HAW HAMBURG

BACHELOR THESIS
David Kurniadi Weinardy

Progressive Web Apps for Data Visualization: A Business Intelligence Perspective

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE
WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

David Kurniadi Weinardy

Progressive Web Apps for Data Visualization: A Business Intelligence Perspective

Bachelor thesis submitted for examination in Bachelor´s degree
in the study course *Bachelor of Science Wirtschaftsinformatik*
at the Department Computer Science
at the Faculty of Engineering and Computer Science
at University of Applied Science Hamburg

Supervisor: Prof. Dr. Stefan Sarstedt
Supervisor: Prof. Dr. Martin Schultz

Submitted on: 15. April 2024

David Kurniadi Weinardy

Thema der Arbeit

Progressive Web Apps for Data Visualization: A Business Intelligence Perspective

Stichworte

Progressive Web Application, Data Visualization, Business Intelligence

Kurzzusammenfassung

Um auf dem Markt bestehen zu können, sollten sich Unternehmen und Firmen auf das wichtigste ökonomische Prinzip besinnen: die Gewinnmaximierung bei gleichzeitiger Minimierung von Verlusten. Business Intelligence (BI)-Dashboards haben sich nachweislich als hilfreich für Entscheidungsträger erwiesen, da sie durch die Visualisierung von Geschäftsdaten in Form von Diagrammen und Grafiken Trends im Unternehmen analysieren können. Diese Arbeit schlägt vor, die Kosten für solche Tools zu senken, indem man sich die Vorteile von Progressive Web Application (PWA) zunutze macht, die mit einer einzigen Codebasis eine Alternative zu nativen Anwendungen bieten. Um dies zu belegen, wird im Rahmen dieser Arbeit ein Prototyp erstellt. Dazu werden zunächst die relevanten Forschungsfragen als Einleitung formuliert und die Prototyping-Methodik angewendet. Die Anforderungen an den Prototyp werden aus verwandten Arbeiten und bisherigen Erkenntnissen abgeleitet. Danach werden die Code-Infrastruktur und die Architektur des Prototyps detailliert vorgestellt. Abschließend werden Diskussionen, Schlussfolgerungen und alle relevanten Links, wie z.B. der Zugang zum Prototyp, am Ende der Arbeit eingefügt.

David Kurniadi Weinardy

Title of Thesis

Progressive Web Apps for Data Visualization: A Business Intelligence Perspective

Keywords

Progressive Web Application, Data Visualization, Business Intelligence

Abstract

Businesses and Companies may want to apply the most important principle to survive: maximizing profits and minimizing losses. BI dashboards have been proven to help the decision makers to analyse and companies' trends, through the visualization of business data in form of charts and diagrams. A way to cut down the cost for such tool is, proposed by this thesis, to take the advantage of the PWA, that offers an alternative choice for native applications by having only one code base. To come to proof, a prototype is built along with this thesis, by arranging the relevant research questions as an introduction, and applying the prototyping methodology. The requirements for the prototype are defined from related works and previous findings. Then, the thesis will go in the details by presenting the code infrastructure and the architecture of the prototype. Finally, discussions, conclusions and all relevant links such as where to access the prototype are included at the end of this thesis.

Contents

List of Figures

List of Tables

Acronyms

AI Artificial Intelligence.

API Application Programming Interface.

BI Business Intelligence.

CLI Command Line Interface.

CRUD Create Read Update Delete.

CSS Cascading Style Sheets.

CSV Comma Separated Value.

DBT Data Build Tool.

DOM Document Object Model.

DV Data Visualisation.

ELT Extract Load Transfer.

ETL Extract Transfer Load.

FK Foreign Key.

HATEOAS Hypermedia as the engine of application state.

HCI Human Computer Interaction.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

JS JavaScript.

JSON JavaScript Object Notation.

KPI Key performance Indicator.

MiTM man-in-the-middle.

OLAP Online Analytical Processing.

ORM Object Relational Mapping.

OS Operating System.

PK Primary Key.

PWA Progressive Web Application.

RDBMS Relational Database Management System.

REST Representational State Transfer.

RIA Rich Internet Application.

SQL Structured Query Language.

SSL Secure Sockets Layer.

SW Service Worker.

TLS Transport Layer Security.

UI User Interface.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

UX User Experience.

VAPID Voluntary Application Server Identification.

1 Introduction

1.1 Motivation

When developing strategies for a company to survive in, or even to surpass the market, an analysis of not only the company's overall performance, but also each department's performance, is often vital and needed by the decision makers such as CEOs and managers. The analysis, that is not different from gathered and processed business data, will be handed as information in form of reports through multiple processes of BI [30]. Owing to this reason, the information has to be useful in a way, so that the decision makers could retract and utilize them to determine the next possible action. For example, a raw table of 10,000 orders with quantities and revenues would be hard to understand and is minimally helpful.

In 2022, Huang et al.[17] studied the impact of BI on the financial performance of total 250 startup CEOs, and experts. They concluded that although BI does not show a direct impact on the financial performance, BI have a significant impact on the innovation, and network learning capabilities of the studied startups, which confirmed to have a direct impact on financial performance. In other words, being the meditating role, BI indirectly helps companies to gain financial performance. Statista shows a positive revenue development of BI Software in Europe as well as worldwide over the years, and predicted to grow statically until 2028 (Figure 1.1). These increasing demands confirm the necessity of BI Software.

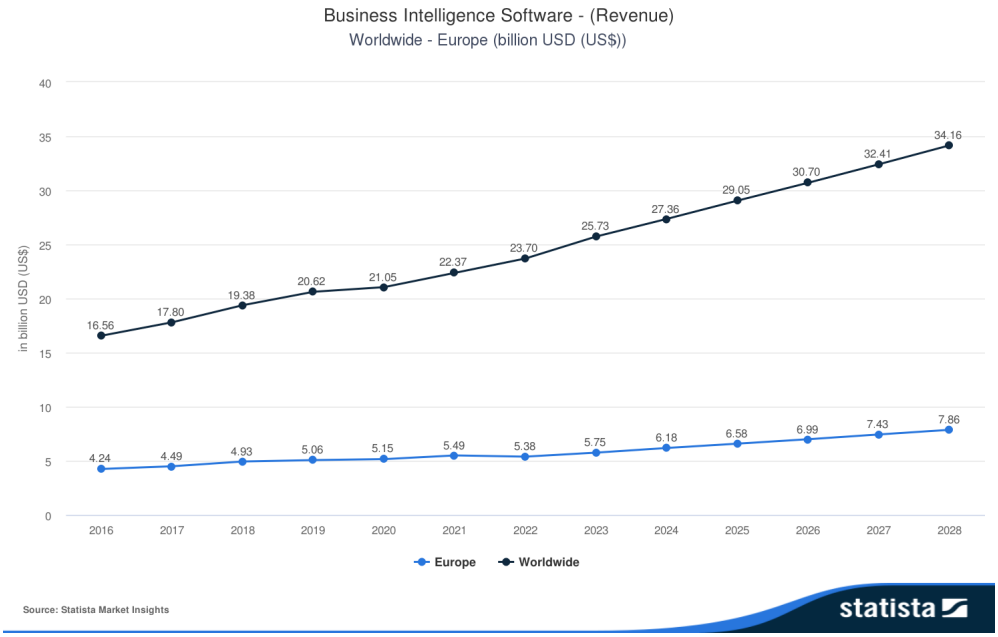

Figure 1.1: Revenue Development of BI Software. Adapted from [42]

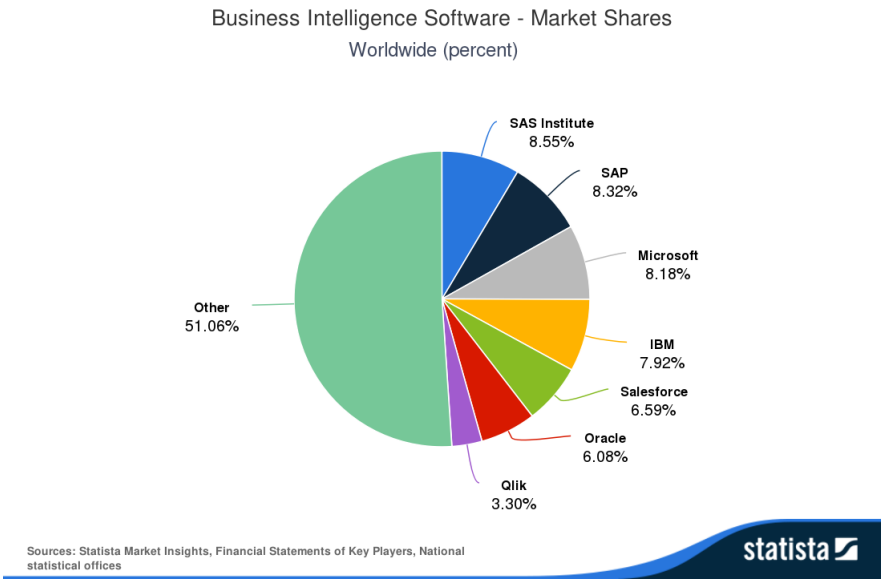

Figure 1.2: Market Shares of BI Software. Adapted from [42]

Statista also shows the worldwide market shares of BI software in Figure 1.2, with SAS Institute (Visual Analytics), SAP (Cognos Analytics), and Microsoft (Power BI) as the

top 3 biggest holders. These software offer excellence solutions for companies to be able to experience the advantages of BI, and are implemented as native applications. A native application is a software program developers build for use on a particular platform or device [13], which focus on user experiences, especially on presenting Data Visualisation (DV)s, and native device features such as file access, Bluetooth, etc. Developing a native application is not an easy matter, as the developers have to constantly monitor, and maintain more than one code base, which when combined with the development of the BI system, could result in a relatively high cost of development, and hence the cost of subscription to such services. The author believes that the web technology PWA could be a promising solution for this issue.

PWAs are fundamentally enhanced web applications, that offer users a native-application-like experience, and work across devices, regardless of the OS. Unlike native applications, PWAs could be installed directly from browsers, and can be built on standard technologies for websites such as HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript (JS). This benefits both the developers and the stakeholders, as the idea of maintaining one code base will reduce the development time, hence minimizing the cost and increase the production rate of features and bug fixes.

1.2 Purpose Statement

Paired with the *state-of-the-art* web technologies, increasing supports from operating systems, and modern BI architecture, it is the author's intention to give an outlook through this thesis within a prototype, that:

RQ1 : PWA technology and its features are able to act as a cross-platform medium for DV in the context of BI application.

RQ2 : PWA could be an affordable technology of choice to deliver a client-side application of a BI software.

1.3 Research Gap

Until the time this paper is written, there is no scientific research that study this particular subject of matter.

1.4 Methodology

This study chose prototyping as its research methodology. IEEE defined prototyping as: "A type of development in which emphasis is placed on developing prototypes early in the development process to permit early feedback and analysis in support of the development process" [18]. (General) Prototyping could be summarized in five iterative phases and one final phase [35]:

1. **Requirement Analysis.** In this phase, the requirements of the software prototype will be analysed. Ways to achieve this phase can be e.g. implementing use case scenarios, establishing a list of requirements, or interviewing a set of users for their expectations towards the application.

2. **Quick Design.** The second phase will provide the system's basic design. However, the design should not be complete, instead it should give a quick overview of the system.

3. **Build.** This phase consist of building the prototype itself, with the requirements and the formed design from phase one and two.

4. **Evaluation.** This phase requires users' input and feedback. There are two outcomes of this phase. Either users are unsatisfied with the prototype, thus the prototype has to be reviewed in phase five, or the prototype meets the requirements and users' expectation.

5. **Refinement.** In this phase, users' feedback from phase four will be reviewed and refined, resulting in new refined requirements. The build phase will then start again, however, based on the refined requirements.

6. **Implement Product.** If the requirements are met and users approve the prototype, the development of the product will begin.

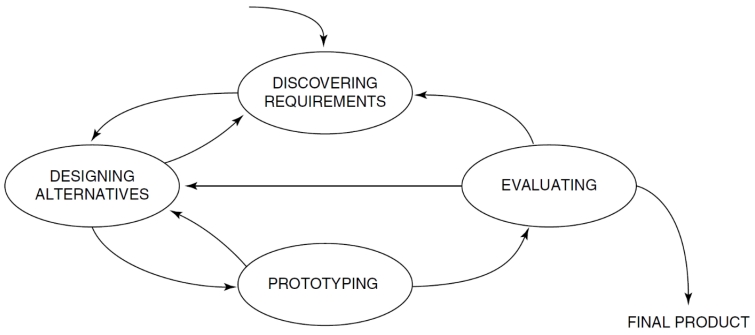

Figure 1.3: Typical prototyping processes. Adapted from [36]

Kordon et al. differentiate prototyping into two approaches [23]. The evolutionary approach, which mirrors the complete iteration of prototyping, produces a system design and a code base, that will be primarily used in the production, resulting in the necessity of more and better resources. On the contrary, the throw-away approach is used, if the available resources are limited and the ability to communicate the advantages of a new approach with a low-cost demonstration can be critical for creating a new project, so the produced code base and system design may or may not be used later in the production.

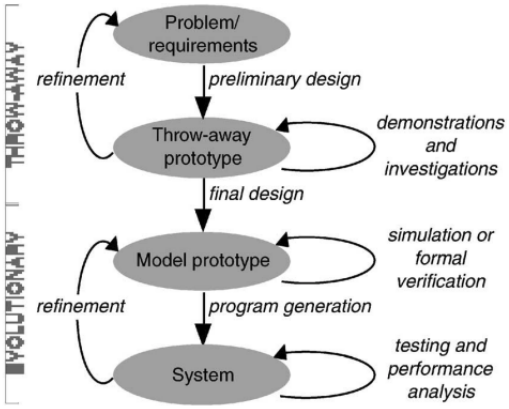

Figure 1.4: Throwaway and Evolutionary Prototyping. Adapted from [23]

This study aims to analyse and demonstrate the combination of BI and PWA technology with limited resources on time and cost. Thus, the author will conduct the throwaway approach of prototyping, where the development will focus mainly on the requirements stated in the fourth chapter 4, and feedback are gathered from self evaluation and the

supervisors of this thesis. Then, for future development, depending on the feasibility of this concept, the prototype could be used as a base for forthcoming enhancements and optimizations.

1.5 Structure

Consisting of eight chapters, the first chapter of this bachelor thesis outlines the motivation, where the research gap lies, and the purpose statement. The second chapter provides an overview about a related work that includes some key information about how to build a BI PWA, and where this thesis could complement that work. The third chapter underlies the theoretical backgrounds of PWA, and BI. This chapter also gives an idea to the readers of what the main concerns of this thesis are. The fourth chapter describes the functional as well as the non-functional requirements, derived from the purpose statement in 1.2. The design details of the prototype, important snippets, and the implementation processes, which are based on the theoretical backgrounds, are included in the fifth chapter. Then, the sixth chapter exposes the results of the implementation in form of snapshots. Here, a short hint for the readers to be able to experience the app directly is also given. In the seventh chapter, a discussion about whether the results provide an answer to the research questions, and if the requirements are all effectively fulfilled. Lastly, this bachelor thesis is concluded in the eighth chapter. In this final chapter, the author gives wthe limitations of this work, and at the same time, some insights for future researches that can be done based on this thesis.

2 Related Works

2.1 MyBI

In 2018, Castilho[5] published an article with the title of *Building a Business Intelligence Progressive Web Application*. As the title of the article said, he, and his team built a BI PWA named MyBI, using User Interface (UI) components from Wijmo [19]. MyBI was built without any external library or framework. The reason is that the creators of MyBI wanted to keep the construction of MyBI simple. This resulted in the application size of 20k. Unfortunately, a valid unit for the application size was not provided. The screenshots of the app were provided in Figure 2.1, and the link to the demo application is also available [47].

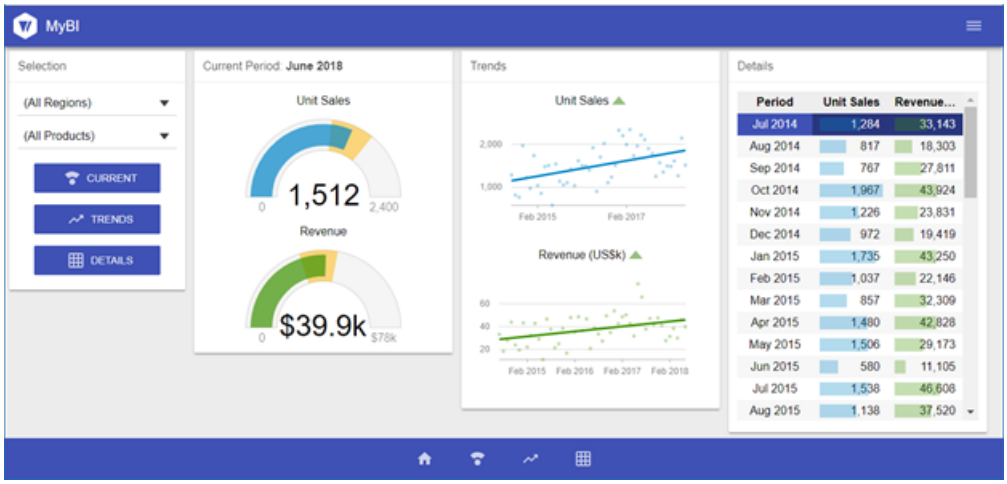

Figure 2.1: Screenshots of MyBI Desktop. Adapted from [5].

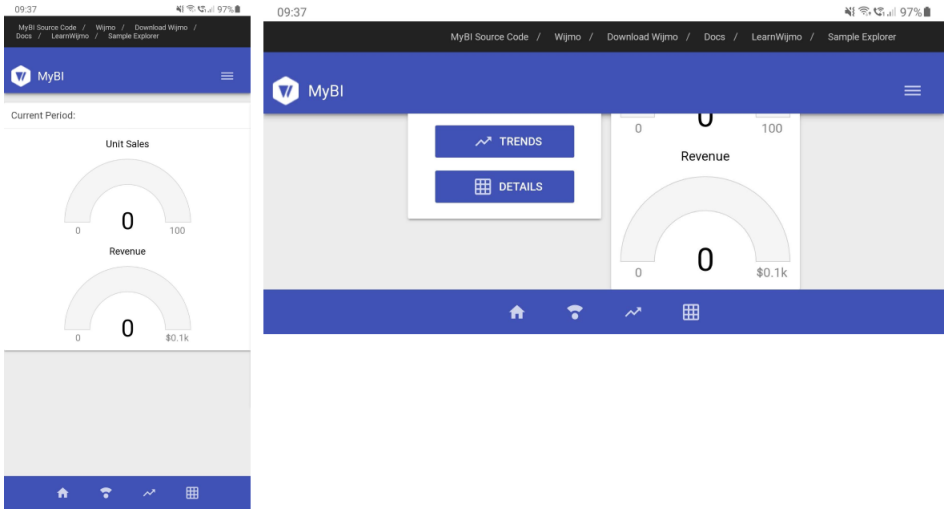

Figure 2.2: Screenshots of the MyBI PWA on a mobile device.

When examining MyBI through the developer tool on Chrome browser, MyBI comes with a service worker file named `serviceWorker.js` and a manifest file named `manifest.json`, which is why the author could install MyBI as a standalone application on laptop (macOS) and smartphone (Android). In depth explanations of service worker and manifest can be respectively found in the subsubsection 3.1.1 and the subsubsection 3.1.1. Furthermore, Castilho included precaching of some static files in the service worker, so that MyBI works offline (Listing 2.1).

As from the interface, the author could conclude that MyBI was already designed to be responsive, i.e. the interface adapts to the screen size of the device, whether installed or not. The main page is a single page, that turns into sections, once the screen width is under 501 pixels(px), and the page become horizontally scrollable instead. The icons in the menu bar refer to the three DV components; a section of two gauge charts, a section of two line charts, and a section of a table. It is also worth mentioning, that Castilho uses the `scroll-snap-type` CSS property, to make the scrolling experience of the charts feels more like a native application than just a website.

Unfortunately, Castilho did not publish the data integration process of MyBI nor the application architecture, e.g. and as can be seen in Figure 2.2, MyBI does not display the existence of any data to analyse. The network requests to the data source gave back the HTTP status code of 410, which means that the access to the target resource is no

longer available at the origin server and that this condition is likely to be permanent [28]. Thus, an analysis of MyBI's functionality could not be done.

```
1  // initialize cache when app is installed
2  self.addEventListener('install', async e => {
3      const cache = await caches.open('mybi-static');
4      await cache.addAll([
5          './',
6          './styles/app.css',
7          './scripts/app.js',
8          './scripts/vendor/wijmo.theme.material.min.css',
9          './scripts/vendor/wijmo-bundle.min.js',
10         './resources/home.svg',
11         './resources/current.svg',
12         './resources/trends.svg',
13         './resources/details.svg',
14         './resources/options.svg',
15         './resources/wijmo-logo.png'
16     ]);
17 });
```

Listing 2.1: MyBI's Precaching Event Listener in Its Service Worker. Adapted from [5]

Despite the insufficient details on the functionality, the article depicts some core ideas on how to build a performant PWA, which inspired the author to give the reader an even more thorough research on this particular matter.

3 Theoretical Background

3.1 Progressive Web Application

In contrary to a website or a web page, which is usually a static HTML file displayed on a web browser (*document-oriented*), a web application offers more aspects of a software rather than a textual depiction (*application-oriented*). Taivalsaari et al.[44][43] divided the web development unto phases three. In the first phase (1990s), the Web is seen only as a distribution environment for documents. In the second phase (2000s), the Web leaned towards the application of the software development disciplines and capabilities. The third phase (2010s and so on), which is occurring as this thesis is written, is the phase where the term Rich Internet Application (RIA) came in. RIA is generally another name for desktop-style web applications, which is also referred to as the *Web 2.0* technologies, where it is not required to reload a whole web page for a single change in a UI component any more, making it dynamic. In this phase, the Web is rather seen as a choice or an alternative to the native application development.

Web applications are available to the end users on browsers, assuming to be built on top of HTML, CSS and JS. Thus, unlike native applications, they should not depend on any OS, as long as these OSs actively support web browsers. Luckily, the three main used OSs on the planet, as of now, macOS, Android and Windows do support mostly all known web browsers, which also indirectly support the development of web applications [41]. The Web Application Programming Interface (API) plays a major role in providing all kinds of implementation for various purposes and functionalities.

From the definition, PWA itself is a web application, yet with the progressive enhancement of the Web. The term *progressive* points to the goal of the PWA, which is to bridge the gap between the native application and the web application.To distinct itself from usual web applications, Lepage, Pete et al.[26] summarize the three most important concepts that a PWA should mirror:

1. **Capability.** A PWA should be capable to project functions that a platform-specific application does in its own right.
2. **Reliability.** A PWA has to give users a good, fast, and reliable experience, regardless of the network connection speed.
3. **Installability.** Users could install a PWA on any device with a browser. So instead of running as a tab in a browser, the PWA runs as a standalone window, that users can launch directly, and therefore, be a part of the device they are installed on.

3.1.1 PWA Requirements

To invoke the progressiveness of a web application, a manifest file and at least one service worker script are needed.

Manifest

(Web App) Manifest acts as a bare bone or a skeleton of the PWA, by providing all the necessary information for the OS to represent the PWA. Usually, a manifest is provided as a single `.webmanifest` file in the root folder of a project, and takes the form of a JavaScript Object Notation (JSON) file structure. Without a manifest, it is impossible to install a web application, as it contains some required fields for the installation process.

Category	**Fields**
Basic	`name`, `short_name`, `icons`, `start_url`, `display`, `id`
Recommended	`theme_color`, `background_color`, `scope`
Extended	`lang`, `dir`, `orientation`
Promotional	`description`, `screenshots`, `categories`, `iarc_rating_id`
Capabilities	`shortcuts`, `share_target`, `display_overrides`

Table 3.1: The various fields of the manifest.

A use case example for the manifest is the splash screen. The splash screen appears, once an application on the mobile environment opens. The basic fields in the Table 3.1

provide exactly the properties that are needed for the splash screen in the context of a PWA. Providing a splash screen also contributes to the reliability aspect of the PWA, and delivers a good User Experience (UX), so that users will not think that the PWA is stop working or stalling [2].

Service Workers

On the contrary to the manifest, SWs are scripts, that are separated from the web page, so they are rather running in the background. Instead of providing static information for the formation of the "shell" of a PWA, SWs and their API facilitate the functionality capabilities. That said, SWs have the major role in deciding the offline capability of a PWA.

There are many use cases and benefits in using SWs within a PWA. The core objectives in using one are:

- **Network Proxy**. An active SW can listen to requests and responses of the PWA. The listening SW can also determine the next action it wants to take as the next step.
- **Precaching and caching**. Persisting cached copies for offline capability is one of many jobs of the SW.
- **Push Notifications**. Like native applications, through an active SW, notifications can be shown in all devices across all platforms.
- **Background Activities**. A SW can also manage background activities, e.g. downloading a file while the PWA, in which the SW works, is closed.

Running separately from a PWA's code base runtime, SWs possess their own life cycle. Understanding the life cycle of SW helps to determine, which strategy is best to provide a better UX than a normal web application presents. The life cycle of a SW can be broken down into five stages or phases.

- **Registration**. A SW has to be firstly registered to the browser. This is typically done by running a registration script while the initiation of the PWA occurs.

- **Installation**. After a successful registration process, the SW will go into the installation phase. In this phase, the SW can perform the precaching mechanism of resources to support the offline capability.
- **Installed**. This phase is reached after a complete installation process.
- **Activation**. A new SW does not automatically control the page that falls within its scope. This phase is great for invalidating old caches and other resources.
- **Activated**. This phase indicates, that a SW is in control and ready to handle events.
- **Idle**. If after some time the SW does not receive any event, it will enter the idle state, where it could either be activated again upon receiving an event, or terminated.

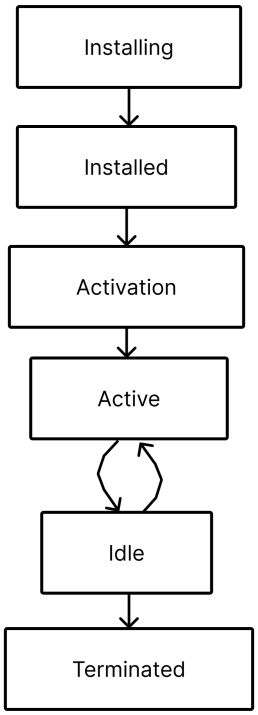

Figure 3.1: A simple graph of the SW life cycle.

3.1.2 Caching Strategies

As stated above, PWAs with the help of SWs utilize cache to facilitate an offline experience. Cache data are typically key-value based, and stored in the browser, locally in the device system. Depending on the use case, there are strategies for caching assets and resources. In his book, Hajian[15] clustered the caching strategies into six patterns.

Cache-Only

In this strategy, the SW will intercept outgoing requests, and match them with the value stored in the cache storage. Practically, the SW will firstly precache all resources in its installation phase. The fallback of a fail match in this strategy is an error message, and updates will come on the next deployment of the PWA.

Network-Only

This strategy does not make use of the cache storage, or rather, the SW does not intercept any request. With that said, using this strategy makes it impossible to provide an offline environment, and depends on the default offline indicator of the browser.

Cache-First

Here, the SW firstly prioritizes the value of a request in the cache storage. If the match fails, the SW will fire the request to the network. This strategy is basically the dynamic approach to the Cache-Only strategy.

Network-First

In this strategy, the SW will always fire requests to the network, and update the value in the cache storage for each request. The SW will only match the request if a network failure occurs as a fallback.

Cache and Network

This strategy expects real-time updates, by firing and matching requests at the same time, asynchronously. An ideal environment to apply this pattern requires a guaranteed response for each request.

Generic Fallback

This strategy provides a guaranteed fallback upon cache and network failure, e.g. by precaching a default offline HTML file.

3.2 Business Data Visualisation

DV is a broad concept, that has been an essential part of the human communication system, especially in the digitalisation era. For example, the weather application widget in commercial smartphones uses different sets of images to represent the weather condition, without having to break the details down in a text form. As defined by Kirk [22], the goal of DV is to facilitate understanding by visually representing data, and being a presentation of data. DV can also be seen as a practice of using graphical representation to provide visual insights in sets of data [12][27].

BI systems as a whole generate insights by utilizing analytical techniques, and then visualizing the insights in the form of dashboards to support decision-making [34]. In other words, DV is an integral part of BI systems, that collaboratively works with other sub-systems. Negash [30] summarizes possibilities of sub-systems within a BI system as seen in Figure 3.2.

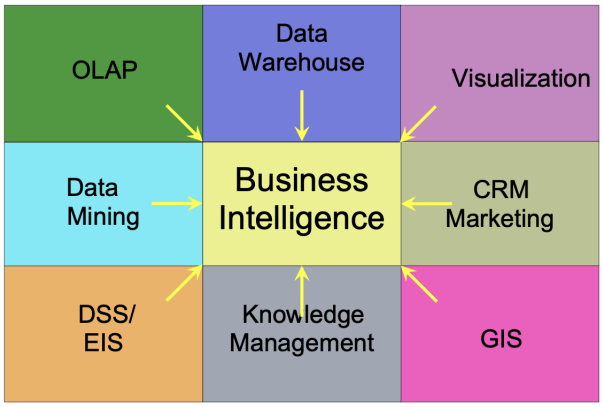

Figure 3.2: OLAP = on-line data processing, CRM=customer relationship management, DSS= decision support systems, GIS = geographic information systems. Adapted from [30]

Now, the concept of *Business DV* specifies, which type of data are being represented. Zheng [48] describes the characteristics of business data as such:

- **Abstract**. Business data mostly narrates abstract activities and processes, which will then be represented as metaphors in an abstract level, e.g. sales, product movement, customer behaviour, etc.
- **Quantitative**. Most of the case, business data are quantitative data, that will later be qualitatively interpreted through analysation processes.
- **Structured or semi-structured**. Business data share common attributes, thus are often structured.
- **Multidimensional**. Consisting of business operation measurements, facts are commonly found in business data, and can be analysed through different dimensions, e.g. time, location, etc.
- **Atomic**. Raw business data, that are based on business transactions, can be understood independently
- **Comprehendible**. Business data can be understood within a certain domain of knowledge in a relatively short timespan.

3.2.1 Forms

Business data visualisation has many forms and representations, depending on the scope of the visualisation. Zheng [48] also groups the forms of the visualisation in BI reporting and analytics unto three categories.

Form	Definition
Embedded Visuals	Embedded visuals are visualisations that are embedded on top of other bigger visualisation, and can be achieved through the use of conditional formattings or spark lines. For example, to indicate trends that are going downwards, the colour red could be used as font colour to indicate loss. Key performance Indicator (KPI)s are also in this category.
Block Visual	Block visuals stand independently in form of diagrams and charts. Depending on its complexity, a block visual can also be a part of a standalone visual.
Standalone Visual	Standalone visuals act as an application, rather than only an ink on the paper visualisation, where users can also interact with the block visuals within. Examples of the implementation for this form are dashboards and complex maps.

Table 3.2: The different forms of the business data visualisation. Adapted from [48].

3.3 Data Integration

Coming from various sources, the (business) data will be marked as raw data, and integrated into a system called the data warehouse. Raw data can have different metadata, characteristics and forms, e.g. a dataset could be either relational or non-relational data. To make operations on these data, which includes the visualizing process, the differences have to be somewhat tackled by integrating these data onto some level, so that the quality of the raw data can be enhanced, and thus making it reliable and trustworthy.

This strategy is a concept in the data warehousing called the data integration. The ELT process is one of two data integration strategies in the data warehouse.

As the name suggest, the raw data will go through three stages of integration:

- **Extract.** The raw data will be extracted from their sources. The sources could be another application, files or databases.
- **Load.** The extracted data will then be loaded or written in the data warehouse, still as raw data. The loaded data will land in a temporary data storage called the staging area.
- **Transformation.** From the point that the raw data are done loaded, the transformation process will be done iteratively in the data warehouse. The transformation process includes data cleansing, data scrubbing, data deduplication and normalization. In a Relational Database Management System (RDBMS) for example, the transformation process ends with the data being stored in tables or views with defined logical structures and relationships.

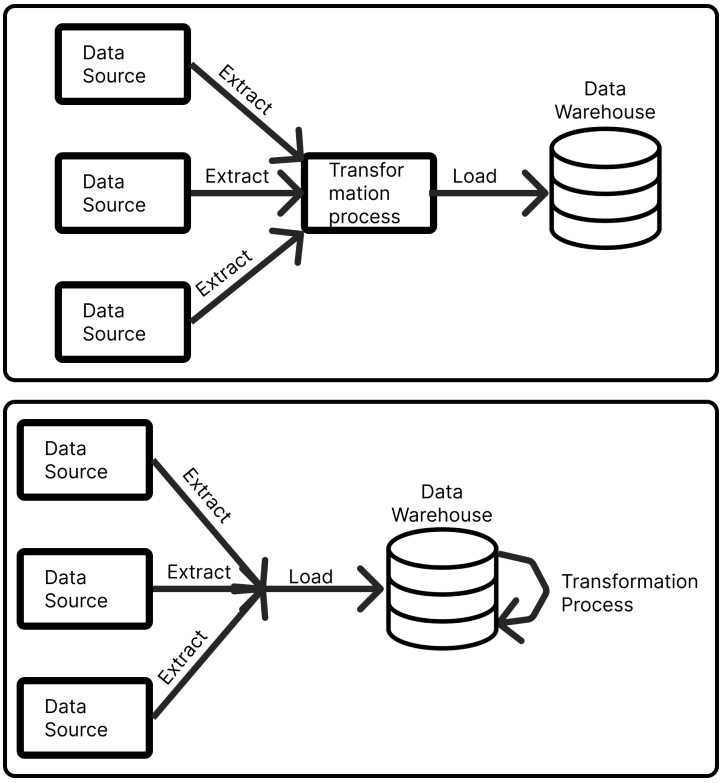

Figure 3.3: Two common practices of the data processing. The ETL (above) and the ELT architecture.

3.4 Dashboard

In the context of BI, dashboards serve as a standalone visual application or the *front-end* of the BI software, as described in the Table 3.2, by combining the concept of the data visualisation and the UI design principles [48]. By maximizing the best practices of the UI principles directly, the effectiveness of human understanding of the dashboard can be enhanced, hence resulting in a form of support for the objective of the DV. For example, Kirk[22] emphasized the effective use of colour in DV for the better understanding process of users and analysers. Providing tooltips and pop-ups for data details also helps to make a more comfortable environment for users to dwell in [37].

Often, users can also find some sort of filtering tools in an interactive dashboard. Filter helps users to highlight some focus points in the dashboard. The mechanism that the filtering tools use is known as Online Analytical Processing (OLAP), an important

concept in the BI plane. OLAP operates on dimensional levels, which business data are. The OLAP operations include Roll-Up, Drill-Down, Slice, Dice and Pivot.

4 Software Requirements

This chapter introduces the functional requirements as well as the non-functional requirements for a prototype of BI PWA. These requirements are derived from the research questions in the purpose statement subsection 1.2 and the theoretical background in the chapter 3.

4.1 Functional Requirements

Reference	Name	Description
FR1	The PWA runs on a secure connection	The essential part of designing a PWA is the user experience (UX) on trust [26]. In the context of web applications, a secure environment could be achieved through a secure connection and/ or communication protocol like the HTTPS. HTTPS ensures protection against eavesdroppers and man-in-the-middle (MiTM) attacks, by using block ciphers and a digital certificate which is verified and then trusted [4][3]. This tremendously helps to gain user's confidence in e.g. installing the PWA on devices or consenting to web features.

FR2	Dashboarding	The prototype should be able to facilitate data collection, analysis and information delivery, which are designed to support decision-making [34]. This can be accomplished through visually representing KPI with charts and graphs.
FR3	Installability on various OS	One of the main goals of choosing PWA as a technology of choice, is its cross-platform capability, despite acquiring one code base only. With the help of a dedicated manifest file and service workers [3], the PWA should always be ready to be installed anywhere and anytime.
FR4	Notification	The PWA, like any other applications, is able to reach users through notifications. A good use case for this requirement is, for example, when the PWA gets an update. Then, we could notify users to open and update the application.
FR5	Data collection	For data collection purposes, the PWA should provide an approach where users i.e. companies could upload raw data to then be processed in the system.
FR6	Data processing	The PWA needs to be able to host data processing operations.

FR7	Filtering	Transformed company data usually contain multiple models and dimensions. Thus, analysing charts in a BI application could be a challenging task. Data filtering helps an analyser to focus on one or some categories. Elias[9] stated, that all participants in one of the studies conducted by her, found data filtering to be required for advanced analysis.
FR8	User feedback	Feedback in Human Computer Interaction (HCI) has been recognized as an importance in improving user performance and satisfaction [38]. In the context of web applications, feedback often come to users as messages, whether they are error or success messages. In a study concerning error messages and affective response by Kukka et al. [25], it is indicated, that friendly and neutral error messages positively affect human's affective states, which contributes in the effort of achieving a good UX of the PWA.
FR9	Offline capability	Another key factor that separates PWAs from the other web applications is, that PWAs could work with the absence of the internet. This also prevents users from coming across the default fallback page of browsers, which is bad, because most native applications have their own mechanism in handling poor connection.

Table 4.1: Functional Requirements

4.2 Non-Functional Requirements

Reference	Name	Description
NFR1	Users know how to install the PWA	Installing PWA from different browsers on different OSs still unfortunately possesses differences. For example, the installation of PWA in Safari takes more steps than the one in Google Chrome. Thus, users may need somewhat of a guide to install the PWA.
NFR2	The PWA should feel more like a native application than a website	Because the UI components of a PWA are rendered through browsers, there will always be a performance gap in many aspects between PWAs and native applications. For instance, in a comparison study done by Fournier, Camille, mobile or native applications are shown to outperform PWAs in general in terms of smoothness [11]. This does not imply, that the gap can not be closed in on. Aside from service workers, advanced web API and CSS could support PWA developers to tackle this problem with creativity.

Table 4.2: Non-funtional Requirements

5 System Design

This chapter covers the system design details of the built prototype. Through this chapter, readers will be able to find out, on which frameworks and libraries the prototype are based on, and how these technologies helped to fulfil the functional as well as non-functional requirements on the previous chapter. Furthermore, this chapter also provides all technical strategies on caching and UI design of the prototype.

5.1 Architecture

At this point, the prototype could be broken down into three main components:

- **Presentation Unit.** A component, where all user interactions occur and therefore, visualisation is one of the important concerns in this unit.
- **Processing Unit.** This component includes all relevant BI processes, such as ELT and Create Read Update Delete (CRUD) operations.
- **Data Storage.** Finally, a concern where the unprocessed as well as processed data will be persisted, makes it possible to access the data anytime ideally.

Each component contains a complexity on its own, and each complexity adds weight to the prototype. Therefore, on the architectural level, it is rather wise to separate these components to not only increase the maintainability of the prototype, but also to decrease its complexity by clustering or separating the concerns of each component. The two-tier architecture could satisfy this purpose.

The two-tier architecture is a variant of the client/ server architecture, where processes of an application are separated into two concerns, simplifying the development process and communication on both ends, and thus, being a suitable architecture for prototyping in a resource-limited environment [20]. The client side of this architecture consists of elements

that are relevant for presentation purposes, e.g. in the context of web applications, the code base of this tier usually contains HTML, CSS and JS. Whereas the second tier deals with the business logics of the application or deals directly with the database, which runs on a different server or machine than the first tier.

There are several preferences when it comes to web communication protocols. Although not every website is applied with HTTPS, it is the fundamental cryptographic protocol to secure information in transit, and to ensure data integrity and privacy between two communicating parties [24]. HTTPS, in combination with the Transport Layer Security (TLS) network protocol, encrypts the header and body content of HTTP's requests and responses, avoiding the modifications of the contents through the MiTM attacks. Also, to make the prototype installable, it has to be secured over HTTPS, which makes it a win-win situation.

Figure 5.1: Overview of the encrypted contents in a HTTPS request from an unencrypted HTTP request. Adapted from [14]

The difference of HTTP and HTTPS lies in the availability of the Secure Sockets Layer (SSL) certificate. For manually hosted sites, which is the server side of the prototype, the certificate can be obtained through Let's Encrypt, an open source certificate authority. The certificate then will be applied to the configuration file of the web server software. The illustration of the architecture can be seen in Figure 5.2.

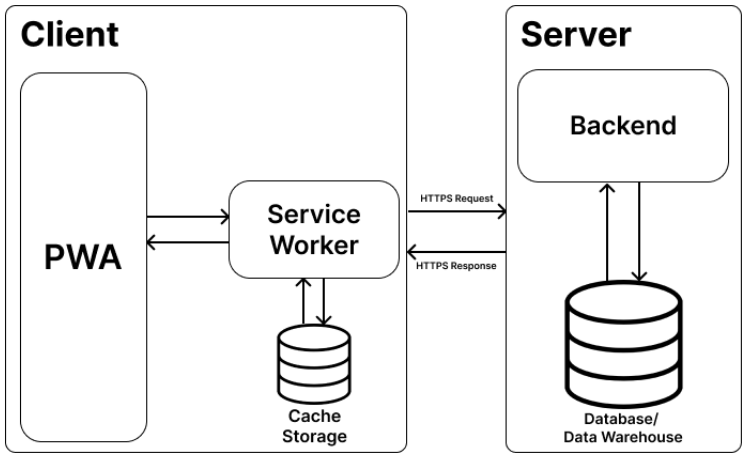

Figure 5.2: The architecture of the prototype and the components in each tier.

Figure 5.2 also shows, that there are basically two data sources. One being the database itself in the server tier, and one being the cache storage in the client tier. This means, that not all the requests coming from the client are being sent to the server, but will be intercepted by the service worker. This plays a huge role in realising the idea of an offline PWA as an answer for the requirement 4.1. More on the (pre-)caching strategy of the prototype can be found in the section 6.8.

5.2 REST(ful) API

The prototype is implemented with the Representational State Transfer (REST) API (also known as RESTful API) on top of the HTTPS communication protocol. RESTful architecture puts it main concern on transferring complete resources over the web, without managing any state on the server, as stated on the introduction of REST by Fielding[10]. In other words, a response from the server contains all information that a request needs. In the frame of the prototype, the responses will be obtained as JSON strings.

Now, in the maturity model of REST APIs defined by Richardson[33], the full maturity is obtained, if:

- the request uses HTTP connection (Level 0),

- the Uniform Resource Identifier (URI) of the response utilizes unique identifications to allocate resources (Level 1),
- the request HTTP verbs (Level 2), and
- the next state is controlled by including Hypermedia as the engine of application state (HATEOAS) links in the response (Level 3).

The requests and responses of the prototype's API can be classified to the level 2 Richardson maturity level, as there is no necessity in the requirements, where the server determines the next possible action for the client. Moreover, the PWA (client) is capable of managing its own state through runtime and browser sessions, and depending on HATEOAS will rather make the PWA static, which is a counter-productive approach in realising an application-like feeling for the PWA.

```
// Request
GET /crossbi/v1/api/views/2 HTTP/1.1

// Response
{
    "data": {
        "aggregate": "sum",
        "categories": null,
        "dashboard_id": 1,
        "date_column": "transaction_date",
        "diagramm_type": 1,
        "id": 2,
        "name": "annual_revenue",
        "title": "Total Revenue 2011 - 2014",
        "updated_at": "Sat, 30 Sep 2023 00:00:00 GMT",
        "workspace_id": 1,
        "x_axis": "transaction_year",
        "y_axis": "total_revenue"
    },
    "message": null
}
```

Listing 5.1: An example of typical request and response of the prototype.

5.3 Database

BI environments are tightly connected with RDBMS, which is desired, because it is known to provide tools, that can be utilized during the ELT processes [8]. One of the widely used RDBMS in the development world is PostgreSQL. As an open-source RDBMS, PostgreSQL offers proven data integrity and robustness, that is perfect from small to enterprise level projects. Because PostgreSQL also always tries to conform to the Structured Query Language (SQL) standards (SQL:2023 Core conformance), it holds a good developer experience. In the prototype, the PostgreSQL instance is fetched as a Docker image and runs in the same network as the backend instance. The prototype utilizes tables to structure day-to-day and raw data, and views to structure the transformed data from the ELT process. Views will then be referenced as sources for the DV part. Figure 5.3 shows the structure of the tables, each with a Primary Key (PK) and Foreign Key (FK)(s), if needed.

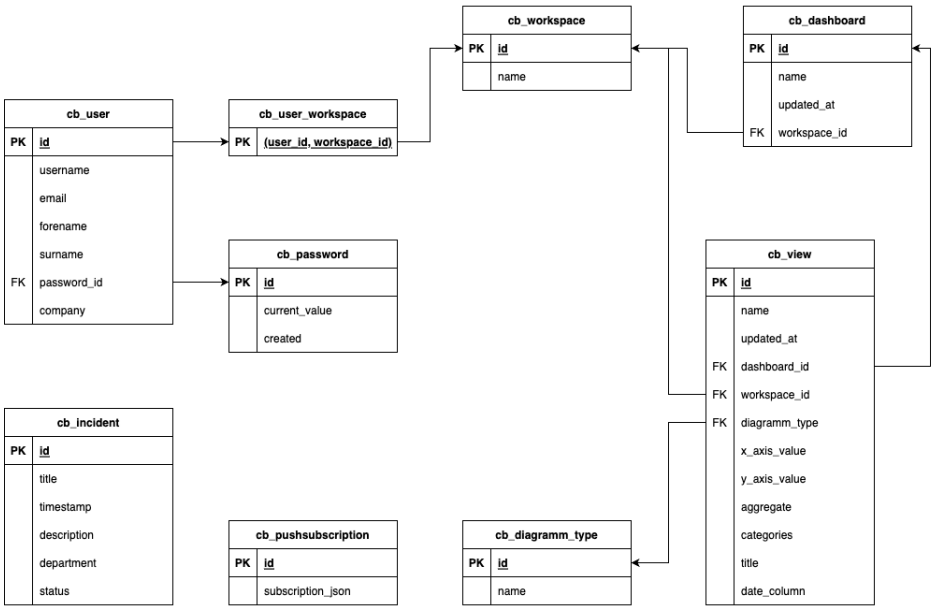

Figure 5.3: The structure of the tables in the database.

Organizationally, the prototype manages its users by assigning them to workspaces, which can be observed in the table `cb_user_workspace`, where the PK of the table is a unique combination of a user ID and a workspace ID. Each user possesses a hashed password, stored in the `cb_password` table, and referenced as the `password_id` FK.

While a workspace's existence is more straightforward, a dashboard (`cb_dashboard`) is a more flexible concept. Dashboards are meant to group the instruments of the DV into focused clusters, e.g. a dashboard that is focused on the production department of a company will only show data or KPIs, that are relevant to that department. A workspace can have several dashboards, as well as views (`cb_view`). A view table contains all information about a literal view object of the PostgreSQL, i.e. the title of a graph, categorization of the data, etc. The information from the view table will then be interpreted by the backend code to serve the actual data from the view object to the client. `cb_diagramm_type` is there to assure the consistency of the diagram types both in the client and the server. The `cb_incident` table holds urgent messages and alerts, whereas the `cb_pushsubscription` contains all registered devices, that are consenting to receive push notifications.

5.4 Frameworks and Libraries

This section hands out all the relevant libraries and frameworks, that helped the development process of the prototype. It is important to note, that the dependencies in the backend are installed within a python virtual environment. This ensures that the root system's dependencies and python version of the server are isolated from those on the backend.

5.4.1 Flask

Flask is a python micro web framework, that is known for its lightweight and least opinionated upbringing, as it initially depends on only two python libraries, Jinja and Werkzeug. Thus, it also offers flexibility in terms of how the framework is being used, by adding the necessary extensions. For example, to connect with the prototype's RDBMS, the package `flask-sqlalchemy` is added as an Object Relational Mapping (ORM) library, so that the backend could build connections, map tables into entities (classes), and execute transactions into the RDBMS. With that said, the code in the backend uses the object-oriented approach.

Since the backend code contains business logics to process data, it is implemented with the controller-service-repository architecture, and follows the repository pattern as a complementary to the architecture. In this architecture, the requests from the PWA will

be received firstly by the controller. For CRUD operations, the controller will retrieve resources directly from the repository. If a request acquires a complex business logic, the controller will firstly call methods from the service, where all business logics exist. This allows an abstraction of the objects and reduces repetition of the code, resulting in decoupling of dependencies between modules [31]. Since abstraction and decoupling are broad concepts on their own, it is out of the scope for this thesis to dwell further on those topics.

5.4.2 Meltano

Often used to optimize and schedule ELT pipelines for BI and Artificial Intelligence (AI) applications, Meltano is an open-source data integration tool, that orchestrates extractors, loaders and transformers for ELT processes. Meltano is also a Command Line Interface (CLI) based automation tool and prioritizes configurations in its workflow.

The extractors and loaders in Meltano are called taps and targets, concepts that are derived from Singer, an open-source standard for writing scripts to move data. So, taps will extract data from data sources and targets will load them. The communication between singer taps and targets is intermediated with JSON. Because Singer has an active and growing community, there are wide-ranging preset taps for various data sources and targets for loading media. As a demonstration, the prototype uses the `tap-spreadsheet-anywhere` package, an extractor that pulls out data from Comma Separated Value (CSV) or Excel files, and the `target-postgres`, a loader that loads data to a PostgreSQL instance.

As for the transformer, Meltano adapted DBT to translate the raw extracted data from previous steps into meaningful information (business queries) and materialize them into tables or views. With DBT, the transformation process could be well implemented into software engineering concepts, such as reusability and modularity. For that, the `dbt-postgres` plugin will be combined with both the extractor and the loader. The ELT workflow will be discussed in section 6.5.

5.4.3 React

As a JS library, React helps developers to build responsive websites whilst maintaining the website's performance, by introducing the virtual Document Object Model (DOM).

DOM is created by the browser once a webpage is loaded, and can be altered by JS [45]. In React, virtual DOMs are created as a lightweight copy of every actual DOM object. When a DOM object is updated, React firstly updates the entire virtual DOM. Then, React compares the virtual DOM with the last snapshot of the actual DOM, to detect which object that needs to be altered. This way, instead of re-rendering all actual DOM objects of the webpage, React only applies the change, where it is really needed. This advantage could be used to implement a good and responsive UX for the prototype.

5.4.4 VitePWA

A complex client-side code contains a huge amount of JS for the mentioned reasons. That is why, JS developers will often need a bundler to translate the written code, that they understand, to a bundled, packaged and minified version of the code, that browsers understand (static files). Examples of known JS bundlers in the front-end community, are Webpack, Parcels and Rollup. The prototype will use Vite (based on Rollup) as its bundler. Beside of the fast build-time for production, Vite actively uses native EcmaScript modules and Hot Module Reload (HMR) technology, which in consequence, establishes a fast development server and thus, a suitable bundler for a rapid development environment. VitePWA is an extension of the Vite bundler and helps developers tremendously in implementing concepts of PWA such as the Web App Manifest and precaching, with a minimal configuration. VitePWA also makes it possible to enable PWA development in a local system.

5.4.5 Workbox

As a JS library, Workbox makes the development of SW easier and more interactive, by providing different kinds of approaches to generate SWs almost automatically. This way, developers can flexibly decide which approach is best for the generation of the SW in their application. VitePWA utilizes and integrates Workbox methods into their workflow.

5.4.6 Tremor

Tremor is a component-based and open-source library for analytical UIs, crucial for the DV of the prototype. Because Tremor is built on top of React, the library could be integrated quite well and fast. To match with the mechanism of Tremor, the prototype

needs to alter the data, that comes from the API, by wrapping each UI component inside a compatible React component.

5.5 Deployment

Both sides of the prototype need to be opened up to the public, so that it is available for any device at any time. To demonstrate the two-tier architecture, the prototype's server-side and client-side are deployed on two different environments.

The prototype's server-side (the backend and the database) was deployed as a docker container in a virtual server and is available on the subdomain `https://api.crossbi.de`. Managing the opened ports of the docker images and the routing mechanism on the server is Nginx, a web server software natively installed on the virtual server. To establish this, the author needed to apply some routing configurations on the Nginx config file. The SSL certificate is also applied on the config file. On the other hand, the client-side is deployed on Netlify, an automated platform for deploying and hosting websites. Netlify has a built-in integration with GitHub, where the repositories of the prototype could be found. The client can be accessed on the address of `https://app.crossbi.de`.

For debugging and administration processes, the database's administration tool (PgAdmin) is available at `http://database.crossbi.de` and will not be exposed to the public. Although theoretically anyone could open the database' address on a browser, the instance is layered with Basic Auth security and admin login authorisation, to ensure that only the author has access to this tool.

6 Implementation

This chapter covers the implementation of the prototype. The chapter goes about the backbone of the PWA, then explains the workflow of the ELT process in the server, and finally clarifies the useful features of the PWA, that helps to satisfy the requirements in the chapter 4. The implementation stemmed on the theoretical background in the chapter 3. This chapter also includes screenshots and code snippets of the prototype.

6.1 File Structure

To understand and grasp the substantial elements of the prototype, the author needs to hand out the file and folder structure from both the frontend and the backend code, which can be observed below. The structures will only include folders or files that are necessary for the subchapters after the current one. The remaining folders and files can be found in the source code in 9.

6.1.1 Front End

```
cross-bi-frontend
|-- src
|   |-- assets
|   |-- components
|   |-- contexts
|   |-- hooks
|   |-- pages
|   |-- styles
|   |-- types
|   |-- App.tsx
|   |-- enums.ts
|   |-- index.css
|   `-- main.tsx
|-- public
|-- statics
|   |-- manifest.json
|   `-- topojson.json
|-- .env
|-- index.html
|-- package.json
|-- registerSW.ts
|-- sw.js
|-- vite.config.ts
`-- ...
```

Listing 6.1: The file structure of the front end code.

6.1.2 Back End

```
cross-bi-backend
|-- .db
|-- .meltano
|   |-- extractors
|   |-- files
|   |-- loaders
|   |-- logs
|   |-- run
|   |-- transformers
|   `-- ...
|-- assets
|-- src
|   |-- controllers
|   |-- repositories
|   |-- services
|   |-- auth.py
|   |-- models.py
|   `-- ...
|-- transform
|   |-- macros
|   |-- models
|   |-- profiles
|   |   `-- postgres
|   |       `-- profiles.yml
|   `-- dbt_project.yml
|-- app.py
|-- config.json
|-- docker-compose.yml
|-- Dockerfile
|-- init.sql
|-- meltano.yml
|-- requirements.txt
`-- ...
```

Listing 6.2: The file structure of the back end code.

6.2 Manifest

As described in the chapter 3, the manifest is needed as some sort of metadata to convert a web application into a PWA. With that said, the manifest file of the client side of the prototype is stored in the repository as a JSON file. In regard to the VitePWA plugin, the manifest file has to be imported as a JS object in the `vite.config.ts`. The Listing 6.3 shows the content of the manifest file of the PWA.

```
1  {
2    "name": "Cross BI",
3    "short_name": "Cross BI",
4    "description": "Your Cross Plattform Business Intelligence App",
5    "id": "/",
6    "theme_color": "#fff",
7    "background_color": "#fff",
8    "display": "standalone",
9    "start_url": "/",
10   "icons": [
11     {
12       "src": "pwa-192x192.png",
13       "sizes": "192x192",
14       "type": "image/png",
15       "purpose": "any"
16     },
17     {
18       "src": "pwa-512x512.png",
19       "sizes": "512x512",
20       "type": "any maskable",
21       "purpose": "any"
22     },
23     {
24       "src": "pwa-192x192.png",
25       "sizes": "192x192",
26       "type": "image/png",
27       "purpose": "maskable"
28     },
29     {
30       "src": "pwa-512x512.png",
31       "sizes": "512x512",
32       "type": "image/png",
33       "purpose": "maskable"
34     }
```

```
35    ],
36    "related_applications": [{
37      "platform": "webapp",
38      "url": "https://app.crossbi.de/manifest.json"
39    }],
40    "screenshots": [
41      {
42        "src": "/sc-portrait.png",
43        "sizes": "778x1684",
44        "type": "image/png"
45      }
46    ]
47  }
```

Listing 6.3: The content of the manifest file

The manifest members are typically shown in the developer tools' of the browser. Chrome and Firefox interactively show the manifest, by grouping the manifest members into sections. Because each browser may or may not support some features, the sections can also vary. For example, aside from the identity, display and icons sections, the Chrome browser also shows some experimental members such as `related_applications` and `screenshots`. Unfortunately, the Safari browser does not have this feature, and therefore show the manifest rather as a plain JSON object. For Chromium browsers (Chrome, Edge and Opera), two sizes of icons (192x192 pixel and 512x512px) are desired, as it will resize the icons automatically to fit the device [1]. The `maskable` purpose in the icon specification tells the device, that the actual graphic of the icon is within the safe zone, and therefore, can be masked.

Figure 6.1: Two different displays of the prototype's icon in two major OSs after installation. The left icon is the masked icon on Android and the right icon is the masked icon on iOS.

It is also noticeable from the manifest, that the author picked **Cross BI** as the prototype's name and the prototype's display will be standalone, as an answer to the second non-functional requirement 4.2. By setting the display as standalone, the installed prototype will have its own window toolbar in the desktop, and in all devices, the prototype will not be opened as a browser tab, but as a conventional application.

6.3 Service Worker

The SW is also part of the front end code base. In the Listing 6.1, the SW can be spotted with the name of `sw.js`. The file is then to be registered in the VitePWA configuration, as seen in the Listing 6.4 on the seventh code line below. The `registerSW.js` file is needed to initiate the registration of the SW. The initiation begins after all the resources of the application page are loaded, in which the `load` event of the browser's `window` object will then be fired.

```
1  ...
2  VitePWA({
3      manifest: manifestObject,
4      registerType: 'autoUpdate',
5      strategies: 'injectManifest',
6      srcDir: '.',
7      filename: 'sw.js',
8      injectRegister: null,
9      workbox: {
10       globPatterns: ['**/*.{js,css,html,ico,png,svg}']
11     },
12     devOptions: {
13       navigateFallbackAllowlist: [/^index.html$/],
14       enabled: true,
15       type: 'module',
16     },
17     selfDestroying: shouldSelfDestroy == 'true'
18 }),
19 ...
```

Listing 6.4: The configuration bit of the VitePWA plugin.

6.3.1 Registration

There are two approaches in Workbox to register a SW for an application. One being the `generateSW` approach, where Workbox automatically creates a SW file. While this approach is able to provide precaching process and runtime caching, developers can not alter the file, which makes it impossible to utilize further SW features. The other approach is the `injectManifest` approach, which consecutively offers precaching process, runtime caching and the flexibility to obtain SW features, resulting in a steeper learning curve in the SW development. To be able to make the best of existing PWA features, which is needed to answer the research question item **RQ1**, the author chose the `injectManifest` approach for the prototype. This approach has also to be defined in the VitePWA configuration file, as seen on the fifth line of the Listing 6.4.

Same as the manifest, a successful SW registration could be indicated through the developer tool of a browser. If the registration is unsuccessful, the developer tool will point out the cause in the SW.

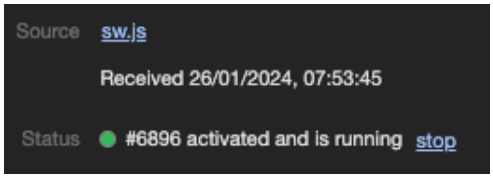

Figure 6.2: An active SW of the prototype.

A change in the SW creates a new version of the SW, which makes the old version outdated. If the new version is successfully registered, browsers will first compare the newly registered SW with the old one. If there is a difference between the two, browsers will outdate the old one and substitute it with the new version. Though, the update does not happen instantly, as users will need to manually reload the browser to let the new SW have control over the client. There are some mechanisms to update the SW provided by Workbox. The `registerType: autoUpdate` key-value in the Figure 6.2 for example, tells the browser to automatically update the SW. Additionally, the two lines of code in the Listing 6.5 have to be present on the top level of the SW file.

```
1 ...
2 self.skipWaiting();
3 clientsClaim();
4 ...
```

Listing 6.5: `skipWaiting()` is used to pass through the waiting phase of the SW, while the Workbox method `clientsClaim()` is used to let the changes on browsers instantly.

While this sounds practical, users might still get confused because of the sudden update. Thus, the prototype provides a mechanism, by listening to the `updatefound` event of the `ServiceWorkerRegistration` type [29]. If the event occurs, a dialog element will appear on the screen to notify users about the ongoing update, as can be noticed in the Figure 6.3.

Figure 6.3: The update dialog on the prototype.

6.3.2 Caching Strategy

To facilitate the offline functionality off the prototype, the network first caching strategy is implemented in the SW.

```
1  ...
2  self.addEventListener('fetch', (event) => {
3    let url = event.request.url;
4
5    if (!url.startsWith('http') || event.request.method !== 'GET') {
6      return;
7    }
8
9    event.respondWith(
10     caches.open(CACHE_NAME).then(async (cache) => {
11       const headers = new Headers();
12       headers.append('x-api-key', API_KEY);
13       headers.append('Content-Type', 'appplication/json');
14
15       return fetch(url, { headers: headers })
16       .then((fetched) => {
17         cache.put(event.request, fetched.clone());
18         return fetched;
19       })
20       .catch(() => {
21         return cache.match(url);
22       });
23     })
24   );
25 });
26 ...
```

Listing 6.6: A fetch event interceptor in the SW.

The SW listens to the `fetch` event, i.g. the outgoing requests from the prototype to the server. However, not all requests are intercepted. The SW practically ignores all requests that are not HTTP, or all requests that are not using the `GET` method. Otherwise, the SW will fire the request further, and add or update the cache value with the Uniform Resource Locator (URL) as its key. This way, the cached response, e.g. the data of a particular view, will always be overwritten, until the prototype must switch to the offline mode because of network absence, where the SW only retrieves data from cache.

6.4 Installation

To see, if the prototype is installable in all existing browsers, could be a challenging task. The data in the Figure 6.4 gives a hindsight on how to tackle this.

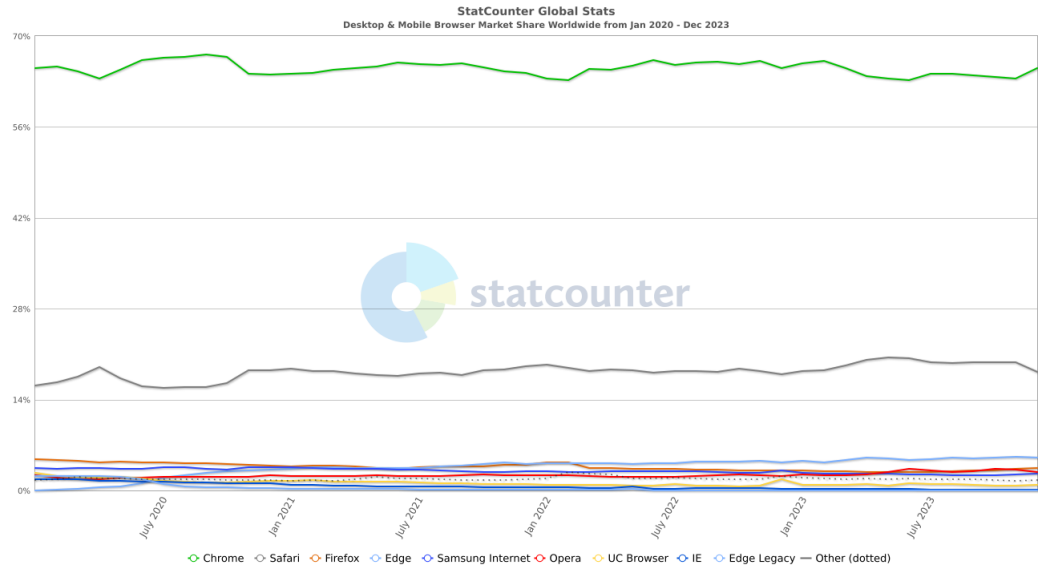

Figure 6.4: The market share of major browsers for all platforms from January 2020 until December 2023. Adapted from [39].

From the chart above, it can be concluded that the Chrome browser dominates the market share of browsers for all platforms worldwide, until now. Having roughly 65% of the browsers market share, Chrome is built based on the chromium engine, which browsers like Edge, Opera and Samsung Internet also are. If combined, Chromium browsers take more than 70% of the market share. This is rather not shocking, because all android devices come with pre-installed Chrome. In terms of the PWA development, Chromium browsers are friendlier towards developers, as most of the experimental features are already available in them, both in desktop and mobile devices.

Also, due to the 18.5% worldwide market share the Safari browser possesses, it is beneficial to review the installation of the prototype in Safari. While Chromium browsers take most of the browser's market share, according to Statcounter, Apple devices lead the mobile vendor market share worldwide with a percentage of 29.32% (until January 2024) [40]. Apple devices (iOS) comes with Safari, and it is the default native browser on these devices. Therefore, contributing to the functional requirement 4.1, an overview

of the installation of the prototype in desktop with Chrome and Safari, in android with Chrome Mobile and in iOS with Safari is provided in this section.

6.4.1 Desktop

Chrome

After navigating to the PWA link on the chapter 9 through Chrome, users will be greeted by the installable indicator in the URL input section. By clicking the indicator and agreeing to the installation prompt, the prototype will be installed on the device, and it is ready to use.

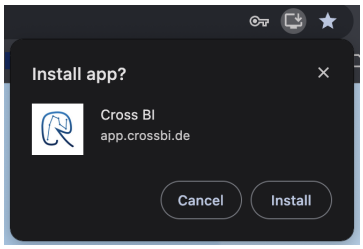

Figure 6.5: The install prompt in Chrome desktop.

Safari

The installation on safari takes a few more steps than Chrome. In the toolbar, users will need to click on the share button. From there, users will be provided with the *Add to Dock* option. By clicking the option and the add button on the *Add to Dock* dialog as seen on the Figure 6.6, the prototype will be added into the device as an application.

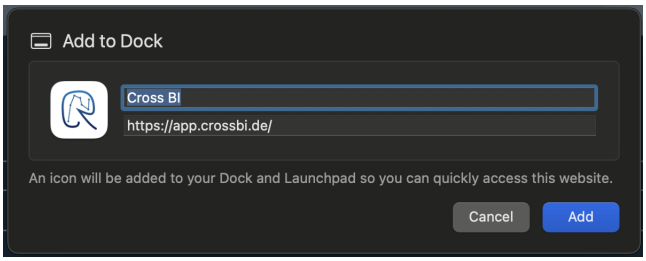

Figure 6.6: The Add to Dock dialog in Safari desktop.

6.4.2 Mobile

Chrome Mobile on Android

Unlike the desktop version, the mobile version of Chrome does not prompt the availability of the prototype's installation. Users will need to press on the menu (the three dots on the right) and then install the prototype. Additionally, Chrome provides an experimental feature to preview the application before installing. This can be achieved through the `screenshots` property in the manifest file.

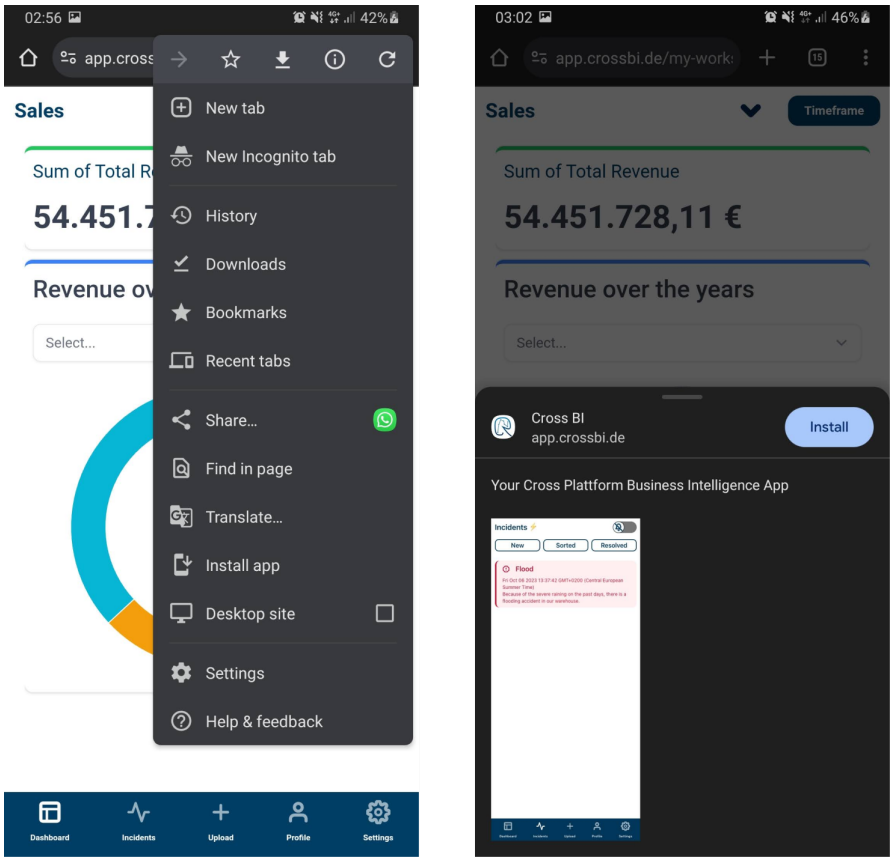

Figure 6.7: The prototype's installation steps in Chrome mobile.

Safari Mobile on iOS

On the other hand, the installation in Safari mobile does not expose too much of a difference with the desktop version. On Safari mobile, instead of clicking *Add to Dock*, users will need to choose the *Add to Home Screen* option.

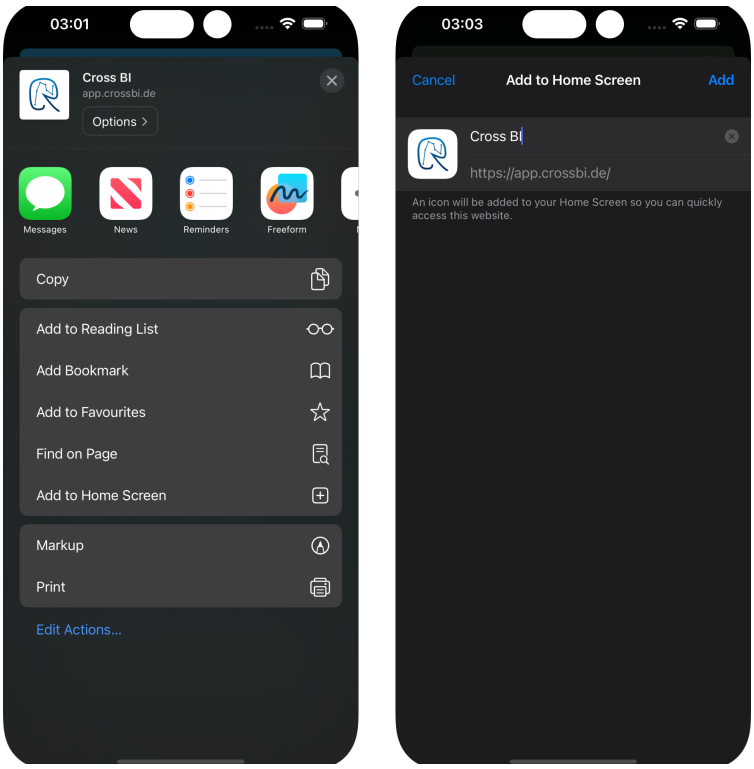

Figure 6.8: The prototype's installation steps in Safari mobile.

6.5 Data Processing

As mentioned, the ELT process is one of the core processes of the prototype, and it will be executed by the Meltano framework. To engage in this process and to perform the ELT operation, the author has to namely access the remote server. The plugins, which are added through the CLI `meltano add <plugin>`, can be summarized as follows:

Role	Plugin	Description
Extractor	tap-spreadsheets-anywhere	extracts data that comes from CSV or Excel files.
Loader	target-postgres	loads the extracted data to the PostgreSQL instance of the prototype.
Transformer	dbt-postgres	transforms, cleanses, and refines the data modularly.

Table 6.1: The meltano plugins of the prototype.

By successfully running the `meltano add` command line, Meltano will alter the content of the `meltano.yml` file. From there, some configurations will also need to be manually re-configured to adapt to the file structure and workflow of the prototype. The details of the plugins will be elaborated in the next subsections.

6.5.1 Data Source

The data source is taken from Kaggle, a platform to find various datasets. Since BI is business related, the author sampled the *Retail Case Study Data* dataset, whose link can be found in the chapter 9 below. The initial dataset contains three CSV files:

- **Customer.csv** contains information about the customers of the retail, which are `customer_id`, `DOB`, `gender`, and `city_code`.
- **Transactions.csv** acts as the "fact" data, that gathers all information about the transactions' history of the retail. This includes the customer ID, the transaction date, product category code, etc. There are 23,054 recorded unique transactions in the file.
- **prod_cat_info.csv** yields the code and description of the category, as well as the sub category of the products available.

The author added the `Cities.csv` file as an extra data source, that describes the cities and countries of each code in the `Customer.csv` file. The files are located in the `assets` folder (Listing 6.2) and are organized based on the `workspace_id` they are assigned to.

The PWA also offers users the ability to upload CSV or Excel files through the Upload section.

Figure 6.9: The Upload feature of the PWA. The first image (left) shows the state of the PWA, after a file is given. If the upload is successful, then the screen will change to the second image (middle). The third image on the right indicates the behaviour of the PWA, if a false formatted file is given.

6.5.2 Extract

The `tap-spreadsheets-anywhere` extractor needs an array of objects called "tables" in its configuration. Each object of the array describes the required and optional properties of each data source, or in this case each file. The properties, that are used in the implementation, can be observed in the Table 6.2 below.

Property	**Necessity**	**Description**
delimiter	Optional	The delimiter in the raw CSV files.
key_properties	Required	Unique identifier(s) of the data, that will eventually be used as the primary key of the target connector.

name	Required	Name of the extraction, or in other words the Singer stream. The defined name will be used by the target as the name of the table.
path	Required	The directory of the source file.
pattern	Required	The name of the file in regular expression.
quotechar	Optional	A specific character surrounding a string that contains a delimiter character.
start_date	Required	The start date that the tap uses to filter the files, based on the last modified date of the files.

Table 6.2: The properties of a table object.

6.5.3 Load

Listening to the Singer Stream produced by the tap, is the target. Thus, the target can not be executed without the tap, as this will result in an infinite loop in the algorithm. A minimal configuration properties of the `target-postgres`, that leads to successful writings of the stream into the database, are as follows:

Property	**Necessity**	**Description**
database	Required	The name of the database inside PostgreSQL instance.
default_target_schema	Required	The default schema of the target, for instance the PUBLIC schema.
host	Required	The reachable address of the instance.
port	Required	The port of the host.
user	Required	A username of a privileged user. The role of the user can be defined in the PostgreSQL instance.
password	Required	The password of the given username.

Table 6.3: The configuration properties of the `target_postgres` plugin.

6.5.4 Transform

The DBT transformation concept is based on building recyclable models. Two ways to define a DBT model are through defining SQL models with SQL files, or python models with python files. To avoid adding more package to the current python environment, the author chose to define the SQL methods to define the DBT models. As recommended by DBT, the models are separated into three folders:

- **Staging**. The folder for the DBT models that select, cleanse and transform the raw generated tables from the database/ data warehouse.
- **Intermediate**. The folder for more focused and granular models.
- **Marts**. Refined and meaningful models that satisfy specific business queries and requirements.

The models in the Intermediate and Marts folder usually need to reference the models in the Staging model. For this, DBT provides the `ref()` function, that receives the file name of a model as its parameter. The usage of `ref()` contributes to the idea of the modularity of the models, as developers will not need to write the same SQL code over and over again, resulting in redundant and dirty code practice. By default, the transformed models will be materialized as views, and will be updated after each transformation run. As for the configuration, `dbt-postgres` accepts almost the same values as `target-postgres` in the subsection 6.5.3.

```
WITH monthly_revenue AS (
  SELECT
    extract(year from transaction_date) as transaction_year,
    extract(month from transaction_date) as transaction_month,
    sum(total_amount) as monthly_revenue
  FROM
    {{ref("stg_1__transactions")}}
  group by transaction_year, transaction_month
  order by transaction_year, transaction_month
)

SELECT * FROM monthly_revenue
```

Listing 6.7: An example of a DBT model in the intermediate folder. This model will be materialized in the PostgreSQL instance as a `monthly_revenue` view.

6.5.5 Execution

To execute the ELT process, Meltano provides the `run` or the `invoke` command, followed by the plugins, that are to be included. So, taking account of the previous subchapters, the complete ELT workflow command of the prototype will be `meltano run tap-spreadsheets-anywhere target-postgres dbt-postgres:run`. Meltano also allows developers to set frequently occurring jobs. For example, if the system requires the transformation to run on a daily basis. Since there will be no changes on the data source, the author will not be dwelling further on this matter.

```
09:53:29  Running with dbt=1.3.5
09:53:29  Found 13 models, 0 tests, 0 snapshots, 0 analyses, 290 macros, 0 operations, 0 seed files, 0 sources, 0 exposures, 0 metrics
09:53:29
09:53:30  Concurrency: 2 threads (target='dev')
09:53:30
09:53:30  1 of 13 START sql view model public.int_1__transactions_count_customer ..... [RUN]
09:53:30  2 of 13 START sql view model public.int_1__transactions_date ................ [RUN]
09:53:30  2 of 13 OK created sql view model public.int_1__transactions_date ........... [CREATE VIEW in 0.16s]
09:53:30  3 of 13 START sql view model public.stg_1__cities ........................... [RUN]
09:53:30  1 of 13 OK created sql view model public.int_1__transactions_count_customer .... [CREATE VIEW in 0.17s]
09:53:30  4 of 13 START sql view model public.stg_1__customers ........................ [RUN]
09:53:30  4 of 13 OK created sql view model public.stg_1__customers ................... [CREATE VIEW in 0.09s]
09:53:30  5 of 13 START sql view model public.stg_1__product_categories ............... [RUN]
09:53:30  3 of 13 OK created sql view model public.stg_1__cities ...................... [CREATE VIEW in 0.10s]
09:53:30  6 of 13 START sql view model public.stg_1__transactions ..................... [RUN]
09:53:30  5 of 13 OK created sql view model public.stg_1__product_categories ........... [CREATE VIEW in 0.09s]
09:53:30  7 of 13 START sql view model public.best_transactors_origins ................ [RUN]
09:53:30  6 of 13 OK created sql view model public.stg_1__transactions ................ [CREATE VIEW in 0.09s]
09:53:30  8 of 13 START sql view model public.customer_origins ........................ [RUN]
09:53:30  8 of 13 OK created sql view model public.customer_origins ................... [CREATE VIEW in 0.07s]
09:53:30  9 of 13 START sql view model public.annual_revenue .......................... [RUN]
09:53:30  7 of 13 OK created sql view model public.best_transactors_origins ........... [CREATE VIEW in 0.10s]
09:53:30  10 of 13 START sql view model public.country_customers ...................... [RUN]
09:53:30  9 of 13 OK created sql view model public.annual_revenue ..................... [CREATE VIEW in 0.09s]
09:53:30  11 of 13 START sql view model public.country_revenue ........................ [RUN]
09:53:30  10 of 13 OK created sql view model public.country_customers ................. [CREATE VIEW in 0.08s]
09:53:30  12 of 13 START sql view model public.int_1__monthly_revenue ................. [RUN]
09:53:30  12 of 13 OK created sql view model public.int_1__monthly_revenue ............ [CREATE VIEW in 0.08s]
09:53:30  11 of 13 OK created sql view model public.country_revenue ................... [CREATE VIEW in 0.09s]
09:53:30  13 of 13 START sql view model public.best_month_revenue ..................... [RUN]
09:53:30  13 of 13 OK created sql view model public.best_month_revenue ................ [CREATE VIEW in 0.06s]
09:53:30
09:53:30  Finished running 13 view models in 0 hours 0 minutes and 0.97 seconds (0.97s).
09:53:30
09:53:30  Completed successfully
09:53:30
09:53:30  Done. PASS=13 WARN=0 ERROR=0 SKIP=0 TOTAL=13
```

Figure 6.10: A successful run of the DBT transformation process in the server.

6.6 Data Visualisation

After the data processing has been initially done, i.g. there are some generated views in the database, the transformed data are then to be visualized in the PWA. As stated in the section 5.3, a request of a view will trigger the server to refer the `cb_view` table to pass the data back as a response. This section elaborates how data are visualized in the prototype, specifically in the PWA, and which diagrams are used to support further analyzations and decision-making processes.

6.6.1 Diagrams

Big Number

Big numbers in dashboards attract the attention of users, because they contain mostly important key numbers of a dataset. The big number chart in the prototype shows aggregated numerical data. Aggregated data are collective data, through the utility of operators such as summations and averages. An example is shown in the Figure 6.11, where the total revenue of the sampled retail data over the years are added. Optionally, the big number chart also shows the trend of the data.

Figure 6.11: The big number chart of the prototype.

Bar Diagram

Bar charts/ diagrams show the comparison of various data categories, and can also subtly show a trend, based on the same value that those categories possess. In the Figure 6.12, the revenue of each year of the sampled dataset is projected with a bar diagram. Yet in this case, the trend of the dataset is not easily interpreted, as the dataset contain incomplete data from the year 2014.

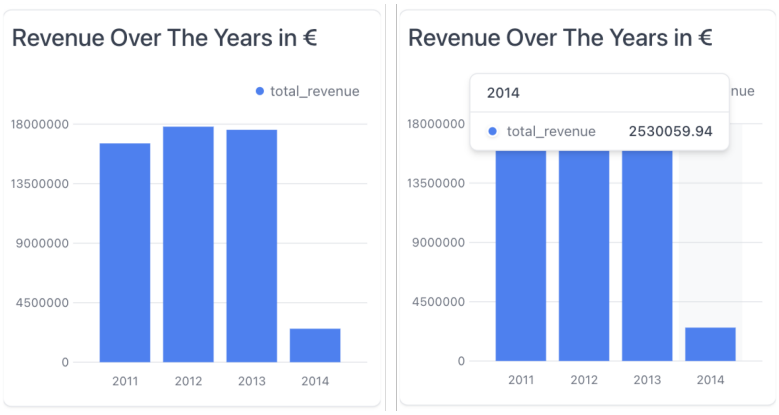

Figure 6.12: The bar diagram of the prototype.

Line Diagram

Line diagrams/ charts are suited to project timeline data and its fluctuation over time. In the prototype, the line diagram can also show the timeline data of various categories. The line diagram in Figure 6.13 for example, shows the revenue of each country over time. The categories can then also be filtered independently, to help to improve the data visualization.

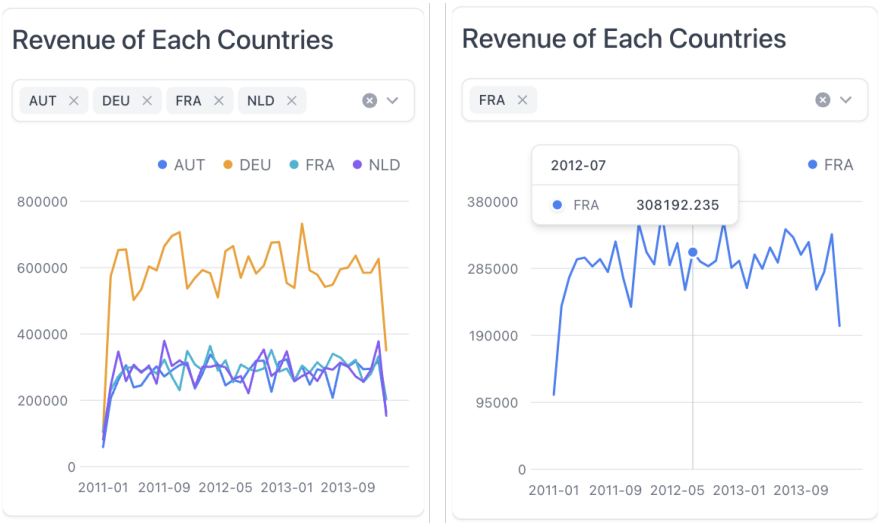

Figure 6.13: The line diagram of the prototype.

Donut Diagram

As a relative to the pie chart, the donut diagram/ chart shows the part of each category as percentages. Like the line diagram, the donut diagram in the prototype comes with a filter option.

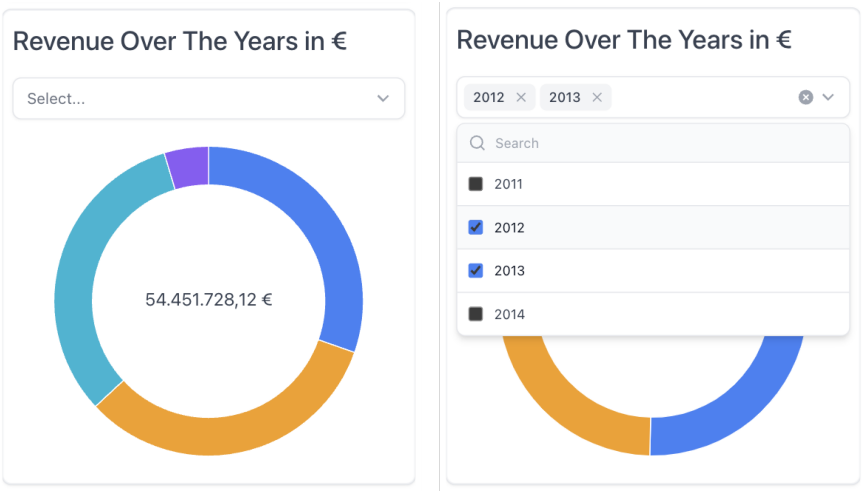

Figure 6.14: The donut diagram of the prototype.

Map Diagram

The visualization of geological representation as a map diagram is also included in the prototype. For this diagram, the author made use of the `react-simple-maps` library, and provided a static file `topojson.json` for its source of longitude and latitude. Because of the limited resources, the map works currently at country level.

Figure 6.15: The donut diagram of the prototype.

6.6.2 Global Time Filter

To enhance the visualization, specifically in the time dimension, the prototype is complemented with a global time filter. By specifying the *From Date* and the *To Date* with a native date picker, the filter will apply to all the diagrams available, containing filterable timeline data. Is this the case, then the data of the diagrams will be updated and refreshed, which means, that the PWA will send some requests to the backend with the date filter as query parameter of the requests.

```
https://api.crossbi.de/crossbi/v1/api/views/inspect/2?from=2012-09-16
&to=2023-09-17
```

Figure 6.16: An example of a time filtered request.

That said, due to the infinite combinations of such requests, it is impossible to cache them in the device's storage. This implies, that the filter only works within a working internet connection. The steps of how the global time filter works, can be briefly seen in the Figure 6.17.

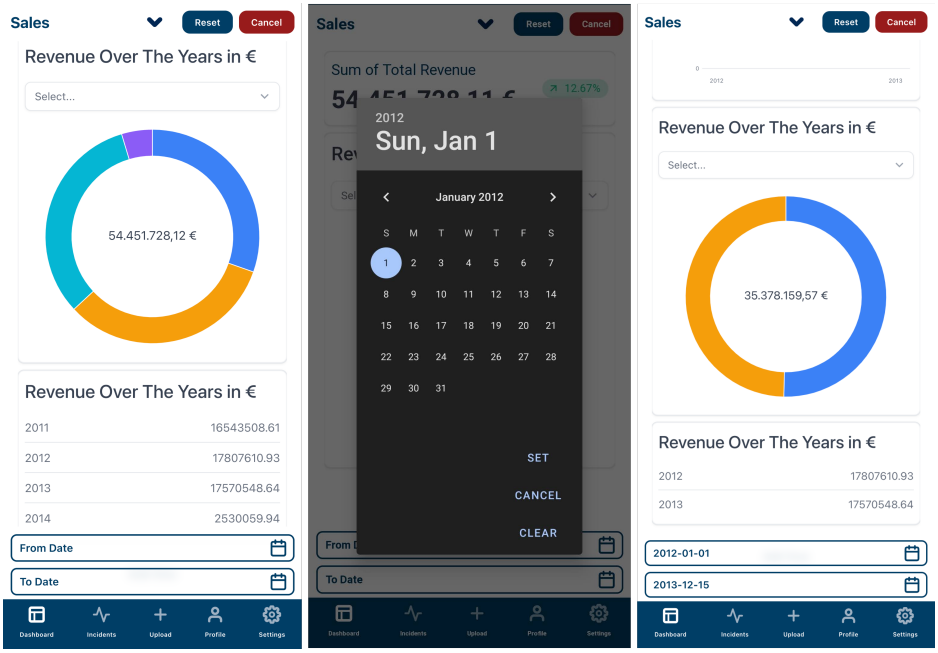

Figure 6.17: The dashboard before (left) and after (right) the global time filter is applied.

6.7 Notification

The notification feature in the web development could be achieved through the use of the Notifications API and the Push API. The Notifications API allows a web page to have a control over the system notification, whereas the Push API grants the ability to receive push messages from a server asynchronously. This feature of the web bridges the gap between web and native applications, as it increases the engagement of the application to users, and it is an implementation of a real-time data visualization. For a working web push notification system, both in computer or mobile devices, some requirements have to be fulfilled to achieve a secure context and environment.

- **A Service Worker.** The SW listens to the `push` event that is generated by the server, and triggers the notification to be shown on the respected device.
- **A TLS connection.** The web push only works on HTTPS connection. This guarantees the security context of the application upon using the feature.
- **Voluntary Application Server Identification (VAPID) keys.** The existence of VAPID keys acts as an authentication mechanism between the two sides of an application, and helps the server to identify its web push receiver and vice versa.

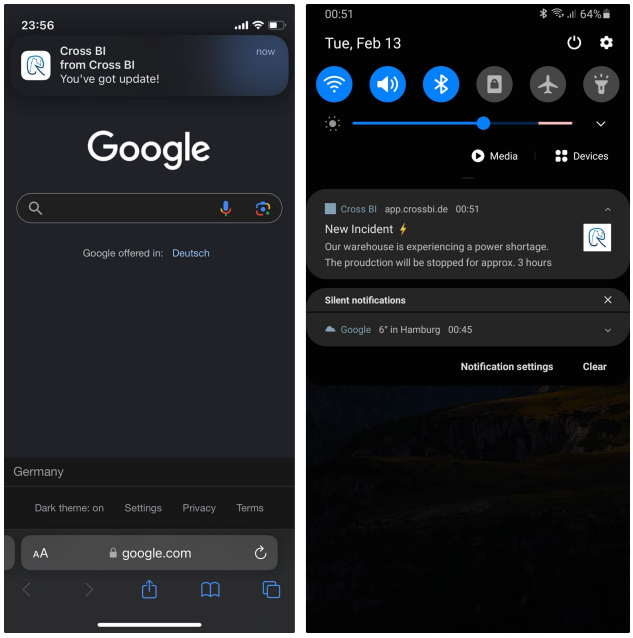

Figure 6.18: The push notification on iOS and on Android.

In the prototype, upon a successful first login attempt, users will be asked about the permission of allowing notification through the prototype. If allowed, the SW will send a subscription request and its details to the server. The details contain the information about the endpoint of the subscription and the public key (VAPID), to then later be paired with the private key in the server. This information will be stored in the `cb_pushsubscription` table from the Figure 5.3. A use case example would be, if an incident has occurred, then an incident object is created and persisted in the respected table. This will trigger the server to send notifications to all the subscribed endpoints. Naturally, the notification will then only appear, if the PWA is working under a working internet connection.

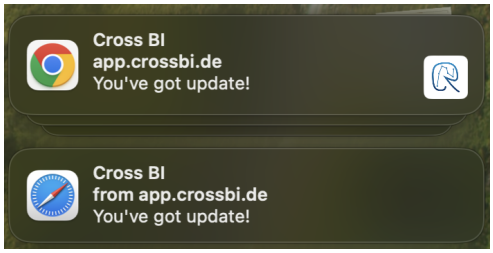

Figure 6.19: The notifications in desktop (macOS).

As seen in the Listing 6.8, the SW in the PWA listens to the `push` event. The payload of the event will be firstly parsed into a JSON object. Then, the `waitUntil()` method tells the browser, that a work is ongoing and therefore not to terminate the process occurring in the SW. Required are the `title` and the `body` properties, which are sent from the server as a payload, and an icon. The icon should ideally be the icon of the application itself. The `vibrate` and `badge` properties are included as experimental complement features that temporarily only work in Chromium browsers.

```
1  ...
2  self.addEventListener('push', (event) => {
3    const pushData = event.data.text();
4    const data = JSON.parse(pushData);
5
6    event.waitUntil(
7      self.registration.showNotification(data.title, {
8        body: data.body,
9        icon: '/pwa-192x192.png',
10       vibrate: [200, 100, 200, 100, 200, 100, 200],
11       badge: '/pwa-192x192.png'
12     })
```

```
13     );
14 });
15 ...
```

Listing 6.8: A push event listener in the SW.

6.8 Offline Mode

The PWA generally detects the absence of an internet connection. In this case, an offline indicator in the menu bar will appear, so that users will know if they currently do not have a proper internet connection. The visualisations will still be able to be seen, as the recent data of the charts are cached in the device's storage. As explained in the subsection 6.3.2, failed `fetch` requests will be directed to their recent responses in the cache storage. In conclusion, all precached and cached assets, including images and icons, will appear normally offline as it is. In the offline mode, users can upload a file nor make a change in the charts, as it will fire a POST request, that is ignored by the caching mechanism.

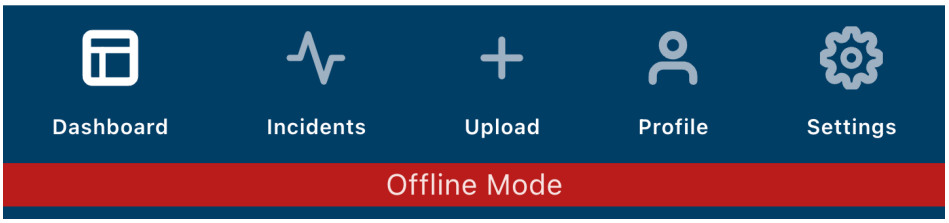

Figure 6.20: The offline mode indicator below the navigation bar of the PWA

7 Discussion

This chapter gives a discussion about how the prototype provide answers to the research questions in the section 1.2. Additionally, readers will find a list of the implementations to each functional and non-functional requirement in the chapter 4.

7.1 Research Questions Evaluation

To answer the first research question (**RQ1**), the author implemented and deployed a prototype consisting of a client side application, which is the PWA itself, and a server-side application to support the data processing activities, and the communication between the client and the server through API services. Upon using modern web technologies such as React and Vite, which are widely used and are proven to show high performance many aspects of software development [21], the author is able to deliver some of the most important diagram and chart types in the PWA to support further analysis of business data and the decision-making process. The PWA is also occupied with notable features of a common native application, resulting in a slight difference between the PWA and other native applications. It is also worth mentioning, that the VitePWA library also helps to make the development of the manifest and service workers a better experience for developers. Furthermore, because the PWA relies significantly more on browsers instead of OSs, users can install the PWA on various devices that have the common OSs. In conclusion, the PWA can be classified as a cross-platform application, which therefore answers the first research question.

There are few arguments based on the prototype and its system architecture, that can be used to give some insights for the second research question (**RQ2**).

- The PWA possess only one code base, that fits for all platforms, which, considering the normal native application development that consists of at least two code bases for Windows, Android, and Apple OSs. This means that the development of PWA

cuts down for at least 50% of the development complexity, implying that it will need less resources.

- Instead of being deployed in the application store of each OS, the PWA could be conveniently deployed over the web. This automatically reduces the complexity of e.g. the DevOps activities in an agile environment, resulting in the need for even less resources.

- ELT is a relatively new data processing discipline, that is meant to be used on par with modern data warehouse technologies. To compare the performance of both methods, Ranjan [32] carried out three data push down experiments, resulting in a performance gain for the ELT. Also, in a qualitative study by Haryono et al. [16], ELT is suggested to have the advantage of being a low-cost alternative than ETL. However, the results of the recent studies mentioned should be taken with a grain of salt, as a consequence of the lack of a standard benchmark for both disciplines [46].

Therefore, the second research questions can not yet be fully answered through this thesis due to the limitations that the author had at the time this work is being written. More limitations of the prototype can be found in the chapter 8.

7.2 Requirements Fulfilment

This section will discuss, whether the prototype as a whole does fulfil each requirement in the chapter 4.

Requirement	**Interpretation**
FR1	This requirement is fulfilled by installing SSL certificates on both client- and server-side of the prototype. Not only is this required for the PWA to be installable, but also it guarantees an encryption between data exchange through the API. Furthermore, the prototype is implemented with user registration system with encrypted passwords, to add to the security aspects.

FR2	The Tremor library is used on top of React to support the DV mechanism. The dashboard feature shows common diagrams and charts, that are useful for the decision-making process and further analysis.
FR3	With the existence of the manifest, service workers and HTTPS connections, the PWA is ready to installed from any commonly used browsers on any common OS.
FR4	The PWA is equipped with the web push feature that allows the display of notifications for updates or incidents. Upon logging-in, users will need to allow the notification permission to get any notification onwards. In this way, the prototype only allows registered users to use its notification service.
FR5	Users can upload some sort of data source in form of CSV or Excel (`xlsx`) file on the upload section of the PWA.
FR6	The server side of the prototype could extract, load and transform (ELT) the available (business) data into views. The transformation process occurs in the data warehouse, which in the case of the prototype is the PostgreSQL RDBMS instance.
FR7	Some charts and diagrams on the client-side are equipped with the filter function. Additionally, the PWA allows users to filter the time range of the charts globally.
FR8	User interactions could trigger the PWA to briefly show a UI element called toasts. The toasts act as success and error feedback, maintaining a good UX overall.

FR9	Logged-in users can still open the PWA even without having an internet connection. This happens because the installed service worker will cache all static files and responses to the latest update. However, since this is the case for the PWA, users have to initially navigate to all available pages to be able to get the cached responses into the offline mode. Also, in the offline mode, users cannot make changes to the application. If there is no network, users will also be notified in the PWA.
NFR1	In the settings section, users will find a small guide to install the PWA. In addition, when using the Google Chrome browser, users will be prompted to install the PWA, so that it can also be done automatically.
NFR2	Other than applying all the best practices of the PWA development, some styling decisions using the CSS are made. As suggested in a quantitative study about PWA by Diekmann et al. [6], the PWA of the prototype uses an explicit navigation element to avoid navigation dead end. Additionally, the PWA also uses the device's system font to keep the appearance of the PWA to be as familiar as possible.

Table 7.1: Requirements Fulfilment

8 Conclusion

8.1 Conclusion

It is an importance for companies in the current competitive economy, to be able to monitor, analyse and improve the company's performance, which could be done by adopting BI software, that integrates and visualizes the (business) data of the companies, such as sales, customer movement and production data. The visual representation of the data will then later be analysed for further strategy decisions and innovations. This bachelor thesis challenges the idea of presenting the DV aspect of BI, by prototyping a technology alternative towards native applications called the PWA. The development of PWAs is less complicated than the development of native applications, due to the fact that it consists of one code base only. This advantage could benefit not only small and medium companies, but also bigger enterprises that want to cut expenses.

The thesis starts by handing off an introduction to the problem, describing the research methodology, and formulating the research questions. Then, an analysis of a related work, that the author found to be relevant and helpful for this thesis, is provided. Next, deriving from the theoretical background, the requirements are defined and divided into functional and non-functional requirements. The thesis then proceeds with the system design and architecture, and the result of the implementation with some screenshots of the prototype in use.

The prototype itself is a client-server system, that utilizes the HTTP protocol and an SSL certificate to communicate with each other. The server consists of a PostgreSQL database/ data warehouse, a flask application for managing communication, and the Meltano framework for the data processing, which implements the ELT discipline as its workflow. The PWA on the other hand, lies in the client side of the prototype. The PWA is built on top of the widely used React library and the VitePWA, a Vite plugin that helps to manage the development process of the PWA.

As a result, the prototype could provide a full answer for the first research question (**RQ1**). The PWA is proven to be able to act as a standalone application on various devices, and to visualise business data with heterogeneous charts and diagrams. However, although the prototype could give some positive insights for the second research question (**RQ2**), a quantitative study with larger resources can be conducted to gain the confidence level of the answer.

8.2 Limitations and Future Research

While the results of the implementation leave positive remarks on the research questions, this bachelor thesis imposes some limitations and potential future researches, that can be done based on this work. The first limitation is the subjectivity of the UX towards the prototype, as the evaluation relied on primarily self-evaluated data and previous findings. For that, a thorough quantitative study could be done to boost the confidence of the findings. Another limitation was, that the system infrastructure of the prototype was funded by the author because of the absence of technical resources from the academic institution. This was caused by a technical incident that occurred while this thesis was being written. Thus, a proceeding research that tests the prototype on a bigger scale could produce a more comprehensive result. Another recommendations on a related future research that focus on the infrastructure of the prototype, can also be the performance study on an N-Tier architecture instead of a two-tier architecture, which enhances the performance of the prototype, by focusing on the paradigm of the separations of concern (SOC). A comparative performance study between the implementation of the PWA on other front-end libraries and frameworks, e.g. Vue, Svelte, Angular etc, could also be useful to determine a better choice for the front-end technology [7]. This thesis and the prototype are also limited to set its focus on the data visualisation aspect of BI, which sets aside the other disciplines such as data mining and Customer Relationship Management (CRM) Marketing as seen in the Figure 3.2, which could also act as a base for forthcoming studies in combination with the ELT discipline.

Another limitation of the PWA that troubles the author, is that, to be able to see the charts and diagrams, users have to open all available pages, so that the responses are then cached by the PWA, which is a bit unpractical. It is safe to say, that this is also a problem for some PWAs out in the web. Further research on a better caching strategy will be a major resolve for this issue.

All in all, PWA is a rapidly growing ecosystem with advancements in technologies and user expectations. Future researches will always be necessary to monitor the evolving environment of PWAs and how they influence user experience. By addressing these limitations, the author encourages further exploration and development of this technology implementation proposal.

9 Links

1. **PWA:** https://app.crossbi.de
2. **API Endpoint:** https://api.crossbi.de
3. **Front End Repository:** https://github.com/dvdkwei/cross-bi-frontend
4. **Back End Repository:** https://github.com/dvdkwei/cross-bi-backend
5. **Data Source:** https://www.kaggle.com/datasets/amark720/retail-shop-case-study-dataset

Bibliography

[1] Beaufort, François ; LePage, Pete ; Steiner, Thomas ; Rodionov, Alexey: *Add a web app manifest | Articles | web.dev*. https://web.dev/articles/add-manifest. – (Accessed on 01/21/2024)

[2] Beaufort, François ; LePage, Pete ; Steiner, Thomas ; Rodionov, Alexey: *Add a web app manifest | Articles | web.dev*. https://web.dev/articles/add-manifest#splash-screen. – (Accessed on 02/28/2024)

[3] Biørn-Hansen, Andreas ; Majchrzak, Tim A. ; Grønli, Tor-Morten: Progressive Web Apps: The Possible Web-native Unifier for Mobile Development. In: *Proceedings of the 13th International Conference on Web Information Systems and Technologies (WEBIST 2017)* (2017), 1, S. 344–351. – URL https://doi.org/10.5220/0006353703440351

[4] Buitrón-Dámaso, I. ; Morales-Luna, G.: HTTPS connections over Android. In: *2011 8th International Conference on Electrical Engineering, Computing Science and Automatic Control*, 2011, S. 1–4

[5] Castilho, Bernardo: *Building a Business Intelligence Progressive Web Application - CodeProject*. https://www.codeproject.com/Articles/1257952/Building-a-Business-Intelligence-Progressive-Web-2. 8 2018. – (Accessed on 10/21/2023)

[6] "Diekmann, Julian ; Eggert, Mathias": *"Is a Progressive Web App an Alternative for Native App Development? - A prototype-based usability evaluation of a health insurance app"*. "3. Wissenschaftsforum: Digitale Transformation (WiFo21)". 2021

[7] Diniz-Junior, Raimundo N. ; Figueiredo, Caio César L. ; De S.Russo, Gilson ; Bahiense-Junior, Marcos Roberto G. ; Arbex, Mateus V. ; Dos Santos, Lanier M. ; Da Rocha, Raimundo F. ; Bezerra, Renan R. ; Giuntini, Felipe T.:

Evaluating the performance of web rendering technologies based on JavaScript: Angular, React, and Vue. In: *2022 XVLIII Latin American Computer Conference (CLEI)*, 2022, S. 1–9

[8] Duda, Jerzy: Business intelligence and NoSQL databases. In: *Information Systems Management* 1 (2012), 1, Nr. 1, S. 25–37. – URL http://cejsh.icm.edu.pl/cejsh/element/bwmeta1.element.desklight-18861c48-494a-457a-b06a-77e9478aed49/c/ISIM_Vol_1_1__25-37.pdf

[9] Elias, Micheline: *Enhancing User Interaction with Business Intelligence Dashboards*, Ecole Centrale Paris, Dissertation, 10 2012. – URL https://theses.hal.science/tel-00969170

[10] Fielding, Roy T.: *Fielding Dissertation: CHAPTER 5: Representational State Transfer (REST)*. 2000. – URL https://ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[11] Fournier, Camille: *Comparison of smoothness in progressive web apps and mobile applications on Android*, KTH ROYAL INSTITUTE OF TECHNOLOGY, Diplomarbeit, 2020. – URL https://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1474729&dswid=7415

[12] Friendly, Michael ; Denis, Daniel J.: *Milestones in the history of thematic cartography, statistical graphics, and data visualization*. 2001. – URL http://www.datavis.ca/milestones/

[13] Gillis, Alexander S.: *What is native app? | Definition from TechTarget*. https://www.techtarget.com/searchsoftwarequality/definition/native-application-native-app. December 2022. – (Accessed on 10/18/2023)

[14] Government, The United S.: *The HTTPS-Only Standard - Introduction to HTTPS*. https://https.cio.gov/faq/. – (Accessed on 12/26/2023)

[15] Hajian, Majid: *Progressive Web Apps with Angular*. Apress, 1 2019. – URL https://doi.org/10.1007/978-1-4842-4448-7

[16] Haryono, Edward M. ; Fahmi ; Tri W, Adi S. ; Gunawan, Indra ; Nizar Hidayanto, Achmad ; Rahardja, Untung: Comparison of the E-LT vs ETL Method in Data Warehouse Implementation: A Qualitative Study. In: *2020 International*

Conference on Informatics, Multimedia, Cyber and Information System (ICIMCIS), 2020, S. 115–120

[17] Huang, Zhongdong ; Savita, K.S. ; Zhong-Jie, Jiang: The Business Intelligence impact on the financial performance of start-ups. In: *Information Processing and Management* 59 (2022), 1, Nr. 1, S. 102761. – URL https://doi.org/10.1016/j.ipm.2021.102761

[18] Booth, Christopher J. (Hrsg.): *The new IEEE standard dictionary of electrical and electronics terms : (including abstracts of all current IEEE standards) / Gediminas P. Kurpis, chair ; Christopher J. Booth, editor.* 5th ed. Institute of Electrical and Electronics Engineers, 1993. – ISBN 1559372400

[19] inc., MESCIUS: *JavaScript UI Components | Powerful UI Controls for Web Applications | Wijmo.* https://www.grapecity.com/wijmo?utm_source=CodeProject&utm_medium=Dynamic&utm_campaign=PWA-Article-August-2018. – (Accessed on 10/21/2023)

[20] javaatpoint: *2 Tier Architecture in DBMS - javatpoint.* https://www.javatpoint.com/2-tier-architecture-in-dbms. – (Accessed on 12/23/2023)

[21] Kaushalya, Thilanka ; Perera, Indika: Framework to Migrate AngularJS Based Legacy Web Application to React Component Architecture. In: *2021 Moratuwa Engineering Research Conference (MERCon)*, 2021, S. 693–698

[22] Kirk, Andy: *Data Visualisation.* SAGE Publications Limited, 9 2019

[23] Kordon, Fabrice ; Luqi: An introduction to rapid system prototyping. In: *IEEE Transactions on Software Engineering* 28 (2002), 9, Nr. 9, S. 817–821. – URL https://doi.org/10.1109/tse.2002.1033222

[24] Krombholz, Katharina ; Busse, Karoline ; Pfeffer, Katharina ; Smith, Matthew ; Zezschwitz, Emanuel von: "If HTTPS Were Secure, I Wouldn't Need 2FA" - End User and Administrator Mental Models of HTTPS. In: *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, S. 246–263

[25] Kukka, Hannu ; Goncalves, Jorge ; Heikkinen, Tommi ; Suua, Olli-Pekka ; Zuo, Yifei ; Raappana, Hannu ; Abdellatif, Mohamed ; Okkonen, Olli ; Jiménez, Raúl ; Ojala, Timo: Touch OK to Continue: Error Messages and

Affective Response on Interactive Public Displays. In: *Proceedings of the 4th International Symposium on Pervasive Displays*. New York, NY, USA : Association for Computing Machinery, 2015 (PerDis '15), S. 99–105. – URL https://doi.org/10.1145/2757710.2757723. – ISBN 9781450336086

[26] LePage, Pete ; Richard, Sam: *What are Progressive Web Apps? | Articles | web.dev*. https://web.dev/articles/what-are-pwas. 6 2020. – (Accessed on 10/20/2023)

[27] Masud, Luca ; Valsecchi, Francesca ; Ciuccarelli, Paolo ; Ricci, Donato ; Caviglia, Giorgio: From Data to Knowledge - Visualizations as Transformation Processes within the Data-Information-Knowledge Continuum, 07 2010, S. 445–449

[28] *410 Gone - HTTP | MDN*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/410. – (Accessed on 12/02/2023)

[29] *ServiceWorkerRegistration - Web APIs | MDN*. https://developer.mozilla.org/en-US/docs/Web/API/ServiceWorkerRegistration?retiredLocale=de. 1 2024. – (Accessed on 01/27/2024)

[30] Negash, Solomon: Business intelligence. In: *Communications of the Association for Information Systems* 13 (2004), 1. – URL https://doi.org/10.17705/1cais.01315

[31] Percival, Harry ; Gregory, Bob: *Architekturpatterns mit Python*. 7 2021

[32] Ranjan, Vikash: A Comparative Study between ETL (Extract-Transform-Load) and ELT (Extract-Load-Transform) approach for loading data into a Data Warehouse By, URL https://api.semanticscholar.org/CorpusID:10204240, 2009

[33] Richardson, Leonard: *JWTUMOIM: Act 3*. 2008. – URL https://www.crummy.com/writing/speaking/2008-QCon/act3.html

[34] Ríkharðsson, Páll ; Yigitbasioglu, Ogan: Business intelligence I& analytics in management accounting research: Status and future focus. In: *International Journal of Accounting Information Systems* 29 (2018), 6, S. 37–58. – URL https://doi.org/10.1016/j.accinf.2018.03.001

[35] Sahu, Naimish: *Software Prototyping Model and Phases - GeeksforGeeks*. https://www.geeksforgeeks.org/software-prototyping-model-and-phases/. – (Accessed on 12/09/2023)

[36] Sharp, Helen ; Preece, Jennifer ; Rogers, Yvonne: *Interaction design*. John Wiley & Sons, 5 2019

[37] Shneiderman, B.: The eyes have it: a task by data type taxonomy for information visualizations. In: *Proceedings 1996 IEEE Symposium on Visual Languages*, 1996, S. 336–343

[38] Shneiderman, Ben: *Designing the User Interface: Strategies for Effective Human-Computer Interaction*. 3rd. USA : Addison-Wesley Longman Publishing Co., Inc., 1997. – ISBN 0201694972

[39] StatCounter: *Browser Market Share Worldwide | Statcounter Global Stats*. https://gs.statcounter.com/browser-market-share#monthly-202001-202401. – (Accessed on 01/29/2024)

[40] StatCounter: *Mobile Vendor Market Share Worldwide | Statcounter Global Stats*. https://gs.statcounter.com/vendor-market-share/mobile#monthly-202001-202401. – (Accessed on 01/29/2024)

[41] StatCounter: *Operating System Market Share Worldwide | Statcounter Global Stats*. https://gs.statcounter.com/os-market-share#monthly-202001-202401. – (Accessed on 02/28/2024)

[42] Statista: *Business Intelligence Software - Global | Market Forecast*. – URL https://www.statista.com/outlook/tmo/software/enterprise-software/business-intelligence-software/worldwide

[43] Taivalsaari, Antero ; Mikkonen, Tommi: The Web as an Application Platform: The Saga Continues. In: *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications*, 2011, S. 170–174

[44] Taivalsaari, Antero ; Mikkonen, Tommi ; Ingalls, Dan ; Palacz, Krzysztof: Web Browser as an Application Platform. In: *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, 2008, S. 293–302

[45] W3schools: *JavaScript HTML DOM*. https://www.w3schools.com/js/js_htmldom.asp. – (Accessed on 01/06/2024)

[46] Waas, Florian ; Wrembel, Robert ; Freudenreich, Tobias ; Thiele, Maik ; Koncilia, Christian ; Furtado, Pedro: On-Demand ELT Architecture for Right-Time BI: Extending the Vision. In: *International Journal of Data Warehousing and Mining* 9 (2013), 04, S. 21–38

[47] Wijmo: *MyBI | Business Intelligence PWA with Wijmo*. – URL https://demos.wijmo.com/5/PureJS/MyBI/MyBI/

[48] Zheng, Jack: *Data visualization in business intelligence*. Taylor & Francis, 11 2017. – 67–81 S. – URL https://doi.org/10.4324/9781315471136-6

A Anhang

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original