Bachelorarbeit am Institut für Informatik der Freien Universität Berlin

Human-Centered Computing (HCC)

How does an editor for dynamic resources for users with different levels of expertise look like and how can it be conceptualized and implemented within the constraints of an existing ecosystem?

Matthias Kind

Matrikelnummer: 5338650

matthias.kind@fu-berlin.de

Betreuer: Florian Berger

Erstgutachterin: Prof. Dr. Claudia Müller-Birn

Zweitgutachter: Prof. Dr. Lutz Prechelt

Berlin, 31.1.2023

Eidesstattliche Erklärung

Ich versichere hiermit an Eides Statt, dass diese Arbeit von niemand anderem als meiner Person verfasst worden ist. Alle verwendeten Hilfsmittel wie Berichte, Bücher, Internetseiten oder ähnliches sind im Literaturverzeichnis angegeben, Zitate aus fremden Arbeiten sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

Berlin, den 31.1.2023

Matthias Kind

Abstract

In recent years, the shift from print to digital publishing channels has increased the need for tools that allow publishers to quickly build and configure digital platforms.

This bachelor thesis addresses this problem by conceptualizing, planning and implementing a User Interface (UI) Editor for apps and websites used by magazine and news publishers as a case study. The UI Editor is built for a proprietary web framework called "Purple Experience" and was developed within the constraints of an existing software ecosystem, which posed challenges and limitations on the design and implementation of the tool.
To gain insights into the needs and workflows of the groups of users who will be using the editor, a variety of Human Computer Interaction (HCI) methods get applied during the user research phase, including moderated observations and interviews.

The outcome of this research is useful as guidance for future software development projects for internal tools at companies. It can also be useful in environments where constraints exist, but a user base is already in place to provide additional valuable input and feedback. As a result of the user research phase, an interactive prototype was built using modern web technologies. In the next step it was deployed to a controlled group of test users. This approach combined with methods of agile software development allowed iterating fast and collect direct feedback until it was rolled out for production use.

The importance of considering HCI principles for brownfield software development projects is demonstrated and I give examples on how to integrate these principles into a real-world context.

Zusammenfassung

In den letzten Jahren hat die Verlagerung von gedruckten zu digitalen Publikationskanälen dazu geführt, dass Verlage Werkzeuge brauchen, mit denen schnell und einfach digitale Plattformen gebaut und konfiguriert werden können.

Die vorliegende Bachelorarbeit befasst sich mit diesem Problem, indem als Fallstudie ein User Interface (UI) Editor für Apps und Webseiten von Magazin- und Nachrichtenverlagen konzeptioniert, entworfen und implementiert wird. Der UI Editor wird für ein proprietäres Web Framework namens Purple Experience gebaut, wobei die Limitierungen des existierenden Software-Ökosystems berücksichtigt werden müssen. Dies brachte Herausforderungen und Einschränkungen für das Design und die Implementierung des Tools mit sich.
Um Einblicke in die Bedürfnisse und Arbeitsweisen der Nutzergruppen zu gewinnen, werden in der Benutzerforschungsphase verschiedene Methoden der Human Computer Interaction (HCI) Methoden angewandt, darunter moderierte Beobachtungen und Interviews.

Die Forschungsergebnisse können als Leitfaden für zukünftige Software-Entwicklungsprojekte für interne Tools in Unternehmen dienen. Zudem können sie nützlich sein, wenn in der Softwareumgebung zwar Einschränkungen gelten, jedoch eine bereits vorhandene Nutzerbasis wertvollen Input und Feedback liefern kann. Ergebnis dieser Forschungsphase ist ein interaktiver Prototyp unter Verwendung moderner Web-Technologien. Darauf folgend wird der UI Editor einer Gruppe von Testnutzern zur Verfügung gestellt. Dieser Ansatz, kombiniert mit agiler Softwareentwicklung, ermöglichte schnelle Iterationen und das Sammeln von direktem Feedback vor dem öffentlichen Release.

Es wird aufgezeigt, wie wichtig die Berücksichtigung von HCI-Prinzipien für Brownfield-Software-Entwicklungsprojekte ist, und ich gebe Beispiele, wie diese Prinzipien in einen realen Kontext integriert werden können.

1. Introduction

1.1. Topic and context

In the ever-growing world of software development, many companies are now in the situation to maintain a large software ecosystem with complex dependencies. Still, there is need for continuous improvement and development to stay competitive. This poses the challenge of improving the software from aspects like user experience, scalability and maintainability while being restricted by the ecosystem.
From my point of view, Greenfield development seems to be implicitly assumed in many books and articles about HCI. This assumption is not applicable to the situation many software companies are in today.
In a Brownfield development project, HCI methods need to be adapted to account for technical constraints while still addressing user needs. User research in general is often neglected due to tight deadlines and limited resources which usually leads to premature releases and unstable software. This thesis therefore aims to demonstrate the advantages of structured user research in theory as well as in practice, using the case study "UI Editor".

1.2. Goals of this thesis

The goal is to demonstrate how HCI principles and methods can be applied in a brownfield project, using a real-world case study at the company Sprylab as an example. Sprylab is a company which is engaged in the digital publishing industry, providing Software-as-a-Service (SaaS) to publishing houses for editing and distributing content. The case study consists of the redevelopment of a UI Editor to improve user experience and usability for web developers, admins and editors.

1.3. Process for research, prototyping and implementation

The software design process used to develop the UI Editor is described in [2, p. 104]. There, the process is divided into the three phases "Design Thinking", "Lean UX" and "Agile" (for more details see Glossary).

For this concrete case study, the process looks like this:

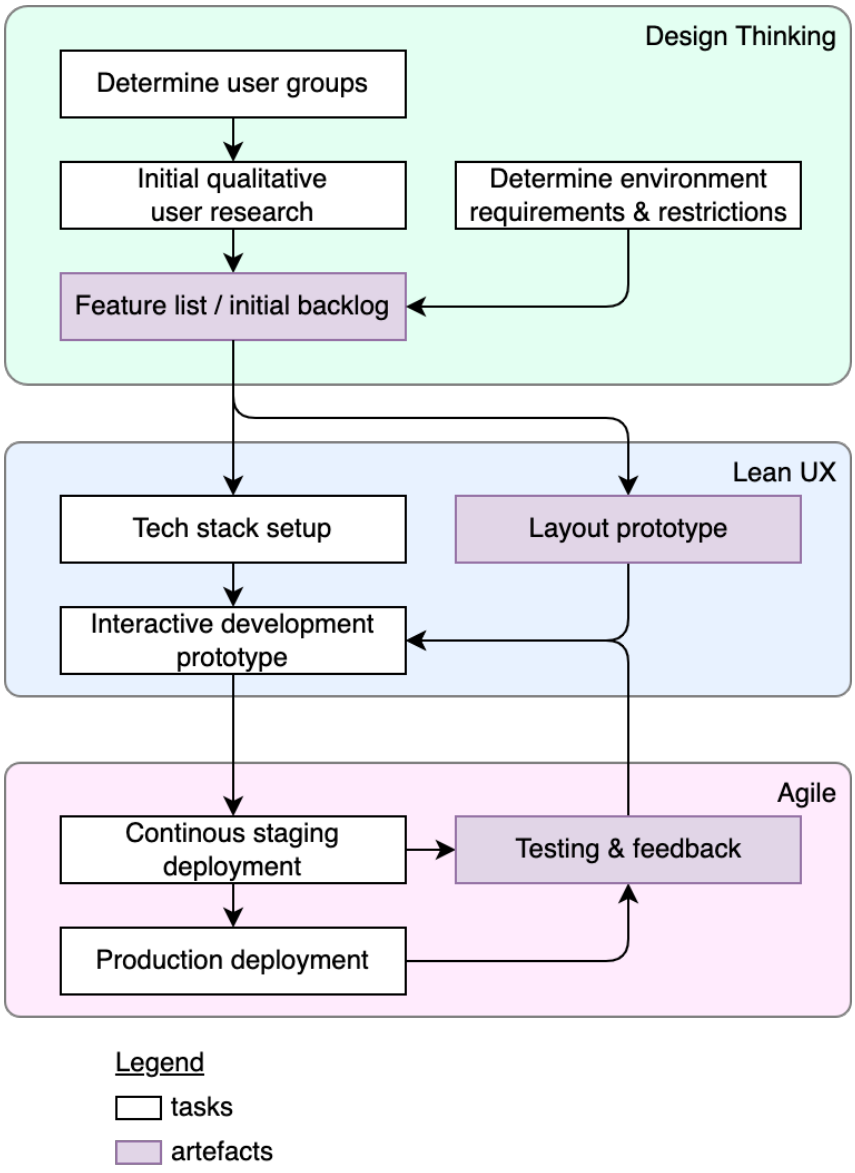

Figure 1.1.: Software design process for the UI Editor.

2. Theoretical background

Before discussing the design and implementation of the UI Editor, it is necessary to give a brief overview of the applied parts of HCI and the specific context, challenges and opportunities in which the UI Editor was developed.

2.1. Applied human-computer interaction methods

This section concretizes the HCI aspects of this thesis and describes how and why the subject is approached from a not so common point of view. HCI is a complex topic with a long list of available methods and even more ways to adapt them to a concrete project, so it is impossible to cover all aspects. The UI Editor development faced challenges in integrating with an existing ecosystem and meeting technical requirements in a Brownfield development project. In my opinion, technical requirements face only limited coverage in HCI literature. Therefore, the focus was set on them during this case study.
To determine the initial functional requirements, qualitative user research methods like moderated observations (3.2.1) and interviews (3.2.2) were chosen to be applied in an initial Design Thinking phase. While quantitative research methods were also used to get insights during the testing phase, the decision was made against using them for this phase. The available user base was small enough to get meaningful and representative statements only from qualitative research. A questionnaire would either be too long or not provide enough helpful insights.
Furthermore, the two concepts ”SMART criteria” and ”three major factors of HCI” were used as foundation and guidelines during the process, which is described in chapter 3 in more detail.

2.2. Project specific background

In the following the functional and technical backgrounds are described to gain an understanding of the application domain.

2.2.1. Functional background

The publishing houses respectively their digital departments (in the following *customer*) purchase the license for an app or website (in the following just *app*, as there is not much difference besides the end medium) from Sprylab. Then, they can import content via multiple ways into the system, or the editors write the content directly inside the tools provided as SaaS.

The UI Editor fits into the use case "Per App configuration & style" of fig. 2.1, with which mostly Frontend Developers and Project Managers from Sprylab as well as some external customer's IT Admins will interact. The goal is to lower the editing burden as much as possible, so that more of the configuration can be handed off to external customers while also improving usability for the developers of the company.

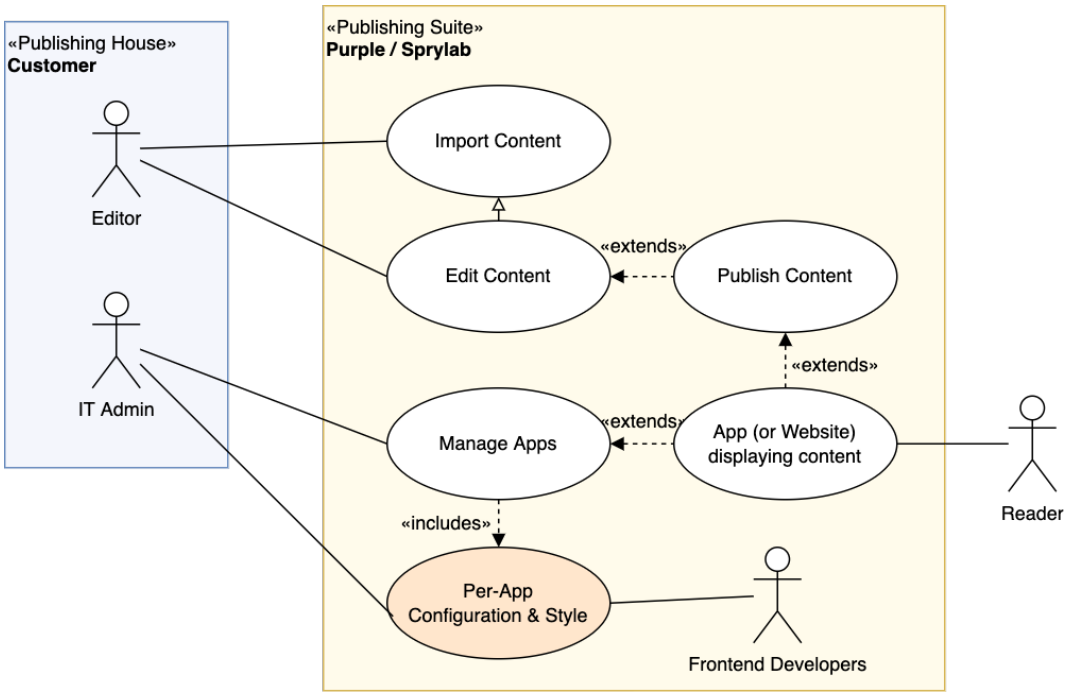

Figure 2.1.: Use case diagram: interactions from publishers, readers and frontend developers with the system

2.2.2. Technical background

The UI Editor will be implemented within the context of the proprietary web framework named Purple Experience which is built on top of Angular. This framework allows complete configurability through JSON files, including routing, rendering of different components, connecting data sources, loading assets and styling the page with CSS. These configs and assets are stored in a per-app file system called dynamic resources.

Dynamic resources are individually managed and loaded for every app. This way, on mobile phones the end users download a native core app, which in turn just downloads the dynamic resources and executes the angular app with the configs provided from the resources. Similar, when an end user requests a website, the backend server just looks up the dynamic resources matching this app's domain and renders the website using that config. As a result, all customers can share the same server instance(s) in case of websites, or at least

do not require extra native sourcecode changes per app.
In addition, there are preview and live resources for every app, so that changes can be tested before they are released to the end users. (see fig. 2.2)

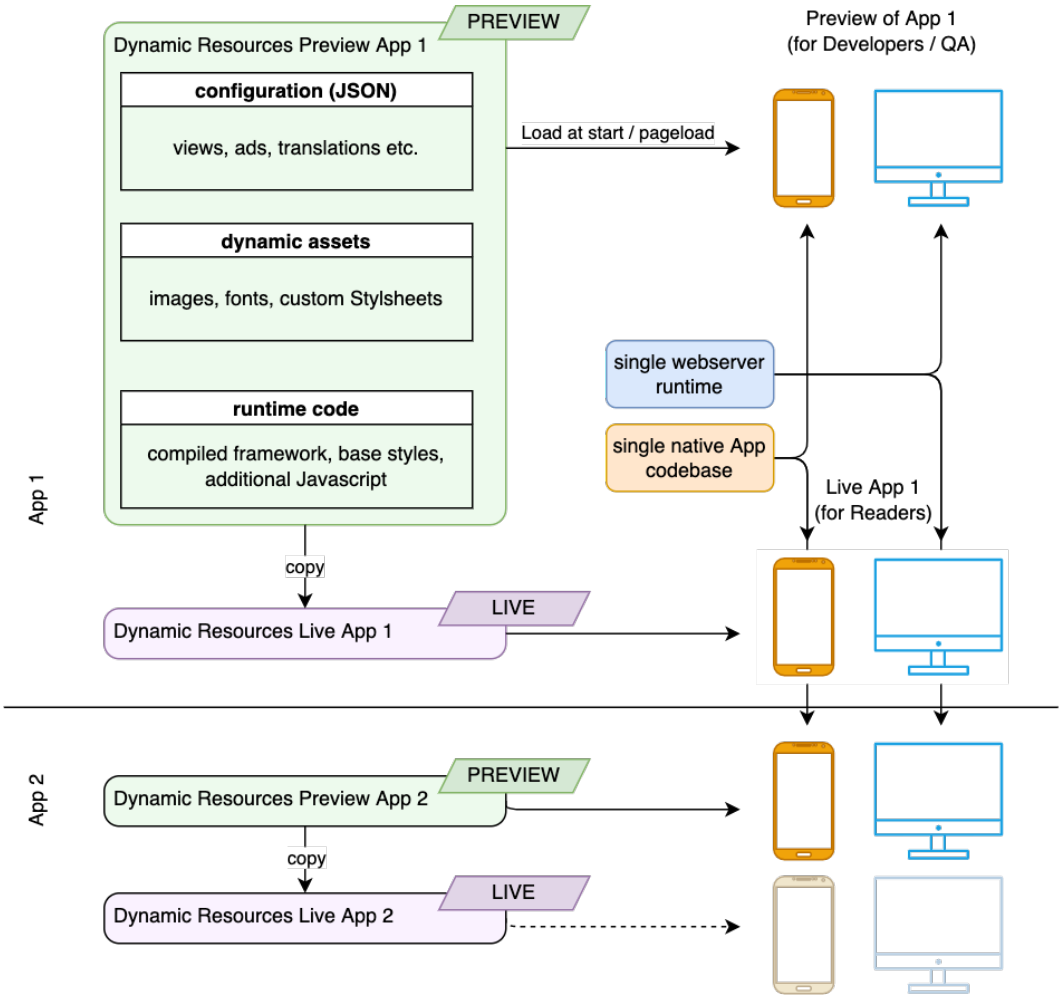

Figure 2.2.: Sprylab preview and live dynamic resources for two imaginary apps

Working with large and deeply nested JSON files quickly becomes convoluted. Manual handling of ZIP files, including modification of assets, repackaging, and the subsequent risk of introducing errors, is a labor-intensive and error-prone workflow.
As a base for making editing of these configuration JSON files easier, Purple Experience generates JSON Schema files directly from the interfaces in the source code for every released version. The use of UI elements generated from JSON Schemas allows for direct validation of user input and prevents the entry of invalid states.

At Sprylab exists a tool called ”Storefront Editor”, which is the predecessor for this new UI Editor. It uses an open source software called *Json Editor* (`https://github.com/json-editor/json-editor`), which is an implementation

of the UI elements for JSON validation mentioned above. In chapter 3, the state of the JSON Editor for that use case is evaluated and the positive aspects and approaches which I reused for the new editor are generally outlined. Furthermore, features are described which are missing, or users noted as confusing, not working or slowing down their work.

Figure 2.3 shows a minimal example of a view configuration with which the Purple Experience could render a simple page. The page would be accessible under the domain of the app; for example *https://app1.com/newsstand* and show a list of issue components. The data for these components is taken from the data source of type *issue* which filters the published content for issues that are purchased by the current user.

```
{
  "type": "view",
  "path": "/newsstand",
  "content": [
    {
      "type": "list",
      "content": {
        "type": "issue"
      },
      "dataSource": {
        "type": "issue",
        "filter": {
          "purchased": {
            "value": true
          }
        }
      }
    }
  ]
}
```

Figure 2.3.: Example of a view configuration showing purchased issues

In real apps, the configurations are more complex than shown above. Examples are conditionals declaring when components are rendered or filters are applied, event handlers when a user interacts with an element and many more functionalities are modelled through the JSON Schema.

2.3. Related Work

Primary sources for the HCI approach and methods were [2] and [8], which provided a broad view about user research methods and prototyping. For the implementation, most sources were the official documentations of the tools and libraries used. To get an understanding of JSON Schema, its structure and its possibilities, [9] provides a good introduction.

The two concrete topics however, building a UI Editor and covering HCI methods in Brownfield development, seem to be only sparsely covered. Despite widely distributed Interface-based UI editors like Wordpress or Wix.com, the scientific coverage of this combination "Model-driven developemnt" and "Page building" is low. Mostly papers about concrete implementations exist like [12] or [7], and some documents from the early era of the World-Wide web like [5], which describes the abstract approach of storing the website structure not as HTML, but an intermediate more domain-specific format and only generate the HTML when requested.

The same rarity exists for the combination of Brownfield software projects with HCI user research methods. Papers and articles that were used as sources for this thesis were e.g. [6] about Greenfield vs. Brownfield development in general and [1], which makes interesting points about the necessity of different approaches for the two development cases.

3. User research and analysis

Over ten years after the publication of Tomer Sharon's book "It's our Research", the list of quotes in the introduction about user research in software companies still feels as relevant as ever.
"Yeah, but this study will delay our launch date.", "Yeah, but we can not learn much from only five participants.", "Yeah, but research sounds so academic." [11, p. 4] are only some of the statements that according to Sharon are often heard in software companies in discussions about User Experience (UX) research.
The pressure from stakeholders often leads to quick implementation of features and workflows without figuring out the user's needs. This seems faster in the beginning, but can badly impact the user's acceptance of the product due to cumbersome and slow workflows, in the worst case it leads to the refusal of the whole product.

To counteract this, it is crucial to conduct user research methods and evaluate the user's needs. This fact is considered in the development of the UI Editor. A starting point for qualitative user research is to define the goals through the help of the SMART criteria, which provide guidelines and formulated goals during research. For the project, the SMART criteria were defined like this:

- **specific** - improve the workflow of users modifying dynamic resources for Purple Experience
- **measurable** - interviews after testing period concerning working speed, confidence and joy when editing resources, automated user tracking
- **achievable** - research and implementation will mostly be conducted by me, with input from CTO & product owner, connections to external users through customer service team
- **relevant** - new software platform which reacts quicker, provides more safety regarding errors and is scalable and extensible in the future.
- **time-bound** - the new software should have a feature set enabling productive work, replace the old *Storefront Editor* and be usable by company internal users until the end of 2022

Remembering these points was helpful when the focus on the actual goals was unclear.

In addition, Becker ([2, pp. 37-41]) states that there are “three major factors that an HCI designer should consider”. These three factors were seldom mentioned in publications but are helpful guidelines.

Usability Factor describes the spectrum from ”usable” to ”unusable” software. This is determined by the implemented software design features and how they support the user achieve their tasks in the environment provided.

Accessibility Factor is high when as many users from different backgrounds can use the software in different environments. It includes access for people with and without disabilities as well as low entry hurdles for new users. At first glance this factor does not look as important as the other two, especially for specialized applications, but it should not be left out when designing and implementing software.

Time-On-Task Factor refers to ”[...] solutions that use up the appropriate of time to solve a problem.” [2, p. 40]. Obviously, to save users time, a fast system is required. But it is inevitable that network delay computation tasks all require some time, and users understand that fact. More important is reducing the perceived lag of interactions and give users feedback if a task takes longer to complete.

3.1. Identifying and categorizing users and user groups

In order to effectively design and implement the UI Editor, it is crucial to understand the needs and preferences of the various users and user groups who will be using the tool. The first step in the process was to identify and categorize them. Then, detailed information is gained through the observations and interviews (3.2.1 and 3.2.2) and finally concrete Personas for the different user groups are built (3.5.2).
It was easy to collect a list of potential users for the new UI Editor because Sprylab already had users working with the ecosystem and the predecessor of the editor. A larger group of different users or external customers would require more effort to perform this step though.
The next step was grouping them to understand the characteristics and needs of each user group. Having this overview made it easier to tailor the UI Editor to their specific requirements so it can be used effectively by all users[1]. Also, when choosing who to use as research subjects this list can help to get broad coverage of users with different needs and abilities.
The information about the users leads to the following common factors:

Quantitative usage - Some users rely on the tools for most of their work, others a few times a year.

[1] “All users” refers to users that are expected to work with the tool. There is an expected technical and domain specific base knowledge that the UI Editor won't cover in its initial form

Common tasks - I roughly categorized the common tasks into three groups:

Heavy configuration - Build new apps and websites from scratch, making a wide range of modifications and structural changes.

Moderate configuration - Copying resources from templates and adapt them for new brands, which includes changing colors and logos, adapting texts or switching authentication flows.

Small changes - Exchange ads, translations or logos, which affects a small set of files.

Expertise -

Technical - Depending on the area of education and work experience in web development, knowledge about web standards and often also intuition differs between users.

Domain- and Platform Specific - Special vocabulary, functionality of Purple Experience and other systems as well as configuration tricks that users learn with time.

3.2. Qualitative user research

The existing user base, most of them at Sprylab, simplified access to participants for qualitative user research methods. Using one or multiple ways of triangulation [8, p. 264] can strengthen the significance of the research outcome. Limitations of method or source can be removed by variation of those, resulting in a less distorted picture. Therefore, methodical triangulation[2] as well as triangulation of data[3] seemed well suited. Moderated observations combined with interviews proved to be a good fit for this case study, as they are interaction driven and the observer can react directly on behaviors / emerging topics and steer the process. This stands in contrast to more passive methods like passive observations or user recording and tracking analysis, where the outcome only depends on the prepared question / task and the user's behavior and which can not adapt to changed circumstances etc. during the application.

The chosen structure for the observations and interviews looks like this:

Introduction (ca. 5min) used to explain the circumstances and the goal of the session, the method, which data will be collected and how they will be evaluated afterwards.

Moderated Observation (10-15min) let the observed perform specific tasks in a prepared environment

Semi-structured Interview (ca. 15min) ask open and closed questions and discuss observation situation

[2]Using multiple data gathering techniques

[3]Collecting data from different people and different sources

Interviews came after observations to allow discussing issues encountered during the moderated observation and to gain deeper insight into the workflow and potential problems.
To verify the methodical approach and whether the concrete questions fit the process and can yield desired information, it was important to pretest the whole process with a small group before scheduling the other meetings. One of Sprylab's working students agreed to be a test candidate and we verified the prepared tasks and questions I planned. Pretesting the methods and specific questions before conducting them on a broader audience helped to sharpen them by identifying ambiguous, repetitive or already known facts. For the tasks of the moderated observation the test gave feedback on the difficulty, time the sessions would probably take on average, which tasks needed clearer formulations and which were already sufficient.
The user research was conducted with six persons, including a core Purple Experience developer, two working students from our project department, one developer from a different department who had worked with the software some time prior, one Customer Support Manager from Sprylab and one external user from a publisher.
An interesting concept relevant for the choice if subjects from [3, p. 41] is the Subject Matter Expert (SME), who represents "authorities on the domain on which the product will operate." [3, p. 41]. The SME is important for the process to give input on technical details and patterns other less involved users might not know. For this case study, the Purple Experience developer represents the SME for the underlying platform the UI Editor is built for.

3.2.1. Moderated observation

For the moderated observation a list of six tasks was prepared. Two of them were optional depending on the time left and how familiar the user felt with dynamic resources. The same first tasks could be given to every interviewee regardless of their level of knowledge, the optional tasks if there was still enough time. Also, it was important that the tasks did not build on each other to prevent the observed person getting stuck because of an earlier mistake. Instead, seeing a variety of tasks getting performed gave more insights than one task being executed without errors. Therefore, an example app was created on Sprylab's staging system, so it was easy to prepare and also reset after each observation session.

The six tasks were:

1. Change app menu entry "Newsstand" to "Home" on all platforms (Web, Android and iOS).
2. Change the advertisement banner target on top of the home page to https://google.com.
3. Change the text "Latest Issues" on the home page to "Read new Issues".

4. Change color of ”Read new Issues” and ”Latest Articles” headers on the home page to the app’s primary color..
5. *(Optional)* Add a dropdown on the home page between ”Read new Issues” and ”Latest Articles”

   It should show all publications connected to the app

   It should set a URL parameter “publication” to the id when selected

   Define the reset message as “All publications”
6. *(Optional)* Configure the filter of the “Latest Articles” list to only show articles from that publication

These tasks were constructed in a way that errors could occur and were expected, which allowed to see how users try to figure the cause of the problem out and how to fix it. As expected these cases then also occurred, for example the first task was often done for only one of the three platforms, the “Latest Issues” text (task 3) was not found in the translation files or the color change was applied to more elements than desired (task 4).

The optional tasks were presented to four people, of which three solved at least one task successfully, only the core developer solved all six tasks completely. With the consent of the interviewees I recorded their screens during the observation to rewatch specific actions or flows if required.

3.2.2. Interview

For the interview, a semi-structured interview was chosen as the appropriate tool. With a list of open and closed questions the interviewer can ensure that important topics are covered. Furthermore, it allows flexibility to delve deeper into specific issues and ideas if needed.
The prepared questions consisted of closed ones, like how often they interact with the tools, how confident they feel implementing changes or fixing errors, and open questions like to describe step by step what the most recent task was they performed with dynamic resources, how they were onboarded etc. If notable points occurred during the observation, these were addressed now either directly or in form of an open question like “If you imagine a UI builder with no technical limits, how would it look like, which features would you expect and what workflows are most important to you?”.
The semi-structured interviews proved to be valuable in gaining insight into the experiences and needs of the users. They made it possible to gather lots of new information and show how different features were massively valued differently by the users. Two interviewees even provided written lists of their ideas and suggestions afterwards. Some outcomes can be found in 3.4 and Appendix B.

3.3. Quantitative user research

At the initial stages of research, the use of questionnaires was considered as a method of gathering information from existing and potential users. However, as the outcome of the qualitative research was already productive, it proved difficult to design a questionnaire that would provide additional information while adhering to common design rules for questionnaires in terms of length, number and type of questions. Prototype questionnaires were created using Microsoft Forms, but they were too long, difficult to understand, or not relevant to the development of prototypes. Therefore, it was decided to rely solely on qualitative research results for prototyping and feedback, along with some automatic tracking implementation (see 5.6).

3.4. User research outcomes

After each observation and interview, which were additionally recorded, the interviewer noted facts about the subject and the session in a prepared template. The mandatory fields consisted of basic information like the role of the user, frequency and typical tasks with the resources. Additionally, important situations from the observation, a description of the workflows, input from the interview and the linked recordings were noted.

A shortened and anonymized version of the notes can be found in the Appendix B. Derived from these notes, combined with the technical requirements, the following subset of essential features emerged:

App Selection

- Having quick access to recently opened apps, search by App name, App ID and Publisher Name.

App Editing

- Automatic downloading and packing of the dynamic resource ZIPs so manual pack/unpack operations are not required by the users.
- Support for syntax highlighting and autocomplete for common files.
- Adapt the workspace to the users needs, hide or resize components.
- Fast load times or the possibility to have multiple files open simultaneously.

App Preview

- See a preview showing the latest changes instantly, load ”real” data from the API.
- Multiple ways to validate changes: preview on the side, in new window or quick merge into preview for “real world” tests.
- Ability to load current state of config and assets inside an app connected for debugging.

Dynamic Resource Change Management

- See which files were changed since last upload.
- Reset single files to an earlier version.
- Have version control features like ”commit messages” and version diffs.
- See who uploaded new changes and if changes are being processed.

3.5. Process and visualize the research outcomes

To gain as much value from the raw research data it is crucial to find methods to process and visualize the outcomes. A common approach at companies is to write ”tickets” in their ticket system of choice, in case of Sprylab Jira (`https://www.atlassian.com/de/software/jira`). From my experience, Jira and co. prove valuable to track progress during development. It is not suitable for discovery and prioritization of features at the beginning because lists of tickets with different priority levels and different sorting do not give a good overview over effort and benefit of each feature. Thus, based on the notes from observations and interviews two alternative ways were tried to figure out different features and how they can be prioritized.

3.5.1. 2x2 Opportunity Matrix

Fig. 3.1 shows a two-dimensional visualization of a subset of proposed features. It proved helpful when prioritizing tasks with other stakeholders, as it shows the approximated cost of implementation as well as the value the feature can have for users. Costs include possible financial costs as well as required time and amount of resources needed to implement the feature.

It is a slightly modified adaption from [2, p. 181], replaced the term “idea originality” on the x-axis with “Value”. The benefit of visualizing multiple possible features is to see at one glance the relation between value for users but also cost of implementation in a more clear way than lists of tickets with different priorities would provide.

3.5.2. Building Personas

Personas are descriptions of fictional users of the product, incorporating assumptions and optionally data for a user group. They aim to give developers and designers more context and depict real potential users, which makes it easier for a developer to empathize with the user. For this case study three Role-based Personas were derived from 3.1 and the outcomes of the interviews, based on the description of Personas in [8, pp. 403-405]. They can be found in Appendix A and represent the users with different levels of expertise already identified in 3.1.

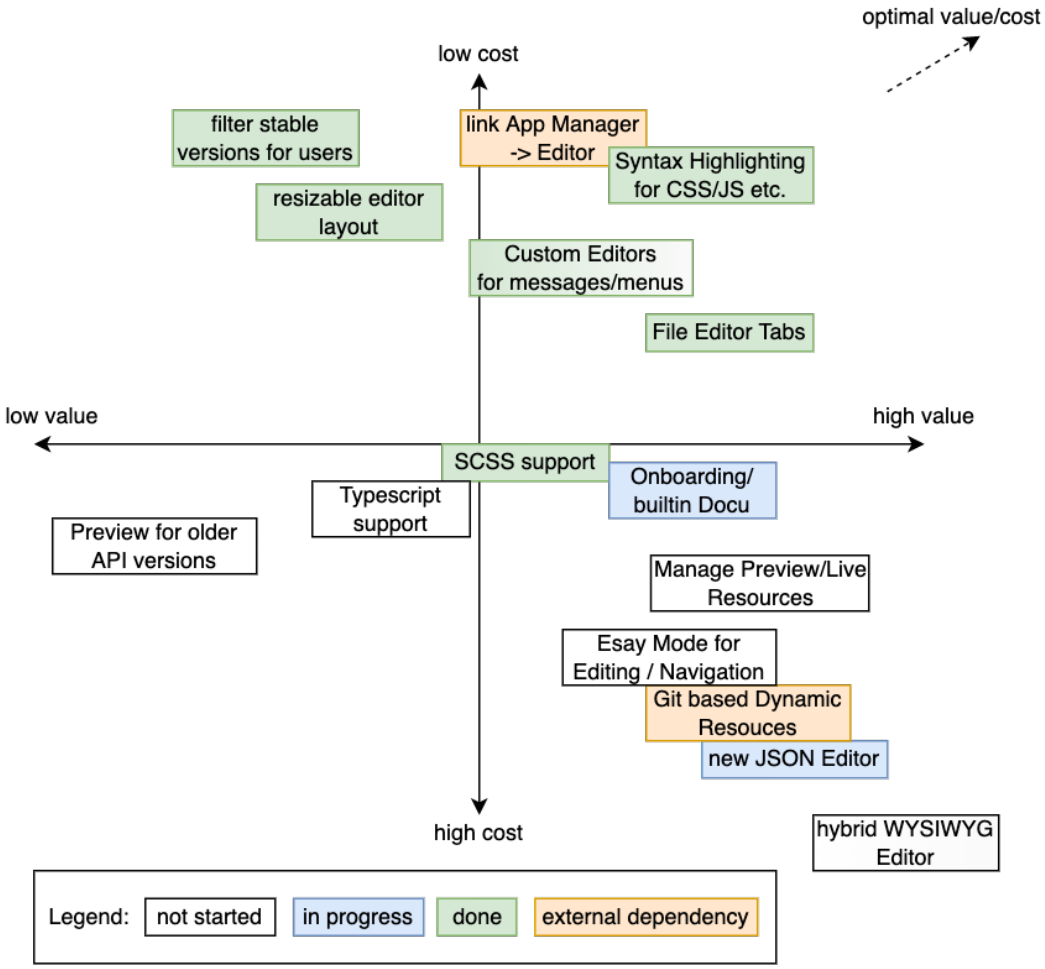

Figure 3.1.: 2x2 Opportunity Matrix (adapted from [2, p. 181])

4. Prototyping

After collecting the initial user feedback, digital "paper" prototypes were created using Figma[1] to gather visualizations of possible UI layouts. Two ideas emerged from the interviews: a (file-)editor-centric layout and a preview-centric layout.

4.1. Editor centric vs. preview centric layout

The **preview-centric layout** is inspired by popular generic website builders like `https://wix.com` or `https://wordpress.com`. A preview-centric UI editor, also referred to as WYSIWYG (what you see is what you get) editor, lets the user directly interact with a preview of the final layout. Often this means composing "elements" or "blocks" per drag and drop (1 and 2 in fig. 4.1), and when a component is selected, additional options of this component can be configured (3 in fig. 4.1). This approach requires little cognitive transfer from the user, since the preview is at the same time the source of truth of the configuration. On the other hand, this WYSIWYG approach is impractical when a lot of the data and parts of the UI are dynamically loaded and dependent on user state. For example, a common use case for Purple apps is to show special offers only on one platform or to hide components for users who are not logged in. Discovering and configuring these invisible elements can lead to confusion by the users.

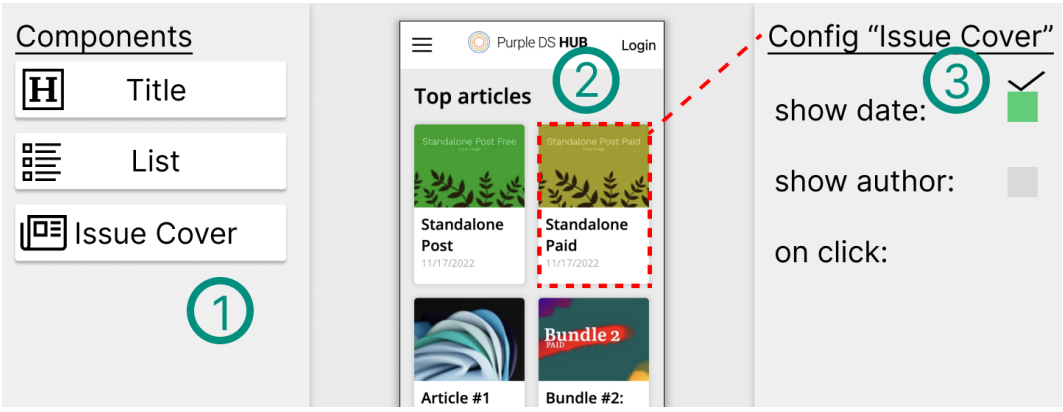

Figure 4.1.: Preview-centric prototype from figma.
1: component library, *2*: preview, *3*: options for selected component

[1]Figma (`https://www.figma.com/`) is a design tool accessible through the browser for collaborative work on design projects.

The **editor-centric layout** is inspired by modern text editors / IDEs like VS Code (`https://code.visualstudio.com/`), which often referenced in the interviews. This approach makes the underlying configuration structure more visible to the user, usually through text files. The central pane is the editor for opened files with side panes for file management and selection, preview and more. The familiarity of this layout, particularly to developers familiar with IDEs, could help new users adopt patterns they use in other tools.

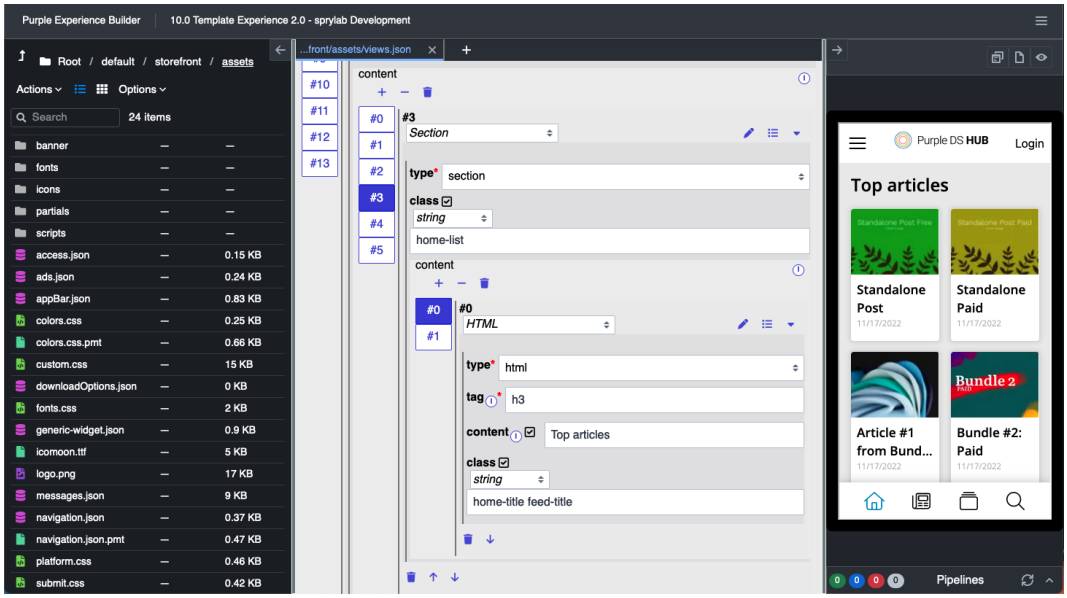

Figure 4.2.: Editor-centric layout

For the UI editor, the choice fell on the editor-centric approach, for which the reasons are explained below.

- Existing tools like `https://vwo.com/why-us/technology/visual-editor/` or WordPress are not suitable for the Purple Experiences structure, which was not designed for preview-based editing. The aforementioned issues with invisible or platform-dependent UI components are critical.
- Third-party libraries like Microsoft's Monaco Editor (`https://microsoft.github.io/monaco-editor/`) can be integrated to create editor-centric UIs while implementing a custom preview-centric solution within the limited time frame of this thesis seemed more uncertain and complicated.
- The user base consists mostly of tech-affine people who are used to layouts of IDEs. As Jakob's Law of the Internet User Experience states, the user's understanding of a website is directly tied to their mental model of that system [10] and [13, p. 2]. Introducing an unconventional workflow comes with the danger of confusing the user, causing mistakes, and potentially leading to dissatisfaction with the tool.

5. Implementation and deployment

This chapter describes the agile development process used for implementation of the UI Editor and gives examples on features. The code can be seen at `https://git.imp.fu-berlin.de/matthiak00/thesis-ui-builder-snapshot`[1].

5.1. Architecture

The architecture and high-level user flows of the three software components relevant for the UI editor include the frontend, backend, and Purple Manager backend. The Purple Manager backend handles authentication, app management, and providing dynamic resources as ZIP files. The user journey begins with logging in at the root domain (e.g. `https://builder.purplemanager.com`, but the details of authentication will not be covered as it is not relevant for the user experience. The REST API of the Purple Manager and the dynamic resource management are basic technical requirements set by the surrounding ecosystem. Fig. 5.1 displays a typical interaction of a user with the editor frontend after selecting an app; pulling the latest dynamic resources, editing a file and merging the changes.

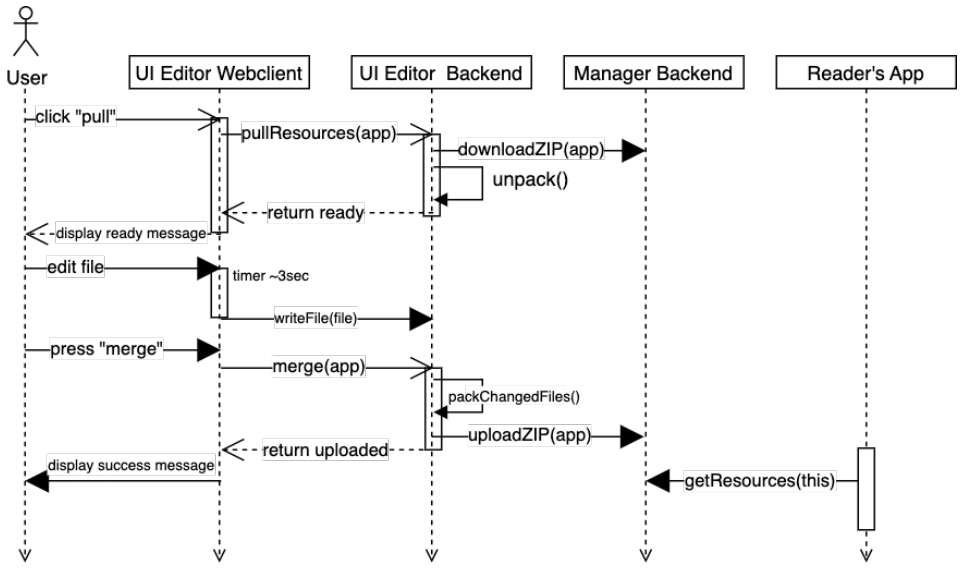

Figure 5.1.: A typical interaction of a user with the UI Editor

[1] snapshot from Sprylab's Gitlab on 2023-01-25, accessible for members of the `https://git.imp.fu-berlin.de` instance

5.2. Software stack

From a company's perspective, it is advantageous to keep the software stack as small as possible. For this case study, the most important point was the availability of additional personnel with knowledge about the frameworks and languages used. To conform with the company's DevOps practices the only requirement is to run the project inside Docker containers in a Kubernetes environment.

For this project the Typescript language (`https://www.typescriptlang.org`) is used on back- and frontend. The advantage is the availability of skilled personnel in the company as well as an established ecosystem and easy sharing of code and type definitions between frontend and backend. The following list gives an overview of the most important tools used:

Frontend

Rendering framework – *React JS v18* (`https://reactjs.org/`)

UI Component Library – *Blueprint JS* (`https://blueprintjs.com/`) provides components so a consistent design system with common functionality like buttons, dropdowns, filters etc. can be used.

Other libraries – *ReactQuery / TanQuery* to manage, cache and invalidate HTTP API requests to the backend, *Zustand JS* for shared reactive state management and *Zod* for type validation at runtime.

Backend

HTTP Server – *Express* on *Node JS*, which is the most common combination to run an JavaScript based HTTP server (see [4]).

Routing abstraction – *TSOA* (`https://github.com/lukeautry/tsoa`) on top of *Express*, which is a Typescript library to provide Java-Spring like syntax with controllers, dependency injection and parameter validation at runtime.

Testing – *Vitest* (`https://vitest.dev/`) for unit tests and *Playwright* (`https://playwright.dev/`) as E2E test runtime.

DevOps – *Gitlab Pipelines* to build, test and package on every merge request or commit to develop and master branch.

Project Setup – Monorepository with *PNPM* as package manager and *Turborepo* to manage package dependencies and automatic optimal build scheduling and caching.

5.3. Continuous Integration / Continuous Delivery

Continuous Integration and Continuous Delivery (CI/CD), a core practice of agile software development, is crucial for agile prototyping and development and enables fast release and deployment cycles. A Gitlab CI/CD Pipeline was set up for the UI builder, consisting of build, package, and deploy stages. The build stage also executed unit and end-to-end tests due to technical reasons for efficiency.
Having a fast and reliable CI/CD process during development and prototyping was valuable, as it allowed quick response to user feedback and deployment to a staging domain in under 10 minutes. Separating staging and production systems allowed more confident deployment of quick fixes for validation in a production-like environment without interrupting users.

5.4. Feature examples

A selection of features implemented for the UI Editor is presented in the following section. These features are examples of how the HCI methods and outcomes from the user research phase influenced their design and how they can improve the user's experience.

5.4.1. File management - multiple file tabs

A common workflow consists of editing multiple files simultaneously, for example having the view configuration open while adding translations for newly added components. The old editor tool allowed opening only one file at a time, which got closed automatically when the user opened another file. During the moderated observation this showed up as a big slowdown. All participants mentioned that they have to work on multiple files and are annoyed by this workflow. Especially the opening of large files can take more than 30 seconds. The solution was inspired by the file management that most IDEs provide; a bar on top where all the opened files are listed so the user can quickly switch between them or even close not needed ones (see fig. 5.2).

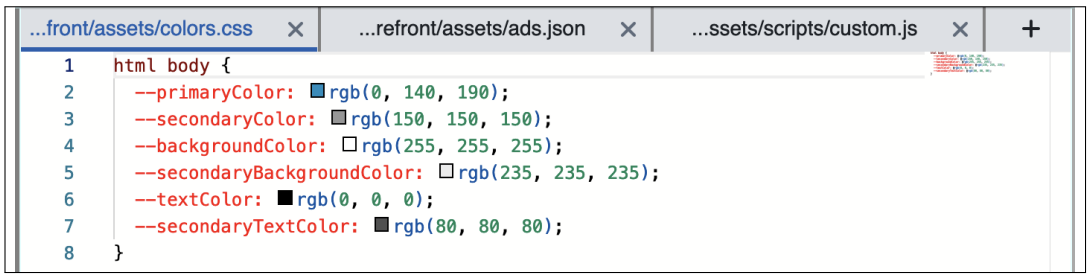

Figure 5.2.: Screenshot of file tabs

5.4.2. File management - quick links

A second feature regarding the file management is "quick links". While all target groups benefit from this, especially for the personas *Steffi, A.1* and *Karsten, A.3* this can speed up common tasks considerably.

The basic idea is to bookmark frequently used files, e.g. the translation or ad config, and have them prominently available when opening a new app. For the implementation, three questions needed to be answered:

Where to save? The quick links are stored in the LocalStorage of the user's browser, so the list is available across apps.

How to manage? The user has a list on the settings view, where entries can be added, deleted and changed, but the file path needs to be inserted manually. A screenshot of the settings can be seen in Appendix C.3. To enhance the user experience, a proposal exists to either add a file picker in the settings or add a bookmark button to the file manager.

How to present the links? The links must be easily accessible to provide the intended benefit. The solution was to show them prominently when entering the edit view and no file was opened yet. Based on user feedback, each link shows via an icon whether it links to a folder which the file explorer navigates to (1), to a file which gets opened in a new file tab (2), or if the path does not exist in the current app (3).

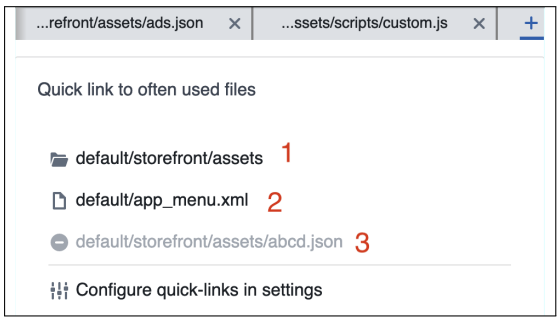

Figure 5.3.: The three types of quick links visualized with icons

5.4.3. Editor - Abstraction to provide per-file custom editors

Purple Experience relies on a variety of configuration files, all with different schemata and functional intents. To provide an efficient and error reduced workflow to users, it is important to have different specialized file editors.

For view configurations for example, the JSON Editor Library (`https://github.com/json-editor/json-editor`) is used, combined with the generated JSON Schema files.

For the translations there exists a custom editor that has a fuzzy search function and makes it easy to manage translation entries.

The abstraction is done solely in the frontend and follows React's "Composition over Inheritance" pattern (see fig. 5.4). Due to modern React JS architecture, component classes are replaced with functional components (marked with $\ll functional \gg$). To integrate a new specialized editor type, the only two steps are necessary: Creating a new functional component that implements the *EditorImpl* interface and adding that editor in the *EditorRegistry* to get returned for the file paths requested. Examples of custom editor implemen-

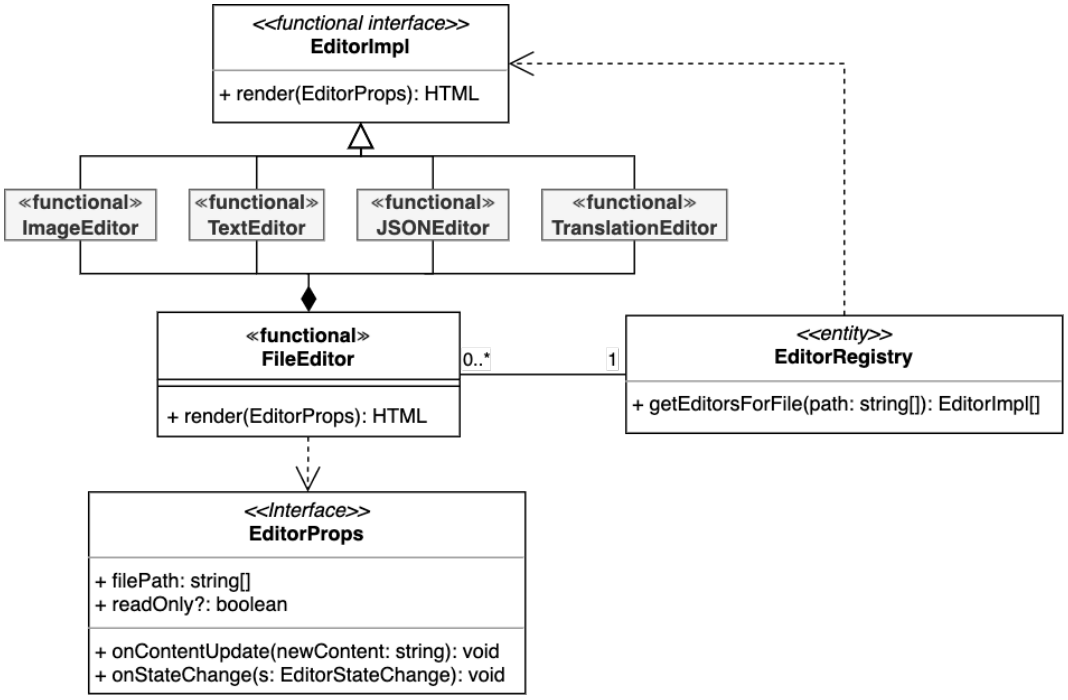

Figure 5.4.: Class diagram showing the editor abstraction in React JS.

tations can be seen in Appendix C.1 and C.2. By implementing Editors for multiple levels of expertise, users like persona "Karsten" (A.3) can make small adaptions like changing a translation without worrying about the underlying file format and without the possibility to break the app due to syntax errors.

5.5. Automated Testing

Automated tests contribute to the confidence of developers to deploy more frequently and can reduce load from the QA team to test common workflows over and over. Two of the most common testing levels were used for the project: unit tests and End-to-End tests (also known as E2E or System tests).

Unit tests check one "component" in a sandboxed environment, often done per-function or per-class. On the client, both UI components and business logic code that is encapsulated in classes or JavaScript modules get tested.

They also enable Test-driven development, where the tests are written upfront based on specifications and constraints with invariants, pre- and post-conditions and the code is continously tested against them.
For agile development, the speed of the test execution and CI/CD Pipelines is important. The developer is blocked during that time and can not progress on the task. Therefore, the aim was to find a fast runtime, compatible with UI and browser testing as well as Node JS for server code. After evaluating commonly used test frameworks, Vitest (`https://vitest.dev/`) was chosen which fulfills all the requirements. Tests can be hot-reloaded and re-executed on change in less than a second, providing a good developer experience.

Due to the complexity of interoperation between components, APIs and the Web standards, unit tests alone are not enough to guarantee the different modules work together as expected.
E2E tests are supposed to cover a typical user interaction with the service to validate the interaction between business logic components, UI and the user. First a headless browser[2] (`https://pptr.dev/`) was used in combination with Vitest, but writing and debugging tests proved to be slow and error-prone.

Later in the process, the E2E tests were rewritten for an alternative test framework called Playwright (`https://playwright.dev/`), which allows recording a test case in a normal browser window and then generates the test code automatically, allowing to be adapted and generalized if needed. After the tests were ported to the new framework, developers can enjoy better debugging tools and the ability to add new tests easier. This may be an example to others that investing time in researching new tools and refactoring code is worthwhile, increasing developer experience and, in turn, speed and confidence.

5.6. Privacy-friendly analytics & monitoring

Sooner or later, you find yourself in a pickle when it comes to tracking. On the one hand, the data can provide valuable insights into the behavior of many users, which would not be possible through qualitative research. On the other hand, the most common tools like Google Analytics track users with cookies, which creates new challenges regarding GDPR compliance.

The Purple product owner proposed `https://squeaky.ai/`, a cookieless solution to track users on this page. While it can not provide the same level of demographic information a cookie based solution can, it still collects the most important data like usage statistics, user interactions and occurred JavaScript errors. It can even show heatmaps per page to visualize which UI elements the user interacts with the most.

[2]Headless browsers are browser instances that do not render the actual content to a user's screen, but run as a CLI application and still execute all JavaScript, CSS and HTML.

These heatmaps for example can show whether a new UI feature is used and how users interact with the page in general. Fig. 5.5 shows that users mostly interacted with the file explorer and only did a few clicks in the editor panel.

Figure 5.5.: Screenshot: Squeaky.ai heatmap of an app's edit view

To analyze the usage growth, fig. 5.6 indicates how the number of page views per week increased steadily since the internal beta version was launched (except the holiday dip). The graph clearly shows that the time-bound goal from the SMART criteria in chapter 3 has been successfully achieved.

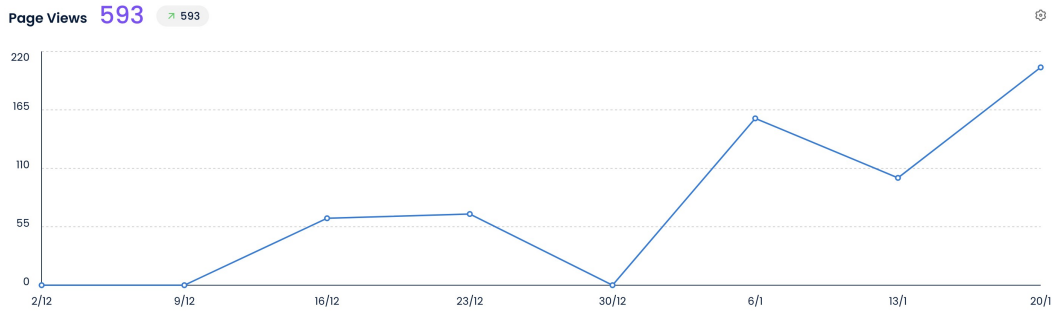

Figure 5.6.: Screenshot: Squeaky.ai page views since December 2022

5.7. User Testing, Feedback, Beta and Monitoring

After a first stable deployment was available for stating and production dynamic resources and all basic requirements were met, the user tests for the Beta-phase started. Four people from our company agreed to use the UI Editor and give verbal feedback. Three of them were already participants of the interviews and they covered all mayor groups of users, so the feedback represented all different expertise levels.

The reported bugs were written into Jira tickets so the status was properly tracked and release notes could be generated. Mostly the bugs were edge cases where a file wouldn't open or the changes were not saved properly.

5.8. Communication and Documentation

Communicating with the test users and documenting the technical aspects of the software, the progress and how to use certain features is another building block towards a good user and developer experience.

5.8.1. Asynchronous communication

For the asynchronous communication, a Microsoft Teams Channel was utilized, a group chat where all invited persons (in case of the internal beta-phase company-wide) can write to each other and create posts. This was used for notifications about new deployments and information about bugs that could affect multiple people.

Even though this adds a bit overhead, it proved easier than having everyone write bug tickets directly, as user often do not know how to describe the problem which leads to unclear descriptions, wrong tags and duplicate tickets. Instead, a developer with knowledge about the system looked at the reports and created a new ticket if the problem was new, otherwise referenced an existing ticket or forwarded the problem to the responsible team. Jira tickets were all assigned to the UI builder component as well as specific releases so everyone can see with one click which changes and fixes are contained in which release.

5.8.2. Presentation and Beta kickoff

At the point where the company-wide beta phase started, a presentation meeting was scheduled where I first explained the project, demonstrated a common work flow and some of the most important features and could directly respond to questions or feedback. Around 30 employees attended, including quality assurance and customer success managers, covering a wide range of people who possibly will come in touch with the tool.

5.8.3. Documentation

For the feature documentation, Sprylab uses Confluence for internal documentation and Archbee for external documentation that customers can access too. As we have the common problem of documentation getting postponed indefinitely, I tried to integrate writing documentation directly int o the development flow and only close a ticket if the documentation was written and reviewed by one additional person. Due to time and personnel shortages, this was unfortunately not always possible and will require more future attention.

6. Conclusion and outlook

This thesis demonstrates the challenges and opportunities of applying HCI principles and methods in a brownfield software development project through the redevelopment of a UI Editor for a digital publishing company, Sprylab. During the process, it was shown how common user research methods can be adapted to this specific case study and how technical limits and needs of users can be combined in this context. Agile development and Lean UX enabled iterating quickly and achieving fast deployments to a staging system. A growing group of users at Sprylab already uses the software in production.

The consistently positive feedback shows that the chosen methods and features derived from their outcome were the right approach to enhance the user experience while complying with constraints imposed by the ecosystem. Also, the way users were involved during the whole process led to a high level of acceptance and interest in supporting the project. Additionally, this thesis can help to argue for the use of HCI methods in future projects started at Sprylab as well. I'm confident that this new software is a stable and extensible tool, especially regarding the two factors Usability and Time-on-Task. In addition to increasing user satisfaction, it also leads to enhancing productivity and customer satisfaction.

TODO: the editor enables users to safely edit JSON configurations and change assets and styles while directly seeing these changes in a preview frame

However, there is still room for improvements in terms of accessibility and entry hurdle for new users. The entry hurdle is still high and a lot of background knowledge is assumed, which is partly due to the complexity of other systems in the ecosystem that were set as technical requirements from the beginning on. Moving forward, replacing the JSON editor with a custom implementation that combines JSON Schemata and generated UI is one of the next steps, which will speed up the user's workflow even more and allow for further improvements that are impossible with a third-party library.

Overall, this thesis has demonstrated the importance of considering HCI principles and methods in brownfield software development projects, and the potential benefits that can be achieved when these approaches are applied in a real-world context.

Bibliography

[1] Johanna Wallén Axehill et al. "From Brownfield to Greenfield Development – Understanding and Managing the Transition". eng. In: *INCOSE International Symposium* 31.1 (2021), pp. 832–847. ISSN: 2334-5837.

[2] Christopher Reid Becker. *Learn Human-Computer-Interaction*. 2020. ISBN: 978-1-83882-032-9.

[3] Reimann et al. Cooper. *ABOUT FACE - The essentials of interaction design*. Wiley, 2014. ISBN: 978-1118766576.

[4] Vano Devium. *Top Node.js Web Frameworks*. URL: `https://github.com/vanodevium/node-framework-stars` (visited on 12/13/2022).

[5] Piero Fraternali and Paolo Paolini. "Model-Driven Development of Web Applications: The Autoweb System". eng. In: (2000). URL: `https://dl.acm.org/doi/abs/10.1145/358108.358110` (visited on 01/17/2023).

[6] *Greenfield vs Brownfield: Understanding the Software Development Differences*. URL: `https://www.johnadamsit.com/software-development-greenfield-vs-brownfield/` (visited on 01/02/2023).

[7] Sandi Harageones. "UX/UI Research-Applied Strategies for Building a Custom WordPress Theme Using Underscores and Elementor". In: (2022). URL: `https://digitalcommons.lindenwood.edu/theses/304/` (visited on 01/23/2023).

[8] Jennifer Preece Helen Sharp Yvonne Rogers. *Interaction design - beyond human-computer interaction, 5th Edition*. Wiley, 2019. ISBN: 978-1-119-54725-9.

[9] et a Michael Droettboom. "Understanding JSON Schema - Release 2020-12". eng. In: (2022). URL: `https://json-schema.org/understanding-json-schema/UnderstandingJSONSchema.pdf` (visited on 11/17/2022).

[10] Jakob Nielsen. "End of Web Design". In: (2000).

[11] Tomer Sharon. *It's our reserach: Getting Stakeholder Buy-in for User Experience Research Projects*. Elsevier Inc., 2012. ISBN: 978-0-12-385130-7.

[12] Marius Feldmann2 et al. Tobias Nestler1. "The ServFace Builder - A WYSIWYG Approach for Building Service-Based Applications". eng. In: (2009). URL: `https://link.springer.com/chapter/10.1007/978-3-642-13911-6_37` (visited on 01/17/2023).

[13] Yon Yablonski. *Laws of UX*. O'Reilly, 2020. ISBN: 978-3-96009-156-1.

List of Figures

Glossary

Agile is a methodology for software development based on iterative and incremantal delivery, providing flexibility and rapid response to change. 2

Brownfield development is the opposite of Greenfield development, brownfield refers to adding "new capabilities on an existing product or product platform, using existing technology" [1]. 1, 3

Composition over Inheritance is a principle originating from object oriented programming to instances of classes instead of extending them so they are more loosely coupled. React, while beeing functional, has a simmilar pattern: `https://reactjs.org/docs/composition-vs-inheritance.html`. 23

Design Thinking is an approach to solve problems or develop new ideas through experimentation and iteration, focused on creativity and collaboration.. 2

DevOps is a set of practices, tools and development culture to reduce time and friction between writing software and deploying it to production, often utilizing automated workflows. 20

Greenfield development Greenfield- and Brownfield development refer to software development concepts, where Greenfield projects start in a new environment and do not have legacy code, while brownfield projects are about upgrading or redeveloping software in an existing environment [6]. 1

IDE , short for Integrated Development Environment, is a common application used in software development, build around a text editor combined with tools for analyzing and distributing the software. 18, 21

JSON Schema is a specification and a declarative language "that allows you to annotate and validate JSON documents." (Retrieved 2023-01-11, from `https://json-schema.org/`). 5

Kubernetes is an open-source container orchestration system for automating the deployment, scaling, and management of containerized applications. 20

Lean UX apporaches development of good user expeirences through fast iteration and quick validation of design decisions. 2

LocalStorage is an API to a persistant key-value storage in the browser. It is scoped per domain, so no external pages can access the contents, but is persistent between browser sessions. 22

Purple Manager provides functionality to create, configure Purple apps, their content, entitlement systems and more settings related to app and website management. 19

Purple Experience is a proprietary meta framework built on top of Angular for the use in apps and websites for publishers of magazines and news. e, g, 4, 9, 18

REST API is an Web (HTTP) API paradigma to provide consumers of the API an standartised and esay to access endpoint structure and structured response data in a common machine-readable pattern. 19

SaaS (Software as a Service) is a software distribution model where a provider hosts the software on own servers (or in the cloud) and provides the users access via the internet. 1, 3

UX describes all the impressions a user has when interacting with the product. 9

A. Personas

A.1. John - Purple Experience Product Developer

A.1.1. Background and Skills

John (34) is a senior Angular Web Developer at Sprylab, working there for two years. He was born in Berlin and lives in Lichterfelde with his wife and mostly works from home. He is passionate about Angular, Typescript and Developer Experience in general, studied Computer Science at the Beuth Hochschule and hosts Angular conferences.

A.1.2. Goals and work with the Editor

- Test newly developed features and the related configurations
- Configure test apps for development and QA purposes
- Support in case Project Developers like Steffi (A.2) encounter problems
- John works with the editor multiple times a week

A.2. Steffi - Project Developer

A.2.1. Background and Skills

Steffi (23) studies media informatics and works as a working student at Sprylab for a year. This is her first job in the industry and she learns new things every day. Her skills include writing CSS and understanding modern web technologies, but she still struggles using native and custom debugging tools if something goes wrong.

A.2.2. Goals and work with the Editor

- Configure new apps based on existing templates and adapt them to customer's requirements
- Add new components or change data sources for existing apps
- Add custom HTML pages or JavaScript snippets to intergrate external services
- Change styles, color schemas or icons when a customer has a rebranding
- Steffi uses the editor as a primary tool for her work

A.3. Karsten - IT department at a publishing house

A.3.1. Background and Skills

Karsten (46) worked in the publishing industry for 20 years, but only during the last years his company, aga magazine publisher, tries to catch up with the digital development and trends. He is still struggling with his role and is thankful for every trick or tool that makes his life managing the digital products easier.

A.3.2. Goals and work with the Editor

- Exchange logos and colors when the magazines he supervies get a redesign
- Add new ads to different views when a new campaign starts
- Manage URLs to external sites when they change
- Karsten uses the Editor once a month on average

B. Raw interview notes

Date:	6.10.22
Usage:	”all the time”
Role:	project dev
Works for:	ca 8 months
Common task:	task: general structuring of websites & apps

Current workflow:

- Opens editor with history search of chrome
- Parallel opens app / website
- does Styling changes on the website / app itself in dev tools
- copies changed files back into editor
- → not using preview

Uses manager UI for:

- seeing what file was merged
- reverting changes

First impression: overwhelmed of complexity, felt unintuitive
Onboarding was playing around with test app, no explicit storefront onboarding

Common problems:

- views.json broken, no info what is the problem
- not seeing what changed inside files, have to remember everything
- No “commit” message

Quotes: “Man muss sich dran gewöhnen, man muss sich selbst ein System bauen“ - problematic as systems of different users might be incompatible.

Observations during moderated observation

- views.json didn’t work, he did not know why
- used “copy raw json text” a lot -> indicates that editing json is faster than editing via extra tools
- used preview popout

Ideas for improvements

- Resizable file explorer
- See if other people are editing same file / app?
- In upload name set “changed file” and “change component”?
- Link manager -> editor
- Quick links to common files
- Easy mode

Date:	11.10.22
Usage:	”every day at least once”
Role:	customer success team
Works for:	multiple years
Common task:	Small adaptions for customers, setting up new apps from cloned resources

Current workflow:
- Opens editor with shortcut
- Only used popout preview
- does Styling changes on the website / app itself in dev tools
- copies changed files back into editor → not using preview

Uses manager UI for:
- Versions Lock & Copy to release
- Comparing files (download, unzip, use extra software to compare folders)

Common problems:
- views.json broken, no info what is the problem
- slow performance
- “internal”: a lot of versions copied around (in thesis: filter input relating to other software)

Observations during moderated observation
- No overview over overwrites of different platforms
- resources not cleaned up -> no templates

Ideas for improvements
- Upgrade Storefront: only show official released versions by default
- Person does not like preview on side (Ref. working student uses it all the time!)
- Uploads should be included in merge
- Upload folders recursively
- hide mac specific files
- JSON validator to show detailed where the error is

Date:	12.10.22
Usage:	”most of work in dynamic resources”
Role:	project dev / support customer success
Works for:	1 year
Common task:	web kiosk migration & bug fixes, small views adaptions, styling, impression tracking

Current workflow:
- open through history + typing name in URL
- search app by manager name
- Noticed challenge if mod. observation task to see all platform overwrites
- Flow for renaming of translation → views-json → messages.json cumbersome

- Merge without clear description for others what change contained

Uses manager UI for:

- merge problems
- reset after broken merge

First impression: pair ”programming” sessions on extra test app, lots of trial and error

Common problems:

- upgrades not automatically, lots of manual steps involved -> more change of errors
- App Menu changes not testable without restarting real app
- merge not reliable, deleted files do not get deleted from ZIP file
- updates not transparent what changed

Observations during moderated observation

- Heavily uses side preview (ref to customer success member who does not use it at all)
- preview → right click → refresh frame

Ideas for improvements

- change log
- hide file manager completely if not needed
- “Ansonsten mehr oder weniger zufrieden” - more or less happy

Date:	11.10.22
Usage:	”depends on task, if working for Purple most of time”
Role:	project dev / other business unit
Works for:	ca. 1 year with breaks
Common task:	building new apps for customers

Current workflow:

- always open tab, a person that never closes tabs :)
- navigate to file
- multiple browser tabs to work on files in parallel → can lead to inconsistencies in data or empty files

Uses manager UI for:

- reset of broken changes
- see when data is ready to be delivered to apps / websites

First impression: -

Common problems: -

Observations during moderated observation:

- missing link manager → App is taking much time to figure out URL + app name
- navigation between files slow
- auto link view key → messages.json when adding new translations

C. UI Editor Screenshots

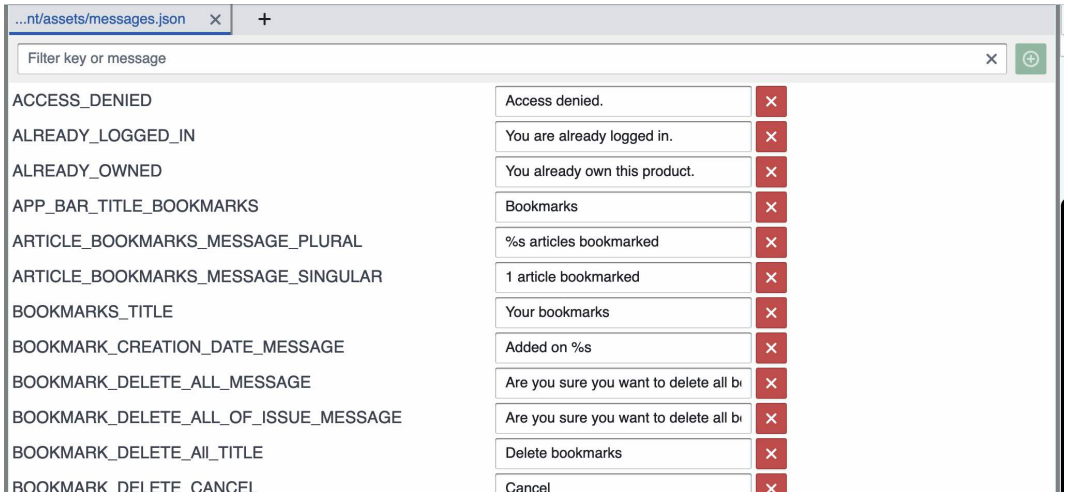

Figure C.1.: Custom Editor implementation to enhance UX for editing translations.

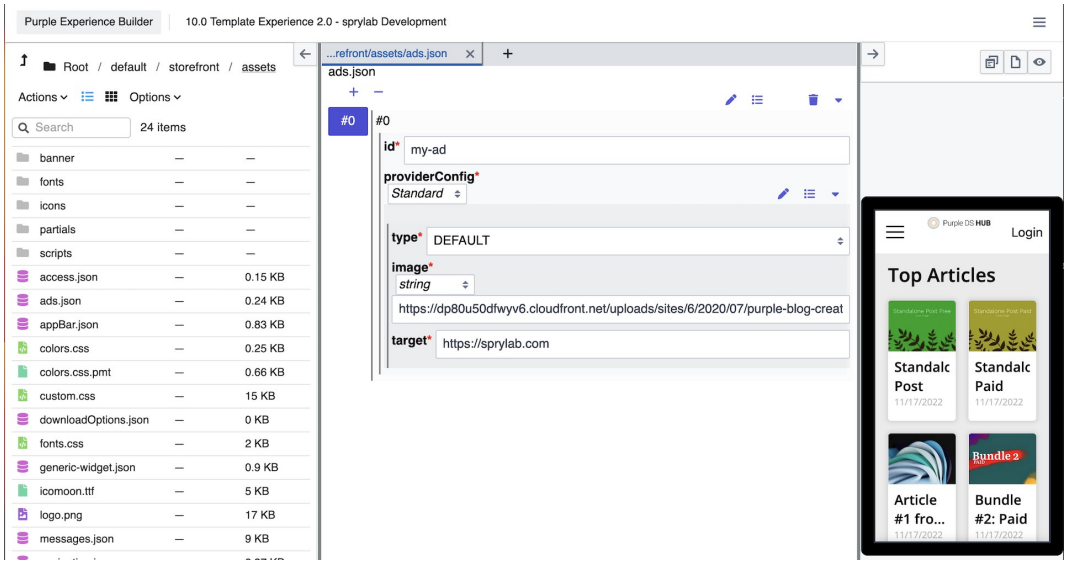

Figure C.2.: Custom Editor: JSON Editor when configuring ads

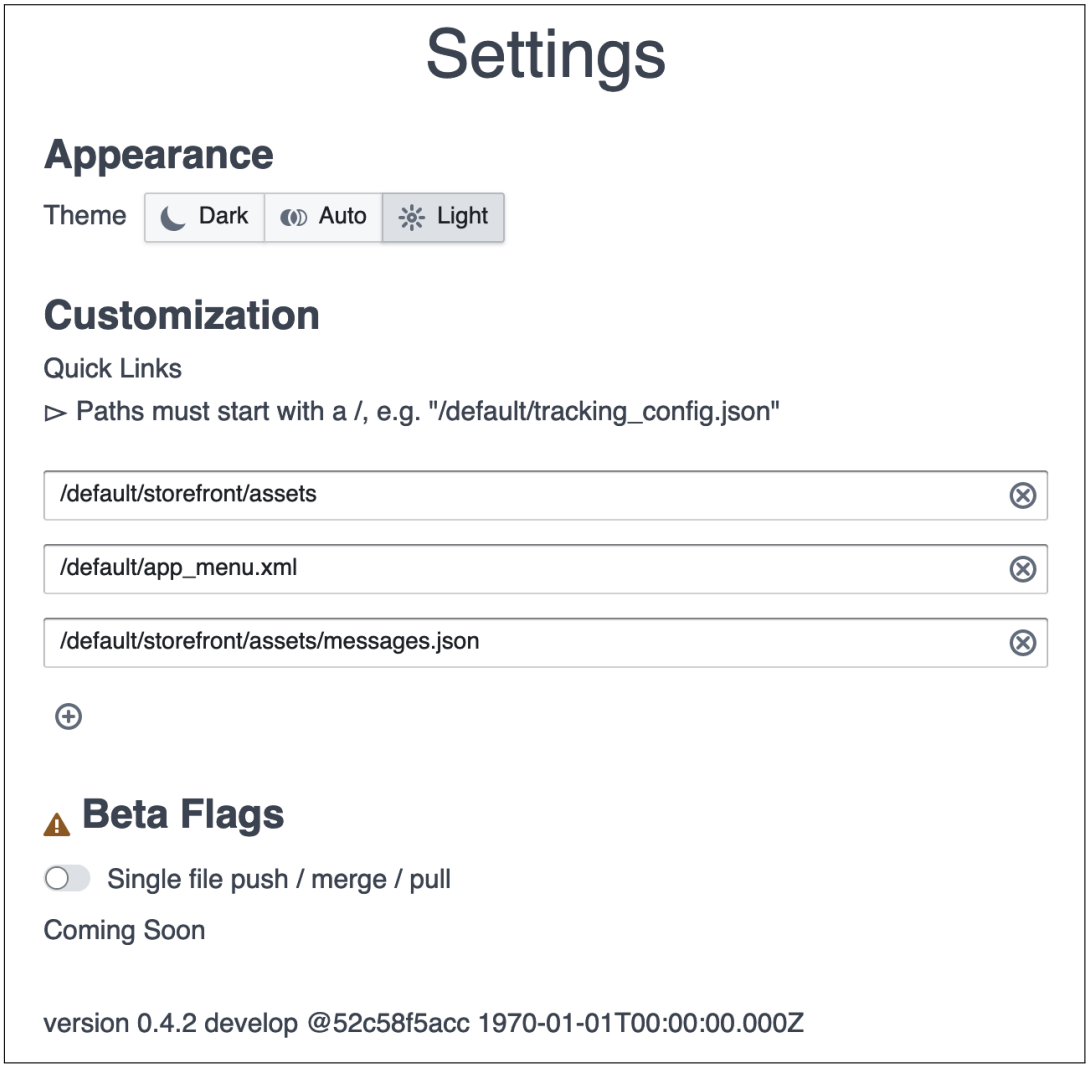

Figure C.3.: Settings view