:
use Spontaneous
db.createCollection('History')

9.5 Infrastructure

To be able to work with the Kubernetes cluster, you need access to the [Google Cloud project](#). Please ask a team member for assistance.

9.6 Automatic Testing

Apart from the possibility to execute local checks or run unit tests manually, the code is checked and tested in the pipeline, executed on every push to the git repository. The following tests are executed:

- Unit tests (Backend)
- GitLab Code Quality (Backend)
- Cypress tests (End to end)

Chapter 10

Personal Reports

10.1 Retrospective Meetings

We write our retro points to our [Confluence](#) as meeting notes and also use retro boards in [Miro](#).

10.2 Project Retro

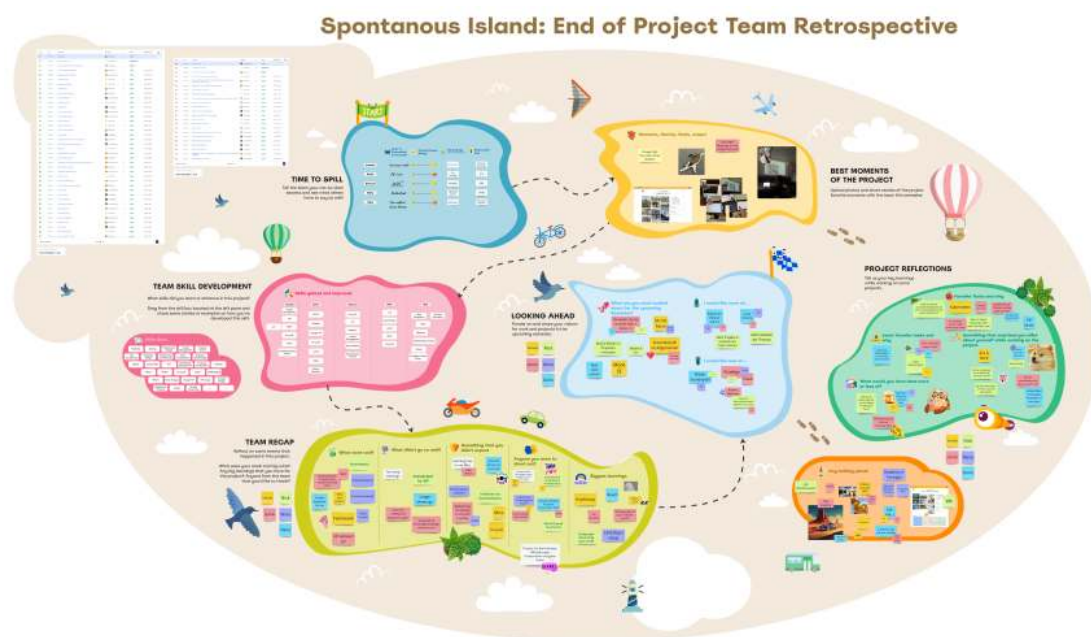


Figure 10.1: End of Project Team Retrospective

10.2.1 Detail Screenshots

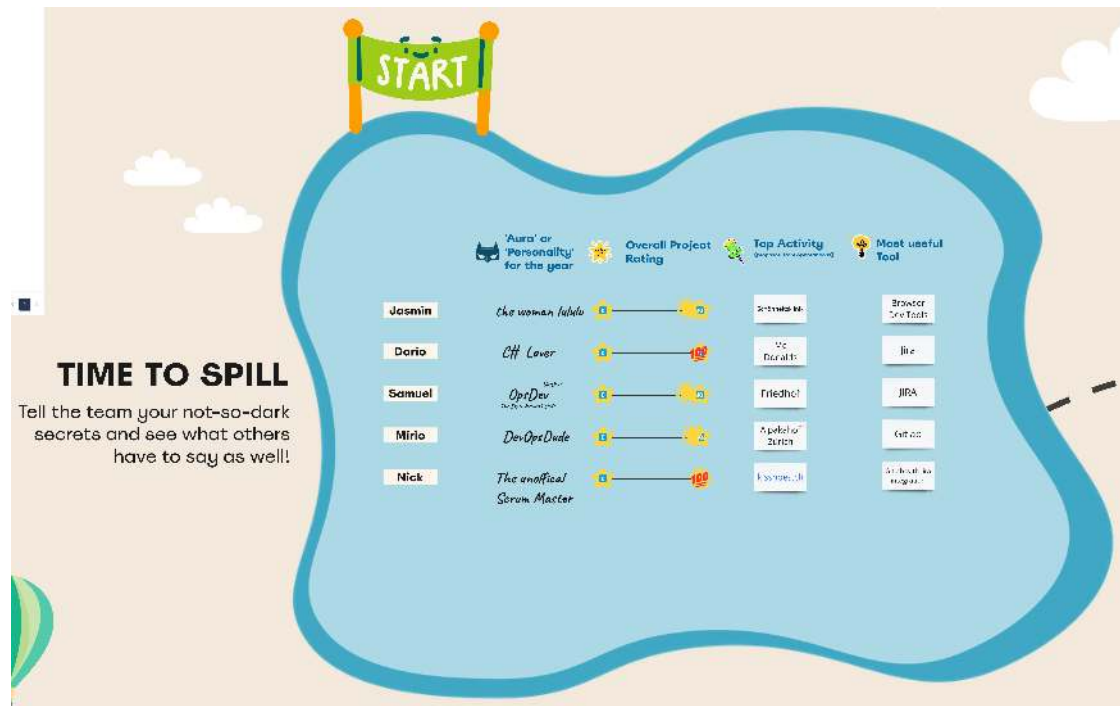


Figure 10.2: Project Retro Board - Start

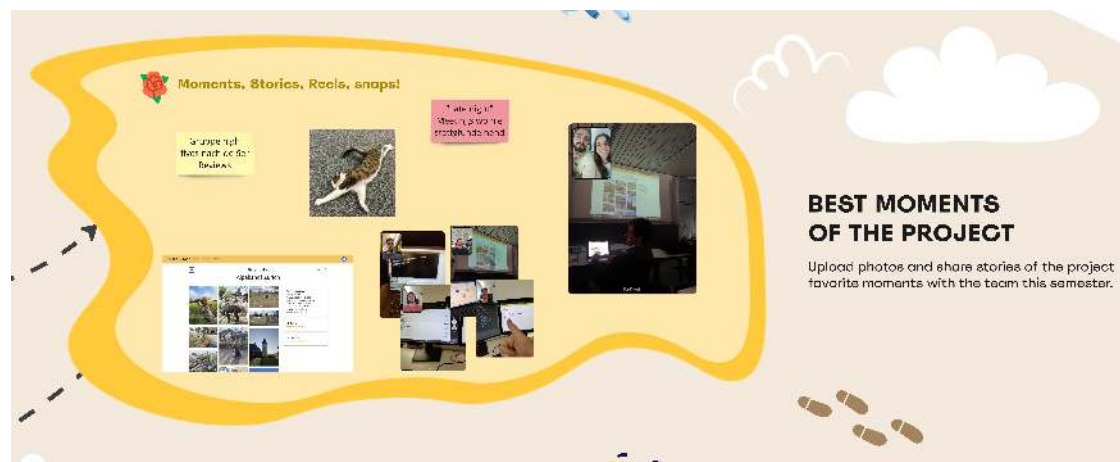


Figure 10.3: Project Retro Board - Best Moments

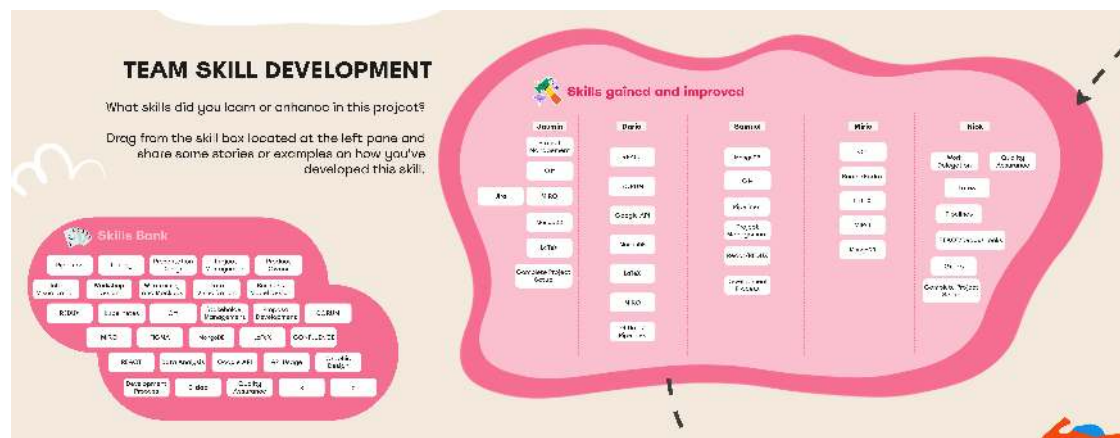


Figure 10.4: Project Retro Board - Skills Gained

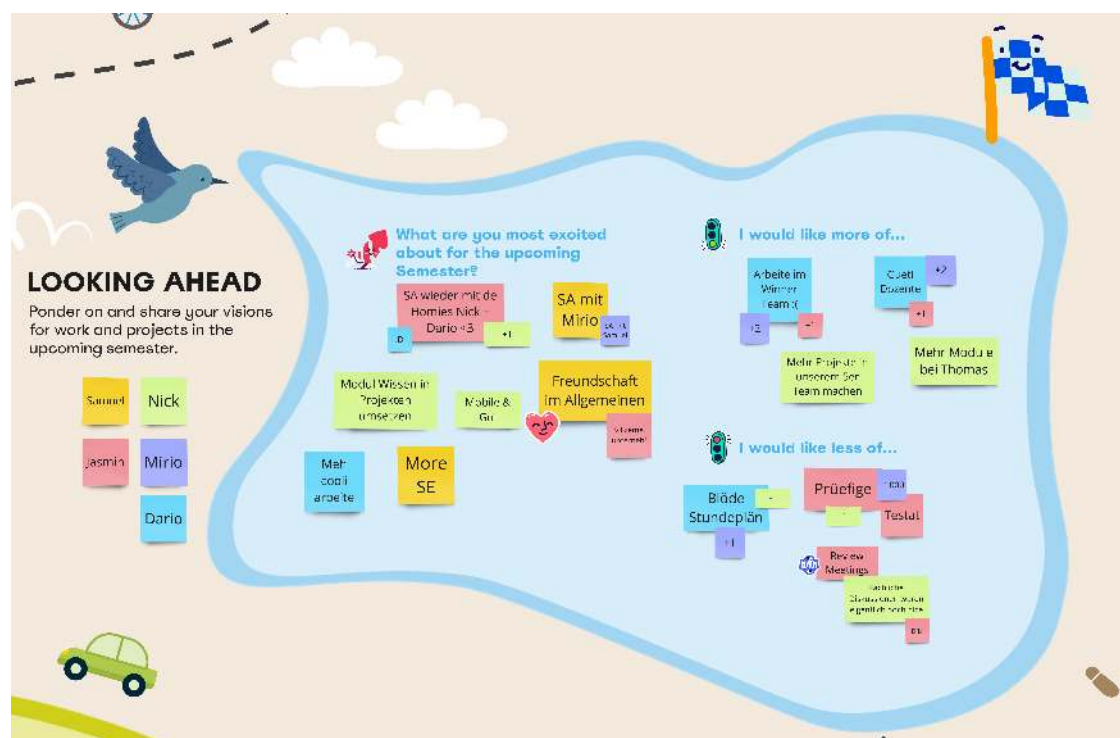


Figure 10.5: Project Retro - Looking Ahead



Figure 10.6: Project Retro Board - Vacation



Figure 10.7: Project Retro Board - Project Reflection

Chapter 11

Meeting Minutes

We write our meeting minutes to our [Confluence](#). Use the attachments PDF to see meeting minutes details.

List of Tables

1.1	Non-Functional Requirements: NFR01	6
1.2	Non-Functional Requirements: NFR02	6
1.3	Non-Functional Requirements: NFR03	6
1.4	Non-Functional Requirements: NFR04	7
1.5	Non-Functional Requirements: NFR05	7
1.6	Non-Functional Requirements: NFR06	7
1.7	Non-Functional Requirements: NFR07	7
1.8	Non-Functional Requirements: NFR08	8
1.9	Non-Functional Requirements: NFR09	8
1.10	Non-Functional Requirements: NFR10	8
1.11	Non-Functional Requirements: NFR11	8
1.12	Non-Functional Requirements: NFR12	9
1.13	Non-Functional Requirements: NFR13	9
1.14	Non-Functional Requirements: NFR14	9
1.15	Non-Functional Requirements: NFR15	9
1.16	Tracking of the NFRS - Sprint 2	10
1.17	Tracking of the NFRS - Sprint 3	11
1.18	Tracking of the NFRS - Sprint 4	12
1.19	Tracking of the NFRS - Sprint 5	13
1.20	Tracking of the NFRS - Sprint 6	14
1.21	Tracking of the NFRS - Sprint 7	15
5.1	Test Protocol Usability Test #1	40
5.2	Test Protocol Usability Test #2	41
7.1	Project Meetings	45
7.2	Project Management Roles	46
7.3	Risk Matrix	49
7.4	Risk Matrix Legend	49
7.5	Risk List	49
7.6	Risk Matrix Sprint 2	52
7.7	Risk Sprint 2 - R12 (NEW)	52
7.8	Risk Sprint 2 - R10	52
7.9	Risk Sprint 2 - R07	53
7.10	Risk Sprint 2 - R01	53
7.11	Risk Matrix Sprint 3	53
7.12	Risk Sprint 3 - R12	53
7.13	Risk Matrix Sprint 4	54
7.14	Risk Sprint 4 - R11	54
7.15	Risk Matrix Sprint 5	54
7.16	Risk Sprint 5 - R04	54
7.17	Risk Matrix Sprint 6	55
7.18	Risk Sprint 6 - R06	55
7.19	Risk Matrix Sprint 7	55

7.20 Risk Sprint 6 - R04 55

List of Figures

2.1	Domain model	16
3.1	Home Screen Mockup	18
3.2	Activity Screen Mockup	19
3.3	Favorites Screen Mockup	20
3.4	History Screen Mockup	21
3.5	Account Screen Mockup	21
3.6	Color Scheme	22
3.7	Logo	22
4.1	C4 Context Diagram	24
4.2	C4 Container Diagram	25
4.3	Activity Proposal Sequence Diagram	26
4.4	Authentication Sequence Diagram	27
4.5	The Clean Architecture Approach	28
4.6	Backend Project Structure	29
4.7	Frontend Architecture	30
4.8	Pipeline and Deployment Architecture	30
4.9	Sharded MongoDB Setup	31
4.10	Infrastructure View	32
4.11	Tech Stack	36
5.1	Git Branching Workflow	38
7.1	Long Term Plan #1	46
7.2	Long Term Plan #2	47
8.1	Current Sprint Overview Time Tracking	57
8.2	Sprint Time Management for Issues Time Tracking	57
8.3	Sprint Time Management for Team Time Tracking	58
8.4	Time Planned and Spend for the Sprints Time Tracking	58
8.5	Overall Time Balance per Issue Time Tracking	59
8.6	Overall Invested Time per Member Time Tracking	59
8.7	Overall Time Tracking	60
10.1	End of Project Team Retrospective	64
10.2	Project Retro Board - Start	65
10.3	Project Retro Board - Best Moments	65
10.4	Project Retro Board - Skills Gained	66
10.5	Project Retro - Looking Ahead	66
10.6	Project Retro Board - Vacation	67
10.7	Project Retro Board - Project Reflection	67