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 exisiting ecosystem?

Matthias Kind

Matrikelnummer: <IhreMatrikelnummer>

matthias.kind@fu-berlin.de

Betreuer: Florian Berger

Erstgutachterin: Prof. Dr. Claudia Müller-Birn

Zweitgutachter: Prof. Dr. Lutz Prechelt

Berlin, 1.2.2023

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 1.2.2023

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 editor was 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 were 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 to iterate 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

TODO: translate english abstract when finalized <Hier sollten Sie eine kurze, aussagekräftige Zusammenfassung (ca. eine Seite) Ihrer Arbeit geben, welche das Thema der Arbeit, die wichtigsten Inhalte, die Arbeitsergebnisse und die Bewertung der Ergebnisse umfasst.>

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 project, HCI methods need to be adapted to account for technical constraints while still addressing user needs. This approach was taken during the design and development of the UI Editor in this thesis. In addition, user research in general is often neglected due to tight deadlines and limited resources which usually leads to premature releases and unstable software.

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 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 Editor is described in [2, p. 104]. There, the process is divided into the three phases ”Design Thinking”, ”Lean UX” and ”Agile”.

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

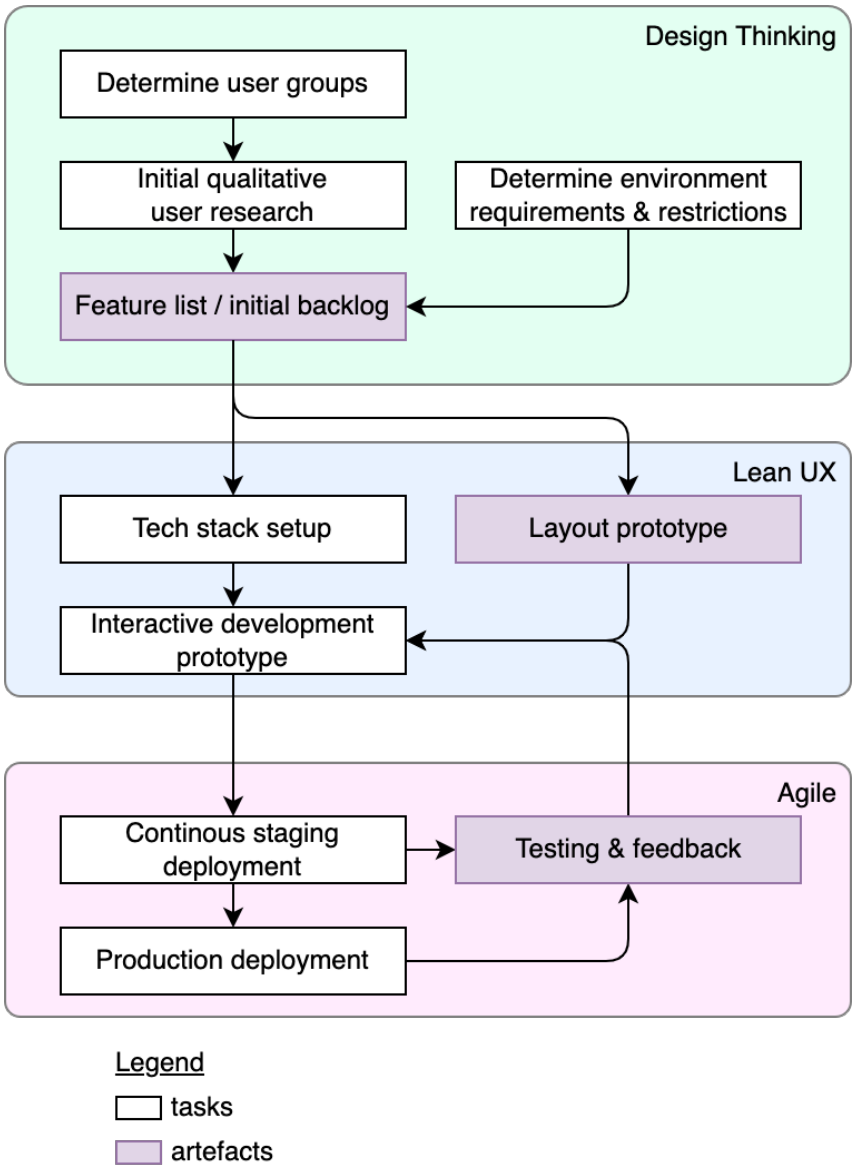

Figure 1.1.: Software design process for the UI Editor.

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 goals" 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). Then, they can import content via multiple ways into the system, or the editors write the content directly inside the tools provided as SaaS.

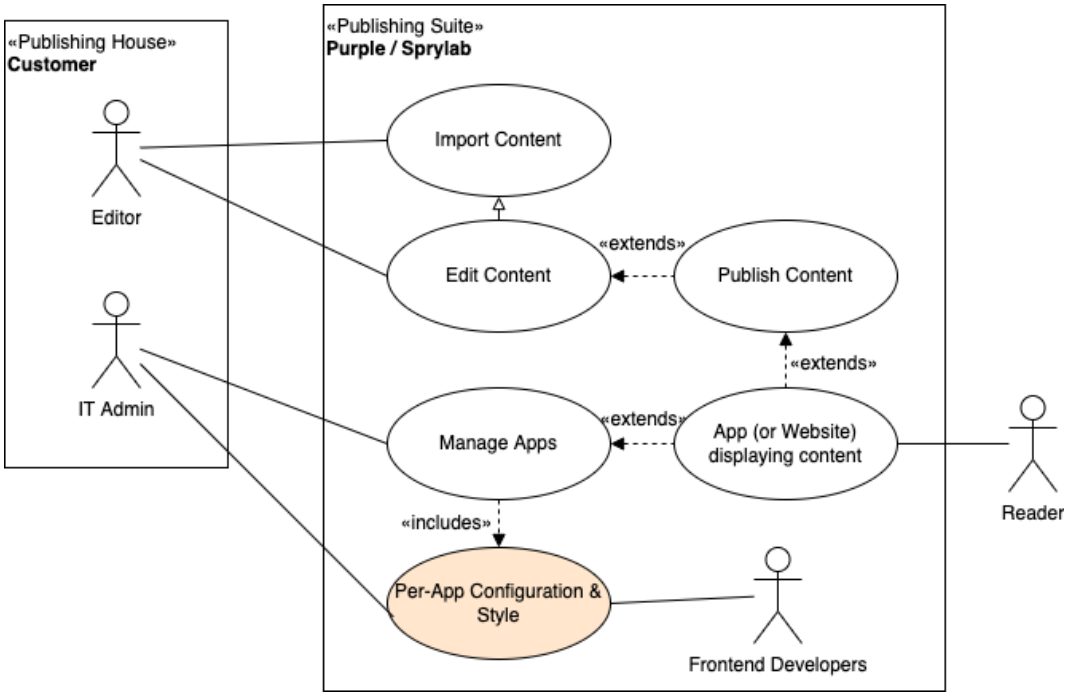

Figure 2.1.: Use case diagram: interactions from publishers, readers and frontend developers with the system TODO: position

The UI Editor fits into the use case "Per App configuration & style" (see 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.

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)

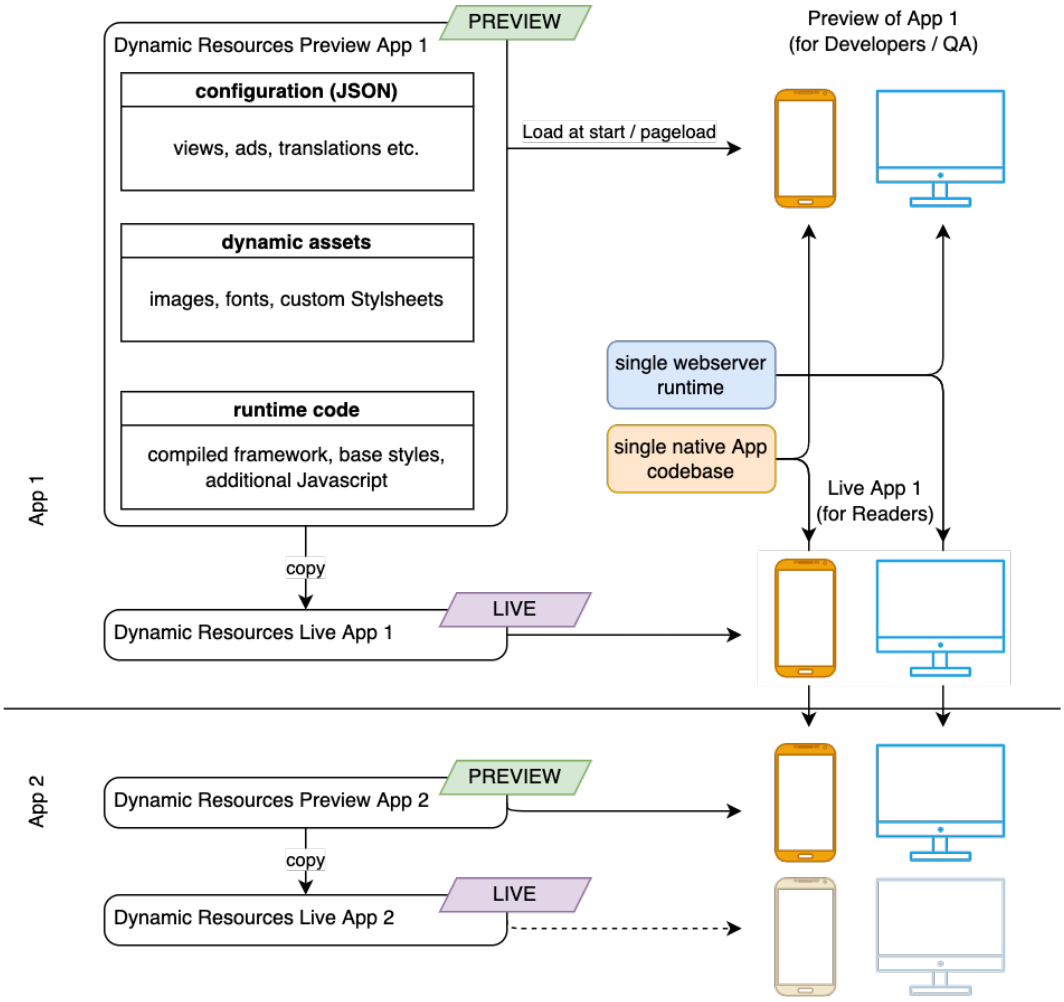

Figure 2.2.: Sprylab preview and live dynamic resources for two imaginary apps

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

sor 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 the interview candidates noted as confusing, not working or slowing down their work.

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

Figure 2.3.: Simplified example of a view configuration showing purchased issues

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.
In public apps, the configs are more advanced with conditionals declaring when components are rendered or filters are applied, event handlers when a user interacts with an element and many more functionalities that are modelled through the JSON Schema.

2.3. Related Work

TODO

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't learn much from only five participants.", "Yeah, but research sounds so academic." [8, p. 4] are only some of the statements that according to Sharon are often heard in software companies in discussions about 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 doesn't 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 the system, network delay and complexity of computation all have some minimal required time, and users understand that fact. More important is to reduce 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 user research process was to identify and categorize them. Then, detailed information is gained through the observations and interviews and finally in section (3.5.2) concrete Personas for the different user groups are built.

It was easy to collect a list of internal and external users for research because Sprylab already had existing users familiar with previous editors and tools from the ecosystem. A larger group of different users or external customers requires more effort to identify and contact suitable users.

With an overview over current and potential users, the next step was grouping them to understand the characteristics and needs of each user group. Having a rough set of groups makes it easier to ensure that the UI Editor is tailored to their specific requirements and 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 - There were users who relied on the tools for most of their work, while others like the external customers accessed the tool a few times a year.

[1] When I refer to "all users", I mean the group of users that are expected to work with the tool. There is an expected technical and domain specific base knowledge that the Editor won't cover in it's initial form

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

Heavy configuration - Mostly internal developers used the tools to build new apps and websites from scratch, making a wide range of modifications and structural changes.

Moderate configuration - Project developers and customer support people copy resources from existing apps and adapt them for new brands, which often includes changing colors and logos, adapting texts or switching authentication flows.

Small changes - External customers often only use the tools to exchange some ads, translations or logos, which affects a small set of files.

Expertise -

Technical - Depending on the area of education and work experience in the web development industry, the expertise about web technologies, languages like CSS and JSON and often also intuition differs between users.

Domain- and Platform Specific - There is a lot of vocabulary, functionality of Purple Experience and other systems as well as permutations of configurations that users learn with time.

3.2. Qualitative user research

The existing user base simplified access to subjects for qualitative user research methods, in combination with the advantage of having most users accessible via company-internal communication channels. Using one or multiple ways of triangulation [6, 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 (using multiple data gathering techniques) as well as triangulation of data (collecting data from different people and different sources) seemed well suited. Moderated observations combined with interviews proofed 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't 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

Semistructured Interview (ca. 15min) ask open and closed questions and discuss observation situation

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 our company 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 as he can 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, and 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

- Change app menu entry ”Newsstand” to ”Home” on all platforms (Web, Android and iOS).
- Change the advertisement banner target on top of the home page to https://google.com.
- Change the text ”Latest Issues” on the home page to ”Read new Issues”.
- Change color of ”Read new Issues” and ”Latest Articles” headers on the home page to the app’s primary color..
- *(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 an URL parameter ”publication” to the id when selected

  Define the reset message as ”All publications”
- *(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 source 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 was not found in the translation files or the color change was applied to more elements than desired.

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 semistructured interview was chosen as the appropriate tool. The interviewer has a list of open and closed questions, through which he 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 an 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.

Here are some examples of the outcomes:

- The way users validate their changes differs widely; some use a preview window in the old editor, some merge changes into the resources and view it on the website or inspect the app's Javascript context, some prefer separate windows, some embedded frames to see code and preview at the same time.
- Rearranging or adapting the layout of the tools / editor to match the current screen and browser window size is important, e.g. hide unecessary panes if not used, make the window the user currently works in larger to see more content at once.
- Editing multiple files at once (in the old editor one could only edit one file at a time, had to close the current and open the next one).
- Colaborative working: Seeing the current state of the resources, e.g. if a live or preview resource is processing, if other users are working in the same app parallel, and having git-like line-by-line diffs of the changes made.

Some of these points and their implementations will be covered in chapter 4 5.

3.3. Quantitative user research

In 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 of questions, and type of questions. While prototype questionnaires were created using Microsoft Forms, 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 every observation and interview, which were additionally recorded, a Word document noting facts about the subject and the session was filled out

by me. 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 list shows a subset of mandatory features that emerged:

App Selection

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

App Editing

- Better support from the file editors, like syntax highlighting and feedback in case of errors.
- Fast load times or the possibility to have multiple files open simultaneously.

App Preview

- See a frame on the side showing the latest changes instantly, load "real" data from the API.
- Ability to pop out into extra window.
- 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.
- Have version control features like "commit messages" and version diffs (this is dependent on external work as the Purple Manager only stores ZIP files, where having git-like version control features is technically not possible).
- See who uploaded new changes and if some changes are being processed.

3.5. Process and visualize the outcomes of the initial user research phase

To gain as much value from the raw research data it is crucial to find some 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. proof valuable once the development process itself started and the state of the tasks must be tracked. But for discovery and prioritization

of features, lists of tickets with different priority levels and different sorting are not suitable for a good overview of effort and benefit each feature brings. 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 cost 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 [6, pp. 403-405]. They can be found in Appendix A and are mentioned later in this thesis.

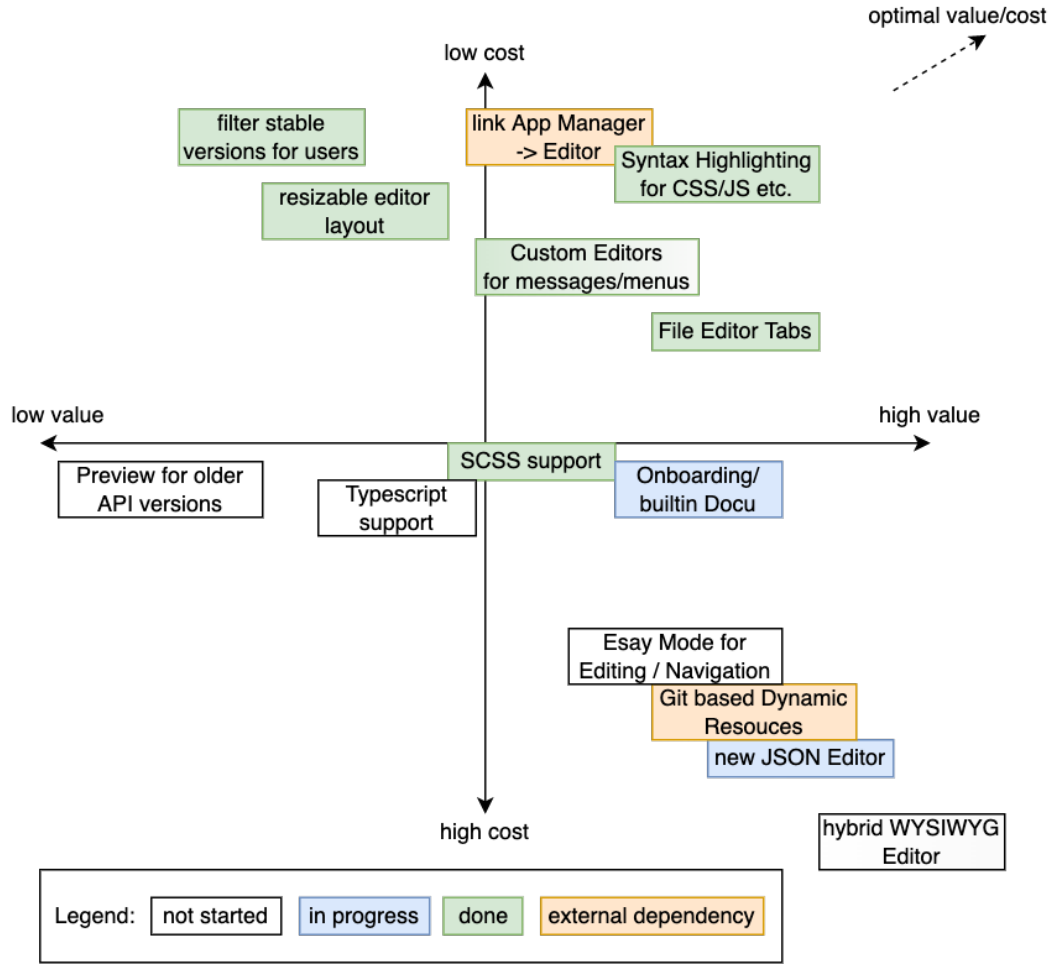

Figure 3.1.: 2x2 Opportunity Matrix during the early phases of development with additional ideas for features

4. Prototyping

After collecting the initial user feedback, minimal digital ”paper” prototypes were created using Figma[1] to gather visualizations of the proposed 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 editor-centric layout is inspired by modern text editors / IDEs like VS Code (`https://code.visualstudio.com/`), which was mentioned as reference during the interviews multiple times. There, the central pane is the editor for the currently open file, while on the sides additional panes for file management, preview and more can be shown. The familarity, especially to developers who are used to IDE layouts, could help new users adopt patterns to work with the UI they use in other tools as well.

The idea for a preview-centric layout was inspired by popular generic website builders like `https://wix.com` or `https://wordpress.com`, where the user can see the page in an interactive mode, move, configure or place elements, and then has on the side additional panels like one with information & options about the currently selected element. There are also framework-agnostic tools like `https://vwo.com/why-us/technology/visual-editor/`, but they either focused more on only editing the style and not the structure of the page or were not compatible with the Experience's framework and data format. Here are some reasons why the editor centric layout was chosen and

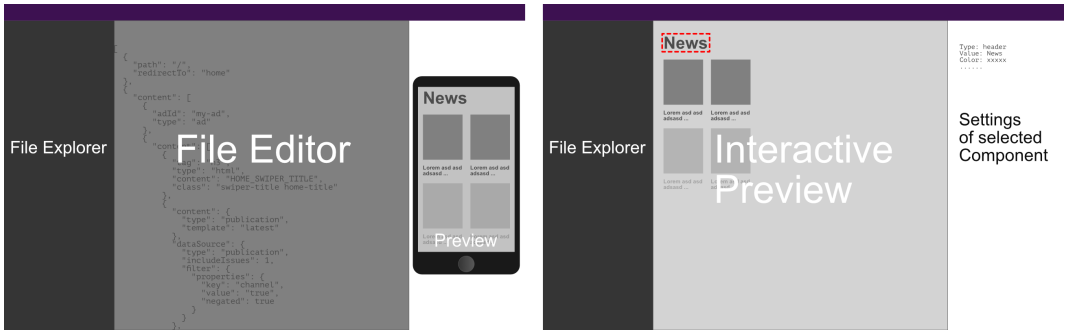

Figure 4.1.: Mockups: Editor centric vs preview centric editor layout

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

not the preview-centric one:

- The configuration structure of the Purple Experience framework was not built with preview-based editing in mind, causing many functionalities to be hidden and invisible to the user, making it difficult to reproduce specific conditions in the editor environment. Thus, editing in a preview-centric mode could lead to more confusion for the editors than speeding up the process. As the configuration schemata are mostly fixed, it was deemed that preview-based editing would not be suitable in this case.

- After evaluating available libraries and examples, we concluded that building a reliable and usable preview-centric editor is more complicated and uncertain to result in a viable product within the limited time frame of this bachelor thesis. For editor-centric UIs, many third-party libraries exist that can be integrated into the UI, such as Microsoft's Monaco Editor (`https://microsoft.github.io/monaco-editor/`) for editing generic web-related files with automatic syntax highlighting and error detection, and JSON Editor for working with JSON configurations with provided schema.

- The user base consists mostly of tech-affine people who are used to layouts of IDEs, and the old tool also had a similar editor-centric layout. 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 [7] and [9, 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

The phase of implementation and deployment followed an agile development process where changes could be deployed easily to get fast feedback from users.

5.1. Architecture

Let's start with an overview about the architecture and high level user flows. We have three software components relevant for a basic interaction with the system: the editor frontend, the editor backend and the "Manager" backend, which is responsible for authentication, app management and providing the dynamic resources in an ZIP format. At the beginning of a user journey, the user visits the root domain (e.g. `https://builder.purplemanager.com` for production apps) and logs himself in. The authentication details will not be covered here as these would go beyond the scope of a bachelor thesis and are not relevant for the UX. The API the Purple Manager exposed and the way dynamic resources get downloaded and merged is one of the basic technical requirements introduced by the surrounding ecosystem.
The UML sequence diagram fig. 5.1 displays a typical interaction of a user with the editor frontend after he selected an app; pulling the latest dynamic resources, editing a file and merging the changes.

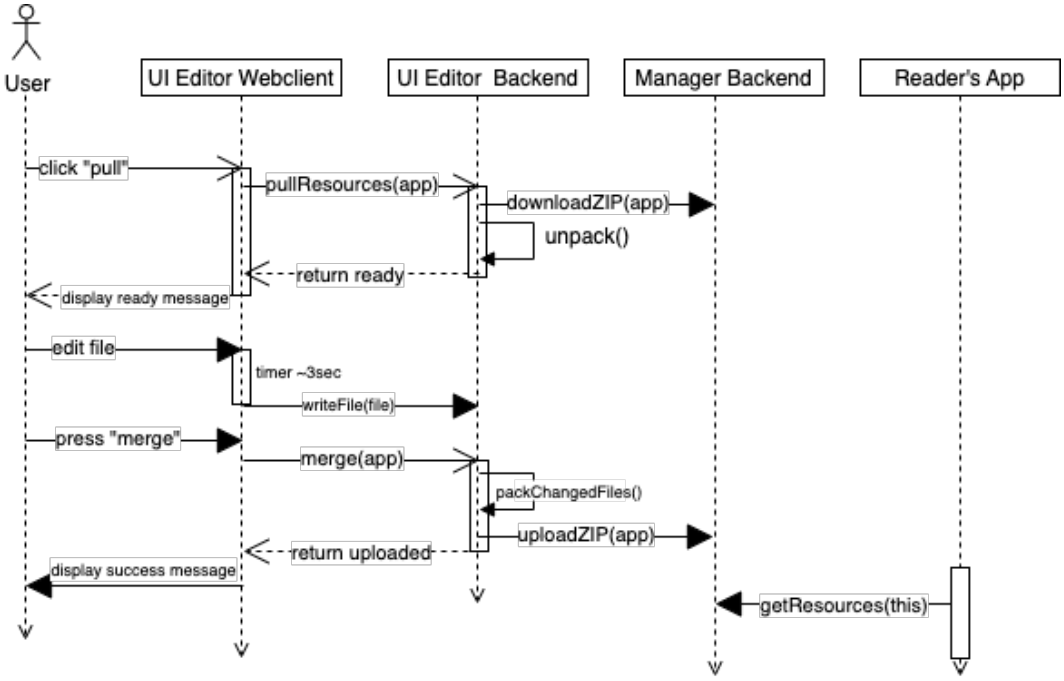

Figure 5.1.: A typical interaction of a user with the UI Editor frontend and it's dependencies

5.2. Software stack

From a company's perspective, it is advantageous to keep the technology stack as narrow 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 application only requirement is to be able to run inside a Docker container 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 many skilled personnel in the company as well as an established ecosystem and easy sharing of code and type definitions between frontend and backend.
The rest of the software stack is fairly common in todays web development industry too:

Frontend

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

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

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 using 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. CI/CD

Continuous Integration and Continuous Delivery, short CI/CD, a core practice of agile software development, enables fast release and deployment cycles which is crucial for agile prototyping and development. A Gitlab CICD 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 for quick reactions to user input and deployment to a staging domain for user feedback in under 10 minutes. Separating staging and production systems also allowed for more confident deployment of quick fixes for validation in a production-like environment without interrupting users.

5.4. Feature examples

In the following section, a selection of features is presented that were implemented during the UI Editor case study. These features are chosen as examples to how the HCI methods and outcomes from the user research phase influenced their design and how they can improve the user's experience with the tool.

5.4.1. File management - open multiple files as tabs

A common workflow consists of editing multiple files at the same time, for example having the view configuration open while adding translations for newly added components. The old editor tool only allowed to open one file at a time and get closed when the user opened another one. This showed up as a big slowdown during the moderated observation. All interview subjects mentioned they have to work on multiple files and that they are annoyed by the workflow, especially when opening large view configs 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 and the user can quickly switch between them or close the ones he doesn't need anymore (see fig. 5.2).

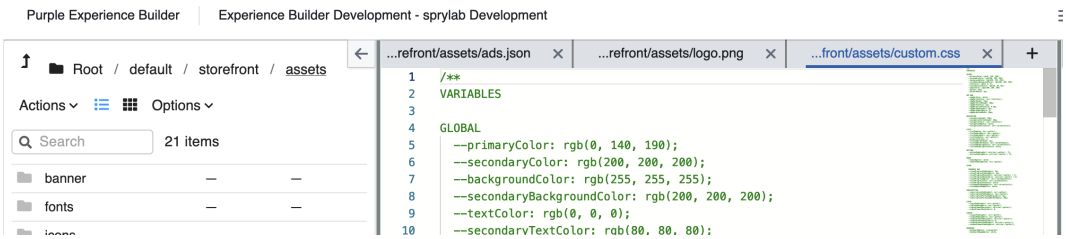

Figure 5.2.: Screenshot of the latest iteration of file tabs

5.4.2. File management - quick links

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

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

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? Currently, the user has a list on the settings view, where he can add or delete new entries, but has to enter the file path manually. A proposal already exists to either add a file picker in the settings or add a bookmark button to the file manager on the primary editing view.

How to present the links? The links must be easily accessible to provide the intended benefit. The solution was to show them when entering the *edit* view and no file was opened yet. In addition, the UI shows if the link points to a file, which gets opened as a new file tab, to a folder which will navigate the file explorer to there, or if the path doesn't exist in the current app.

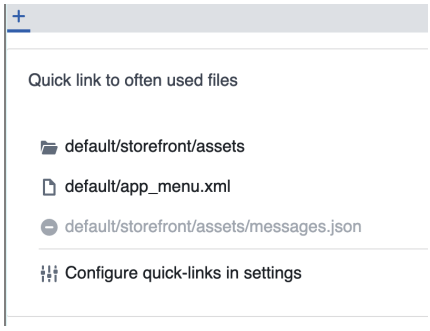

Figure 5.3.: The three types of quick links: Folder, file and not available

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 specialized file editors.

For view configurations for example, we utilize the JSON Editor Library (`https://github.com/json-editor/json-editor`) combined with the generated JSON Schema files, for the translations we have 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. To add a new specialized editor the only two steps are creating a new functional component that implements the *EditorImpl* interface and add that editor in the *EditorRegistry* to get returned for the file paths in question.

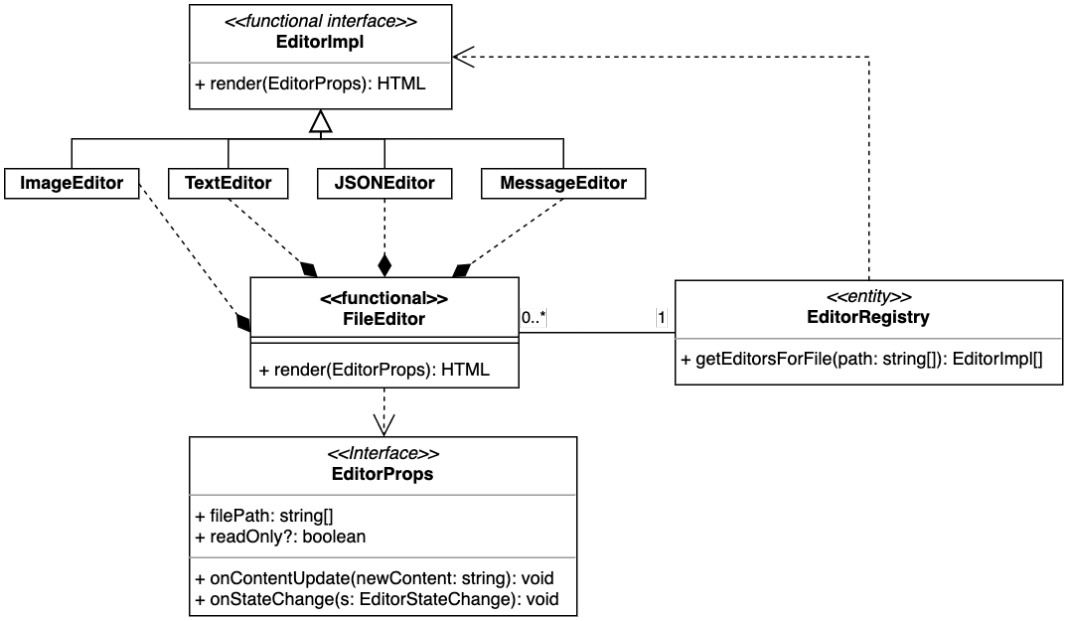

Figure 5.4.: Class diagram showing the editor abstraction in React JS

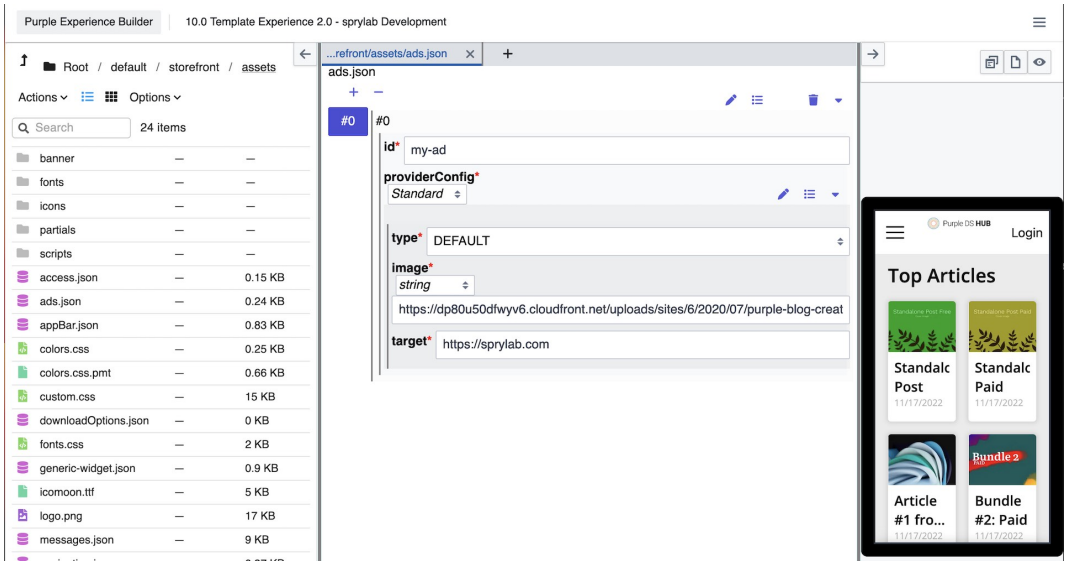

Figure 5.5.: Screenshot: JSON Editor when configuring ads

5.5. Automated Testing

As already elaborated, as fast CI/CD piepline plays a crucial role for agile prototyping and development. Automated tests in turn contribute to the confidence of developers to deploy more frequently and can reduce load from the QA team to test standard workflows every time a new release is planned. For the Experience Builder I stuck to two of the most common testing levels: unit tests and End-to-End tests (also known as E2E or System tests).

Unit tests are mostly used to test one ”component” in an sandboxed environment. For the server, this meant testing Services and classes or even finer grained; single functions. On the client, we distinguished between UI testing of single react components, and business logic code that is encapsulated in classes or Javascript modules.

They also are helpful during development to test software patterns before dogin large refactorings and to do test-driven development (TDD), where the specifications and constraints can be laid out as code with invariants, pre- and postconditions, and then the implementation is done while continously running the tests again until they don’t fail anymore.
Espeically for TDD, but also for the CI/CD Pipelines, the speed of the tests is important. If a single test run takes multiple minutes, the developer is blocked during that time and can’t progress on the task, but the duration is also not long enough to start working on another task in the meantime. That’s why I tried to integrate a fast test runtime comatible with UI- / browser testing as well as node runtimes for the server code. After evaluating diffrent commonly used frameworks, I settled for Vitest (`https://vitest.dev/`), which fullfills all the requirements, runs tests in parallel, thus reducing the time between

saving the change and seeing the test result to often less than a second, and has easy integration mechanisms into our build tools.

While unit tests are a good way to verify encapsulated behaviours, when many components interact with each other, new errors can emerge often as it is quite unrealistic to have all intenral and extenral APIs behaving exactly as expected for every input, and Web based UI applications have unmanagable number of factors that influence the behaviour of UI, network and timings.
E2E tests are supposed to cover a typical user interaction with the service to vaildate the interaction between business logic components, UI and the user itself. I started with using a custom setup of a headless browser[1] (`https://pptr.dev/`) in combination with vitest, but writing and especially debugging the tests prooved slow and error prone.
At an internal training day some colleagues introduced a new E2E test framework called Playwright (`https://playwright.dev/`), which allows recording a test case in a "normal" browser window, it then generates the base code for the test automatically and only needs to be adapted in a few places. After looking through some examples and seeing how it can get integrated into our pipelines, I started porting the exisitng tests to playwright, and after a few hours the tests run on the new framework, now with much better debugging tools and the ability to add new tests much faster.
This can be an example for others, that investing time to investigate new tools and port code to them if they bring value, can improve developer expeirence and thus also 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 showed me a SaaS called `https://squeaky.ai/`, which is a cookieless solution to track users on this page. While it can't provide the same level of demographic information a cookie based solution can, it still provides the most important data like usage statistics, user interactions, occured Javascript errors and can even show heatmaps per page to visualize which UI elements the user interacts with the most.

These heatmaps for example can show if a new UI feature is used by users and how they interact with the page in general. In case of fig. 5.6, it can be

[1] Headless browsers are browser instances that don't render the actual content to a user's screen, but run as a CLI application and still execute all Javascript, CSS and HTML.

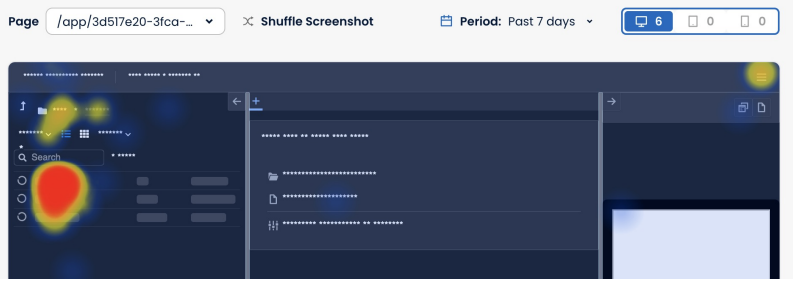

Figure 5.6.: Screenshot: Squeaky.ai heatmap of an app's edit view

seen that the users mostly interacted with the file explorer and only did a few clicks in the editor panel.

To analyze the usage growth, fig. 5.7 indicates how the number of page views per week increased steadily since the internal beta was started (except for the dip during the Christmas holidays). With that graph it is save to state that the time-bound goal from the SMART goals (3) was successfully achieved.

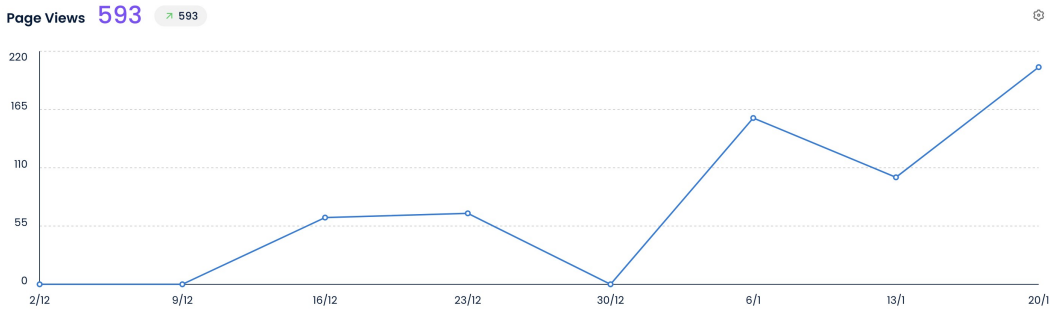

Figure 5.7.: Screenshot: Squeaky.ai page views since December 2022

5.7. User Testing, Feedback, Beta and Monitoring

After a first stable release was done and I verified that all basic functional requirements are met and the editor doesn't destroy the dynamic resources, I spoke with four people from our company, three of which were also interview subjects, to try to use the editor and give verbal feedback.

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

TODO überarbeiten

- deployments to staging - small test group at start - internal beta - problem: people didn't want to adapt new platform?? - beta flags

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.
For the communication, we utilized a Microsoft Teams Channel, a group chat where all invited persons (in case of the internal bet company wide) can write with each other and create posts. This was used for notifications about new deployments and if in the aftermath someone found a bug that could affect multiple people.

Even though this adds a bit overhead, it prooved easier than having everyone write bug tickets directly, as user often don't know the actual reason for the problem which leads to unclear descriptions, wrong tags and more. Rather, a developer with knowledge about the system looked at the reports and created a new ticket if the problem was new, else 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.

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 users needs can be combined in this context. Agile development and Lean UX enabled iterating quickly with new ideas and achieving fast deployments to a staging system. A growing group of users at Sprylab already using the software in production and a majority of positive feedback show that the chosen methods and from their outcome derived features were the right approach to enhance the user experience while complying with technical requirements. 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.

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

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

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

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

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

List of Figures

Glossary

Agile TODO. 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]. 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 TODO. 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 don't have legacy code, while brownfield projects are about upgrading or redeveloping software in an existing environment [5]. 1

Lean UX TODO. 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. 13, 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, 4, 7, 18

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 TODO. 7

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 <TODO> encounter problems
- John works with the editor multiple times a week

B.2. Steffi - Project Developer

B.2.1. Background and Skills

Steffi (23) studies media informatics and works as a working student at Sprylab since a year. This is her first job in the industry and she is learning 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.

B.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

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

C.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.

C.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. 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 Mod. Obs.

- 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

Appendix B.

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 Mod. Obs.
- 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 doesn't 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 don't get deleted from ZIP file
- updates not transparent what changed

Observations during Mod. Obs.

- Heavily uses side preview (ref to customer success member who doesn't 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 Mod. Obs.:

- 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