

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: <IhreMatrikelnummer>

matthias.kind@fu-berlin.de

Betreuer: Florian Berger

Erstgutachterin: Prof. Dr. Claudia Müller-Birn

Zweitgutachter: Prof. Dr. Lutz Prechelt

Berlin, TODO

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 December 21, 2022

Matthias Kind

Abstract

In recent years, the shift from print to digital publishing has increased the need for tools that allow publishers to quickly build and configure apps and websites. While progress has been made in this area with the development of general purpose website builders and headless content management systems, the internal tools and software used by publishing administrators also need to evolve and improve.

The goal of this bachelor thesis is to conceptualize, plan, and implement an UI editor for apps and websites used by publishers, particularly in Germany and the UK. The editor will be built using a web framework called "Purple Experience," which is used to deliver apps and websites generated from the same configuration and assets to end users. This service is closely linked to other existing software systems that are used to edit content, manage apps, and deliver content.

This "brownfield" software project presents both challenges and opportunities. The flexibility of the editor is restricted by existing workflows and software that cannot be changed, but it also has a diverse group of users who have varying levels of experience with these software products. These users, who include internal framework developers, customer support, and project developers, as well as external people at publishing houses, provide valuable insights into their current workflows and how they believe the editor can improve their productivity and enjoyment.

To gain these insights, the use of existing software was evaluated, and multiple user research methods, including moderated observations and interviews, were applied. The outcome of this research will be useful as guidance for future software development projects for internal tools at companies, or in environments where constraints exist but a user base is already in place to provide valuable input and feedback.

The outcome should be usable as guidance for future software development projects for internal tools at companies or in environments where constraints may exist, but also an approachable, already present user base can give valuable input and feedback.

Based on the evaluations of the user research phase, I built an interactive prototype using modern web technologies like react, express.js and Typescript. This was deployed using continuous integration to a controlled group of test users. This allowed to get quick feedback and iterate fast, until the tool can be made available to a broader audience.

TODO: the outcomes of the thesis consist of a working software product that is actively used by early adopters, as well

Zusammenfassung

<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	3
1.1	Topic and context	3
1.2	Goals of this work	3
1.3	Procedure for the research and implementation	4
2	Kapitel	5
3	Theoretical background	7
3.1	HCI	7
3.2	Project specific background	7
4	Related work	9
5	User research and analysis	11
5.1	Identifying and categorizing users and user groups	11
5.2	Qualitative user research	12
5.2.1	Interview	12
5.2.2	Moderated observation	12
5.3	Quantitative user research	13
5.4	Process and visualize the outcomes of the initial user research phase	13
5.4.1	2x2 Opportunity Matrix	13
6	Prototyping	15
7	Implementation and deployment	17
7.1	Software stack	17
7.2	CI/CD	18
7.3	Automated Testing	19
7.4	Scalability	20
7.5	User Testing, Feedback, Beta and Monitoring	20
7.6	Communication and Documentation	20
8	Conclusion and outlook	21
	Literatur	22
	Appendix	25
8.1	Erster Teil Appendix	25
8.2	Zweiter Teil Appendix	25

List of Figures

- 5.1 2x2 Opportunity Matrix during the early phases of development 13

List of Tables

Vorwort

Allgemeine Hinweise zur Erstellung einer Abschlussarbeit

- Beachten Sie, dass diese Vorlage für einen zweiseitigen Ausdruck angelegt wurde.
- Über die Papierqualität können Sie entscheiden, aber wir empfehlen aber Seiten mit wichtigen, farbigen Grafiken auch in Farbe auszudrucken und dabei ein höherwertiges Papier zu verwenden.
- Bitte stimmen Sie mit dem Betreuer Ihrer Arbeit auch den Zweitgutachter ab. Die Anfrage des Zweitgutachters erfolgt von Ihnen. Es ist an dieser Stelle sinnvoll, die Anfrage mit einer kurzen Zusammenfassung der Arbeit zu stellen.
- Bitte beachten Sie, dass Sie Ihre Abschlussarbeit mit einer Klebebindung versehen, eine Ringbindung ist nicht erwünscht.

1 Introduction

1.1 Topic and context

In the evergrowing world of software companies, many once startups are now in the situation where they maintain a large software ecosystem and have complicated dependencies of other services or users, but still want to improve their systems by developing new components and tools.

This poses the challenge of improving the software from aspects like user experience, scalability and maintainability while being restricted by the ecosystem. Greenfield development, as it is taught for the majority of books, can't be applied freely without breaking existing features or behaviours. Thus, applying HCI methods for user research and user experience-focused design might need to be approached in a different way than during greenfield development. Also, the common problem of tight deadlines and limited resources allocated by managers tend to lead to premature releases and unstable software. Instead of developing software to maximize the three HCI factors **See HCD principles / factors src, anme the tree?** for the actual users, often ideas from individual stakeholders like the executive floor are realized without adding real value.

On the other hand, having an existing user base which works with existing tools is a great fundament to evaluate what "real users" need. So HCI methods applied to them can yield more helpful and focused results. **rephrase**

Many of the resources or literature about HCI seem to assume a mostly free degree while developing new tools, and also assume a wide user base with diverse demographic features **is this valid english?**.

List literature srces that dont speak about brownfield development? Find a contra example that does cope with brownfield

1.2 Goals of this work

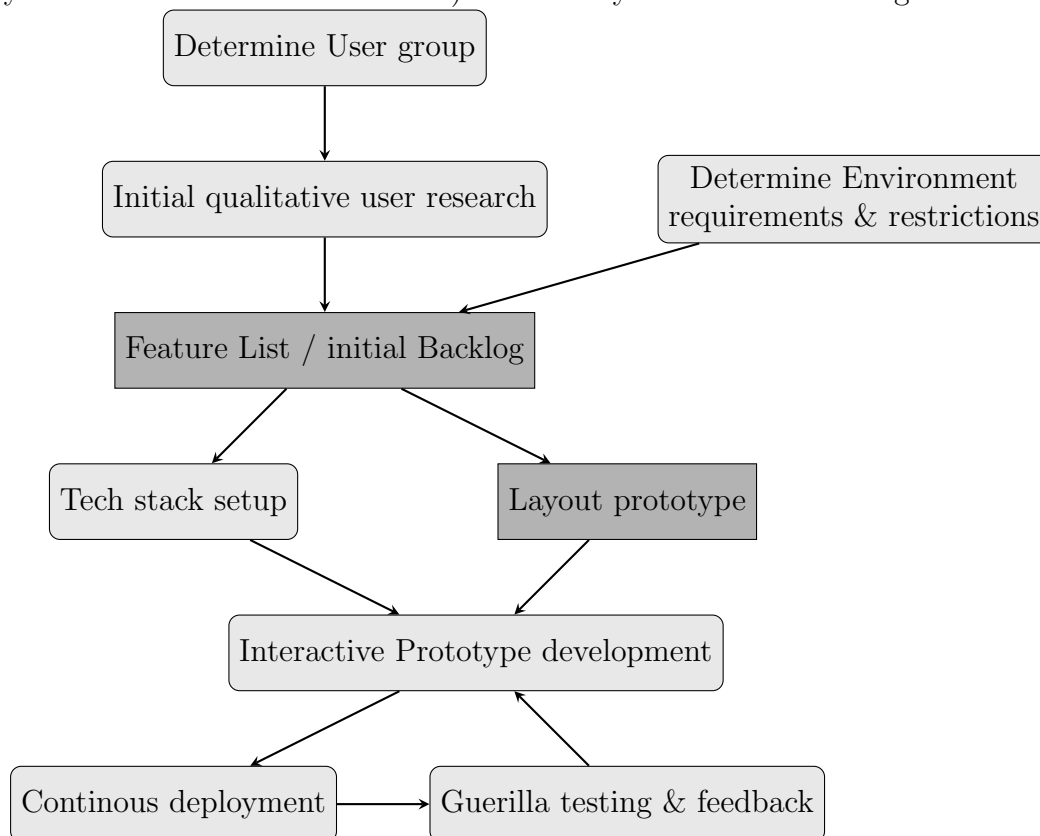
- Was sind die mit dieser Arbeit verfolgten Ziele? Welches Problem soll gelöst werden?
- Eine Beschreibung der ersten Ideen, der vorgeschlagene Ansatz und die aktuell erreichten Resultate
- Eine Beschreibung, welchen Beitrag die Arbeit leistet, um das vorgestellte Problem zu lösen
- Eine Diskussion, wie die vorgeschlagene Lösung sich von bestehenden unterscheidet, was ist neu oder besser?

1.3. Procedure for the research and implementation

The goal of this thesis is to implement a new software product, an UI Editor for Apps & Websites of digital publishing customers, which is embedded in an existing software ecosystem. During that process, different HCI methods get applied and

1.3 Procedure for the research and implementation

While writing the software and thesis, I followed the an software design process described in [2, p. 104]. There, the process is divided into a "Idea" phase using design thinking, lean ux for first prototypes and then agile development (in my case a relaxed SCRUM version). Abstractly it looks as following:



2 Kapitel

- Abhängig vom Ziel der Arbeit und dem verwendeten Forschungsdesign unterscheidet sich dieser Hauptteil der Arbeit erheblich.

- Eine sehr allgemeine Struktur ist die folgende:

- Hintergrund der Arbeit (Theoretische Einordnung der Arbeit)
 - * Hier sollte enthalten sein, welche Anwendungen in diesem Bereich bereits existieren und warum bei diesen ein Defizit besteht.
 - * Falls genutzt, sollten hier die entsprechenden Algorithmen erläutert werden.
 - * Es sollten die Ziele der Anwendungsentwicklung, d.h. die Anforderungen herausgearbeitet werden. Dabei sollte die bestehende Literatur geeignet integriert werden.
- Umsetzung (Praktischer Anteil der Arbeit)
 - * Zunächst sollte die Softwarearchitektur und die genutzten Anwendungen, APIs etc. erläutert werden. Ebenfalls gehört dazu das Datenbankschema.
 - * Es sollten die zentralen Elemente der Software (abhängig von der Aufgabenstellung) beschrieben werden, wie implementierte Algorithmen oder das Oberflächendesign.
 - * Zentraler Quellcode sollte entsprechend aufgelistet werden:

```
1      public class Main {  
2          public static void main(String[] args) {  
3              System.out.println("Hello World!");  
4          }  
5      }
```

- Evaluation (zumeist nur für Masterarbeiten relevant)
 - * Jede Software muss auch getestet werden. Dieses Tests werden entweder mit einem vorgegebenen Datensatz erfolgen oder aber die Evaluation erfolgt auf Basis von Experimenten. In diesem Kapitel sollte daher entweder der genutzte Datensatz oder der experimentelle Aufbau beschrieben werden.
- Ergebnis und Diskussion

2. Kapitel

- * Die Ergebnisse der Anwendung werden in diesem Kapitel vorgestellt und anschließend diskutiert. Wenn möglich sollte die Ergebnisse in Relation zu bestehenden Arbeiten in dem Bereich erörtert werden.

3 Theoretical background

This thesis can be divided into two mayor areas of theoretical background. Human-Computer-Interaction (HCI) and project specific background.

3.1 HCI

Let me start by explaining the HCI aspects and why this thesis approaches the area from an not so common standpoint.

Many HCI books (e.g. [4] or [2]) implicitly assume Greenfield development, which "[...] is in its most distinct form when a new product is created from scratch – a new product or product platform, based on new technology, using new methodology and implemented by people who are new to it all." [1].

While they mention *Environmental requirements* as part of the requirement discovery, this usually is more focused on what abilities the users have (TODO), and not on the restrictions imposed by older software companies, especially those colloquially called "enterprise".

There, besides the omnipresent time and capacity restrictions and sometimes not that constructive inputs from stake holders, often development must be used. In this approach, new capabilities are added to the software, while relying on the existng technology and knowledge. [1].

Therefore, some of the HCI methods must get a bit adapted to fit into this system.

Also, the UI Editor developed for this thesis has to be counted as internal tooling, so it is reserved for a quite specific user base. Not every person on the internet should edit these apps and dynamic resources, as they require understanding of web technologies and the digital publishing nomenclature.

3.2 Project specific background

To understand the usecase and value of the UI Editor, we first have to declare the fundamentals of the environment the editor will be embedded in.

The publishing houses resp. 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, or the editors write the content directly inside the tools provided as an Software-As-A-Service (SaaS).

The apps are running an Meta framework build ontop of Angular, which is completely configurable via JSON files describing the routing, rendering of

3.2. Project specific background

different components, connecting data sources (an API abstraction) with those components, loading assets like images and ads, and styling the whole page with CSS.

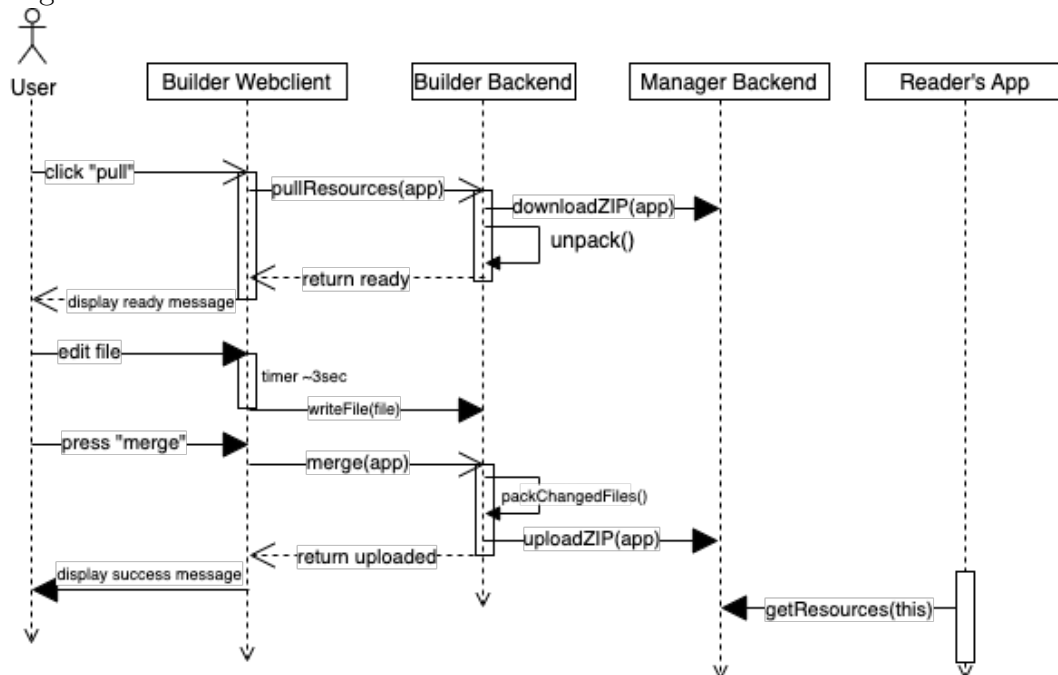
These configs and assets are stored on a file system called *dynamic resources*.

Dynamic resources are individually managed and loaded for every app. This way, on mobile phones the endusers 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 Domain or URL, and renders the website using that config. This way, all customers can share the same server instance(s), or at least don't require extra build artefacts per app.

If you have worked with larger JSON files before, you may recall that they get convoluted quite fast. Also, manually downloading ZIP files, unpacking them, changing assets and config files, packing them and uploading and hoping one didn't introduce a typo anywhere is an inefficient and at times quite dangerous workflow.

At Sprylab, there exists a tool called "Storefront Editor", which is used as the foundation for this new editor. In the section about (TODO: LINK!!!) User Research, I will outline the positive aspects and approaches which I reused for the new editor, as well as show the missing features and features the interview candidates noted as confusing, not working or slowing down their work.

The following UML sequence diagram displays a typical interaction of a user with the editor; pulling the current version, editing a file and merging the changes.



4 Related work

TODO

4. Related work

5 User research and analysis

To avoid the common problem in software development where products are built based on the ideas of individual stakeholders who may not even use the product, instead of relying on meaningful user input, I applied various methods of user research commonly used in Human-Computer Interaction (HCI) and evaluated their effectiveness in the context described earlier.

A starting point for qualitative user research is to define the goals through the help of the SMART Criteria <TODO cite>.

For these criterias, I defined the overall goal as following:

- **specific** - improve the workflow of users modifying dynamic resources
- **measurable** - interviews after testing period, concerning working speed and confidence when editing resources, user tracking
- **assignable** - research implementation will mostly be conducted by me, with input from CTO & product owner, connection to external users through customer service team
- **realistic** - new software platform which reacts quicker, provides more safety regarding errors and is scalable and extensible in the future. Limiting factors are time (as I only have three months for the first phase, including writinh this thesis)
- **time-related** - the new software should have at least the same feature set and be usable by company-internal users until the end of 2022

5.1 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. Therefore, the first step in the user research process was to identify and categorize the different users and user groups who will be using the editor. This included both internal and external users, as well as users with different levels of experience and expertise. By understanding the characteristics and needs of each user group, we can ensure that the UI editor is tailored to their specific requirements and can be used effectively by all users.

As explained earlier, because we had a preceeding, less powerfull tool to edit the resources, which was used mostly by company-internal developers and managers, but also available to some external customers. As a first step, I collected

5.2. Qualitative user research

a list of mail addresses that accessed the tool by looking at the logs, and wrote a mail to our developers who would be interested in working with me for both design phase and later as alpha and -beta testers.

Then, I derived the following common factors from the users, most of which I personally know, which made the communication and categorization a lot easier.

- **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.

- **common tasks** I categorized the common tasks into three groups:

Heavy configuration Mostly internal devs used the tools to build new apps and websites from scratch (or derived from existing apps), making many modifications, from structural changes to the separate views, menus, data sources and more, over styling and translating messages to different languages.

Moderate configuration Project devs 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** <kann man bei schlechter software gut erkennen, leute mit viel erfahrung checken sachen, aber ist für neue nicht intuitiv>

5.2 Qualitative user research

- mix interview / moderated observation
- experiences, outcomes, what went good and bad

5.2.1 Interview

TODO interview preparation and conduction

5.2.2 Moderated observation

TODO prep and conducting

5.3 Quantitative user research

Not used survey / questionnaire -> lay down reasons why not necessary in that situation

Tracking of user behaviours

- on site using G analytics & (the other GDPR compliant tracking tool name??) - using server logs to understand usage patterns

5.4 Process and visualize the outcomes of the initial user research phase

5.4.1 2x2 Opportunity Matrix

This two-dimensional visualization of a set of proposed features 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.

The matrix I used is a slightly modified adaption from [2, p. 181], replaced the term "idea originality" on the x-axis with "Value".

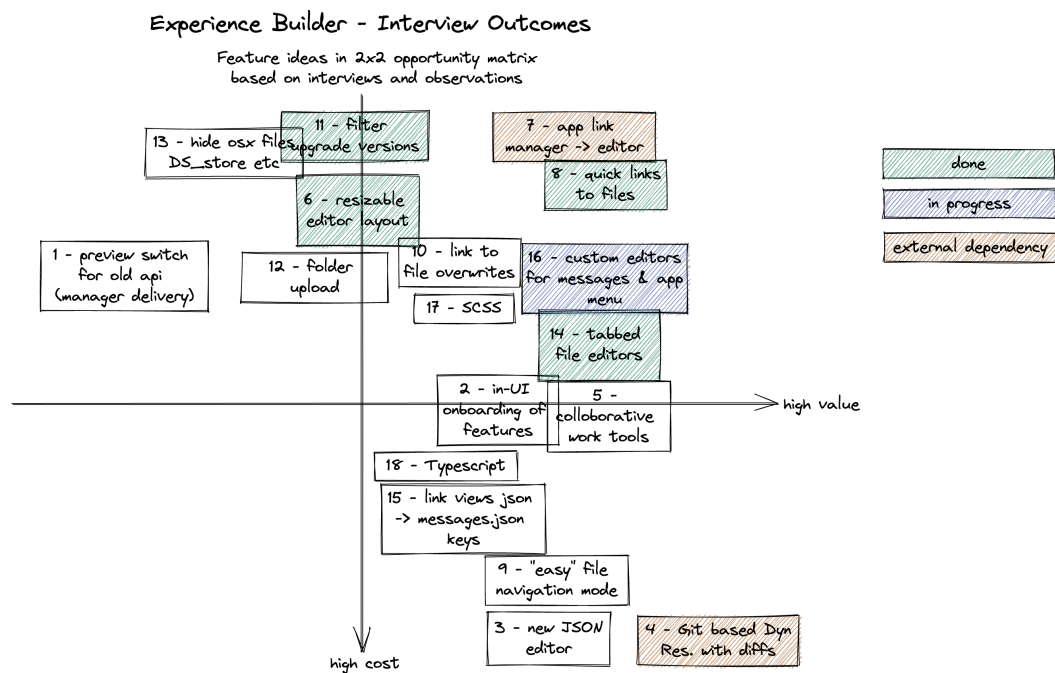


Figure 5.1: 2x2 Opportunity Matrix during the early phases of development

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

6 Prototyping

After collecting the initial user feedback, I started drawing minimal digital "paper" prototypes using Figma to gather visualizations of the proposed UI layouts. Two ideas emerged from the interviews: a (file-)editor-centric layout and a 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 in the sides additional panes for file management, preview and more can be shown. The familiarity 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 web-site builders like <https://wix.com> or <https://wordpress.com>, where the user can see the page in an interactive mode, move, rename or place elements, and then has on the side additional panels like one with information & options about the currently selected one.

Ultimately our choice fell to the editor centric layout, the following are some of the reasons for it over the preview-centric one:

- The configuration structure of the Experience framework was not built with preview-based editing in mind. A lot of functionality is hidden inside the components and invisible for the user, often components only appear under specific conditions that are not easily reproducible in the editor environment. Thus, editing in a preview-centric mode could in many situations lead to more confusion by the editors than speed the process up.
- After evaluation of some available libraries and examples, we concluded that building a reliable and usable preview-centric editor is more complicated, and even without the time restriction of my bachelor thesis, I proposed to not go this way, as it was unclear if it even could result in a viable product in reasonable time. For editor-centric UIs, many thirdparty libraries exist, that can be integrated into the UI. Some relevant are Monaco Editor (the VS Code Open source text editor part) for editing generic web related files like CSS and JS with automatic syntax highlighting and error detection, and an JSON Editor for work with json configs where we could provide a schema.
- The userbase consists mostly of tech-affine people who are used to layouts

6. Prototyping

of IDEs, and the old tool also had a similar editor centric layout. As Jakob's Law of the Internet User Experience states (cf. [5] and [6, p. 2]), the user's understanding of a website is directly tied to his/hers mental model of that system. Introducing a unconventional workflow comes with the danger that the user gets confused, makes mistakes or in the worst case doesn't like working with the tool anymore.

7 Implementation and deployment

The phase of implementation and deployment followed an agile development process, where I could deploy changes easily to get fast feedback from users. I structured the chapter into the following parts:

7.1	Software stack	17
7.2	CI/CD	18
7.3	Automated Testing	19
7.4	Scalability	20
7.5	User Testing, Feedback, Beta and Monitoring	20
7.6	Communication and Documentation	20

7.1 Software stack

A common point appearing in brownfield software projects, or generally in larger companies and software ecosystems, is the limitation in technologies that should be used for a new project. Else, one may chose the latest and greatest language or framework for that usecase, but maintainability and availability of persons to review and collaborate are important as well. To reduce overhead of learning new languages, tech stacks and keeping the infrastrucuture manageable, often a small set of languages and frameworks is provided by the devops- or infrastructure team. In my case, the most important point was the availability of additional persons with knowledge. The constraints from devops side were not that tight, since the code gets deployed in Kubernetes, a containerized environment anyways, so as long as it can run on a linux VM, it could get deployed.

For this project I decided to use Typescript (<https://www.typescriptlang.org>) on both back- and frontend. The advantages are wide availability of persons in the company who also work with it, a mature ecosystem, and shared type declarations between server and client, reducing the risk of working with incompatabile data types by accident.

The rest of the stack is fairly common in the web developemnt industry too, the frontend uses React JS as a rendering and reactivity framework, with additional libraries for state management, UI Components and API query management on top.

For the backend, I used Node as a Javascript runtime, combined with the most used HTTP server framework for Node, Express JS [3]. On top of express, a framework called TSOA (<https://github.com/lukeautry/tsoa>) is used to improve the developer expeirence. It provides a class based architecture simi-

7.2. CI/CD

lar to the well known Spring platform for java, including dependency injection, type safety and automatic API documentation.

Frontend, backend and shared libraries are stored in a monorepo to speed up development and deployments and make refactorings more consistent.

7.2 CI/CD

CI/CD, short for Continuous Integration and Continuous Deployment, is a core practice of agile software development and provides a log of value to HCI projects. The first part, Continuous Integration, means "[...] developers add to a shared repository frequently that integrates their code." [2, p. 81] The integration consists of automatic builds and tests (cf. 7.3) to improve quality of the code and confidence of the developers to publish changes more frequently.

As Sprylab uses and private gitlab, for the Expeirence Builder I set up a Gitlab CICD Pipeline. The stages may change when additional build steps or tests are added, but currently it consists of the following stages: *Build*, *Package* and *Deploy*.

Deploy is only active on the develop and master Branches and contains the code to upload the docker image to the company's staging and production clusters and update the kubernetes deployment to run the latest build. The other two stages are run for every Commit of an Merge request. This has the benefit that as soon as a developer pushes his code to a branch that has a Merge Request open, he and all others can see if the latest commit could be merged safely or if there is more work to do. The *Build* stage installs the dependencies, builds all packages in the monorepo and executes unit- and e2e-tests afterwards. If any of these steps fail, the Merge request can't be merged until this issue is resolved.

Of course the benefit of CI/CD depends on the setup of the build chain, amount, coverage and quality of tests and other factors. To further improve code quality, we integrated Sonarqube into the repository on gitlab (<https://www.sonarqube.org>), which is a service that automatically scans the code and reports code quality, security issues and technical debt added in a commit.

Having a fast and reliable CI/CD process early on during prototyping and development proved very valuable, as I could quickly react to user's input on new features or bugs, implement and deploy them quickly on a staging system and get feedback on the new behaviour in less than 15 minutes. Through the separation of staging and production system, I could deploy quick fixes with more confidence even when beta testers were working on the production system, as I could verify that my changes worked in a production-like environment without interrupting users.

7.3 Automated Testing

As already elaborated, the quality and quantity of automated tests plays a huge role when using CI/CD, as it drastically reduces the time required by Quality Assurance testers to go through all the edge cases on every change. 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 doing 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 performed while continuously running the tests again until they don't fail anymore.

Especially 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 compatible with UI- / browser testing as well as node runtimes for the server code. After evaluating different commonly used frameworks, I settled for Vitest (<https://vitest.dev/>), which fulfills 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 internal and external APIs behaving exactly as expected for every input, and Web based UI applications have an unmanageable 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 validate the interaction between business logic components, UI and the user itself. I started with using a custom setup of a headless browser¹ (<https://pptr.dev/>) in combination with vitest, but writing and especially debugging the tests proved 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

¹Headless 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.

7.6. Communication and Documentation

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 existing 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 experience and thus also speed and confidence.

7.4 Scalability

- REDIS pubsub + observables - code example

7.5 User Testing, Feedback, Beta and Monitoring

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

7.6 Communication and Documentation

- teams - jira - missing documentation

8 Conclusion and outlook

- Die Zusammenfassung sollte das Ziel der Arbeit und die zentralen Ergebnisse beschreiben. Des Weiteren sollten auch bestehende Probleme bei der Arbeit aufgezählt werden und Vorschläge herausgearbeitet werden, die helfen, diese Probleme zukünftig zu umgehen. Mögliche Erweiterungen für die umgesetzte Anwendung sollten hier auch beschrieben werden.

8. Conclusion and outlook

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] Vano Devium. *Top Node.js Web Frameworks*. URL: <https://github.com/vanodevium/node-framework-stars> (visited on 12/13/2022).
- [4] Jennifer Preece Helen Sharp Yvonne Rogers. *Interaction design - beyond human-computer interaction, 5th Edition*. Wiley, 2019. ISBN: 978-1-119-54725-9.
- [5] Jakob Nielsen. “End of Web Design”. In: (2000).
- [6] Yon Yablonski. *Laws of UX*. O’Reilly, 2020. ISBN: 978-3-96009-156-1.

Appendix

8.1 Erster Teil Appendix

8.2 Zweiter Teil Appendix