

Oktober 2022

Programmieren einer Web-App für Teamverwaltung und Projektplanung

Maturaarbeit von Rafael Urben

Betreut durch Lukas Gerber

gym | BURGDORF

ABSTRACT

Im Rahmen dieser Arbeit wurde eine Web-App entwickelt, in welcher kleine Teams oder Einzelpersonen ihre Arbeitszeit erfassen, gemeinsame Kalender verwalten sowie To-do-Listen erstellen können. Dazu wurden unter anderem Sprachen und Frameworks wie Python, JavaScript, HTML, CSS, Django, React und Bootstrap verwendet. In dieser Arbeit werden ausserdem gewisse Technologien und Begriffe rund um Webseiten unter die Lupe genommen.

Die erstellte Web-App kann unter <https://app.rafaelurben.ch/teamized/> betrachtet und verwendet werden.

VORWORT

Da ich mich schon seit einigen Jahren für Informatik interessiere war für mich klar, dass meine Maturaarbeit etwas mit Informatik zu tun haben muss. Auch sollte es ein frei zugängliches Projekt sein, welches anderen Leuten etwas dienen könnte. Vor diesem Projekt hatte ich bereits einige Webseiten und Web-Apps gebaut, jedoch waren die meisten davon entweder schon wieder veraltet oder entsprachen nicht mehr meinen Vorstellungen einer «guten» Webseite/Web-App. Deshalb wollte ich eine Web-App entwickeln, welche in vielen Hinsichten moderner ist als alle meine bisherigen Web-Apps und einen soliden Funktionsumfang beinhaltet. Mit diesem Projekt wollte ich auch React als für mich neues Framework kennenlernen und dieses zusammen mit dem mir bereits bekannten Framework Django kombinieren. Ebenfalls ist es mir ein Anliegen, diese Arbeit für so viele Leute wie möglich zugänglich zu machen und daher gewisse Teile auch für Laien verständlich darzustellen.

INHALTSVERZEICHNIS

| | |
|---|----|
| Abstract | 2 |
| Vorwort | 2 |
| Inhaltsverzeichnis | 3 |
| 1 Einleitung..... | 5 |
| 1.1 Marktanalyse und Zielsetzung..... | 5 |
| 1.2 Vorgehen | 5 |
| 2 Theorie..... | 5 |
| 2.1 Webseiten – Die Grundlagen | 6 |
| 2.1.1 Client und Server | 6 |
| 2.1.2 Requests und Protokolle | 6 |
| 2.1.3 Domains und URLs..... | 8 |
| 2.1.4 Dynamische und statische Webseiten..... | 9 |
| 2.2 Native Apps und Web-Apps | 10 |
| 2.2.1 Native App..... | 10 |
| 2.2.2 Web-App | 10 |
| 2.2.3 Die Probleme..... | 10 |
| 2.2.4 PWAs - die Kombination..... | 11 |
| 2.3 Der Aufbau einer Webseite | 11 |
| 2.3.1 Backend..... | 11 |
| 2.3.2 Frontend | 11 |
| 3 Durchführung..... | 16 |
| 3.1 Der App-Name..... | 16 |
| 3.2 App-Spezifikationen definieren..... | 16 |
| 3.3 Auswählen der Software & Tools | 17 |
| 3.3.1 Backend..... | 17 |

| | | |
|---|--|----|
| 3.3.2 | Frontend | 17 |
| 3.3.3 | Hosting | 18 |
| 3.3.4 | Programme zur Entwicklung | 19 |
| 3.4 | Entwicklungsumgebung und Projekt vorbereiten | 19 |
| 3.4.1 | Installationen | 20 |
| 3.4.2 | Ordnerstruktur | 20 |
| 3.4.3 | JSX-Kompilierung mit Babel.js | 22 |
| 3.5 | Coding | 23 |
| 3.5.1 | Umsetzung Authentifizierung | 23 |
| 3.5.2 | Umsetzung Backend | 23 |
| 3.5.3 | Umsetzung Frontend | 26 |
| 3.6 | Testing | 29 |
| 4 | Endprodukt | 30 |
| 5 | Diskussion | 30 |
| 5.1 | Ausblick | 30 |
| 6 | Verzeichnisse | 31 |
| 6.1 | Abbildungen | 31 |
| 6.2 | Tabellen | 31 |
| 7 | Eigenständigkeitserklärung | 32 |
| 8 | Schlusswort | 32 |
| Anhang A: Sprachen – Erweiterte Details | | 33 |
| A.1 | HTML | 33 |
| A.2 | CSS | 35 |
| A.3 | JavaScript | 36 |
| Anhang B: Screenshots der Web-App | | 37 |

1 EINLEITUNG

Aus Gründen der Einfachheit wird in dieser Arbeit grundsätzlich das generische Maskulin verwendet. Falls anwendbar bezieht sich diese Bezeichnung selbstverständlich auf alle Geschlechter.

1.1 MARKTANALYSE UND ZIELSETZUNG

Im Internet gibt es viele Tools, welche die Zusammenarbeit in kleinen Teams erleichtern. Darunter ist einer der grösseren Namen Trello ¹, eine Web-App für To-do-Listen. Viele dieser Tools haben jedoch entweder das Problem, dass wichtige Funktionen nicht mehr kostenlos sind oder dass die App nur auf etwas spezialisiert ist. Ziel dieser Arbeit ist es daher, eine kostenlose App zu entwickeln, welches die wichtigsten Funktionen diverser Tools in einer einzigen Toolbox vereint.

1.2 VORGEHEN

Um dieses Ziel zu erreichen, wird in dieser Arbeit mit sowohl altbekannten als auch neueren Sprachen, Tools und Frameworks von Grund auf eine Web-App gebaut und diese im Internet veröffentlicht.

2 THEORIE

Webseiten sind für einen grossen Teil der Menschheit etwas, mit dem sie täglich konfrontiert werden. Doch viele wissen eigentlich gar nicht, worum es sich dabei handelt und wie sie funktionieren. In diesem Theorieteil wird deshalb die Funktionsweise einer Webseite für sowohl erfahrene als auch unerfahrene Benutzer von Grund auf erarbeitet. Technischere Beschreibungen und verwendete Software folgen im Kapitel Durchführung.

¹ <https://trello.com/> (Abgerufen am 23.10.2022)

2.1 WEBSEITEN – DIE GRUNDLAGEN

Bei einfachen Webseiten handelt es sich vereinfacht gesagt um eine Sammlung von Textdateien, welche irgendwo auf dieser Welt gespeichert sind und über das Internet der ganzen Welt zur Verfügung gestellt («gehostet») werden. Webseiten können aber, wie es bei jeglichen anderen Dateien der Fall ist, auch lokal gespeichert werden. Dann entfällt jedoch nur die Reise um die halbe Welt, die Funktionsweise bleibt gleich: Die Webseite wird nach Abrufen im Browser (auch «web browser» oder «internet browser» genannt) dem Benutzer angezeigt. Der Browser ist für das «Interpretieren» und Ausführen des Website-Codes sowie für alle Benutzerinteraktionen zuständig.

In diesem Abschnitt folgen einige Begriffserklärungen, welche für das bessere Verständnis von Webseiten und dem Internet erforderlich sind:

2.1.1 Client und Server

Im Bereich der Informatik wird häufig von Clients (*engl.* «client»: Kunde) und Servern (*engl.* «server»: Diener) gesprochen. Als Client wird dabei der Rechner bezeichnet, welcher eine Anfrage lokal oder über das Internet an einen Server stellt. Der Server ist dementsprechend der Rechner, welche die Anfrage erhält und bearbeitet. Das Wort «Rechner» steht hierbei für das Gerät, welches für die Anfrage zuständig ist. Es kann sich dabei also um einen Personal Computer (PC) handeln, kann aber auch für ein Smartphone, eine Smartwatch oder sogar einen intelligenten Kühlschrank stehen. Dies gilt nicht nur für den Client, sondern auch für den Server. Denn alle Geräte, welche sich mit einem Netzwerk verbinden können, können sowohl als Client als auch Server agieren. Bei Servern selbst handelt es sich meistens jedoch um grössere Anlagen, welche dazu ausgelegt sind, Millionen von Anfragen in der Minute zu bearbeiten. Von den Komponenten her betrachtet sind diese jedoch einem Computer bis auf wenige Details sehr ähnlich. Je nach Kontext kann das Wort Client auch für das spezifische Programm stehen, welches die Anfrage ausführt. Für den Menschen hinter dem Gerät wird eher das Wort Benutzer (*engl.* «user») verwendet.

2.1.2 Requests und Protokolle

Jedes Mal, wenn eine Webseite geöffnet wird oder auf eine neue Seite navigiert wird, löst der Browser eine Anfrage (*engl.* «request») aus. Diese wird über HTTP(S) an den Server

gesendet, welcher mit einer Antwort (*engl.* «response») reagiert. HTTP steht hierbei für das «Hypertext Transfer Protocol». Das angehängte «S» steht für «secure», also sicher. Diese beiden Protokolle sieht man häufig in der Adressleiste des Browsers und bei Links, da es sich dabei um das für Webseiten verwendete Protokoll handelt. Ähnlich wie eine Dateiendung gibt das Protokoll an, wie die Daten formatiert sind, damit der Server weiss, welches Programm die Anfrage wie zu bearbeiten hat. Es gibt auch viele andere Protokolle, darunter FTP («File Transfer Protocol») oder SSH («Secure Shell»). Grundsätzlich kann jede Applikation ihr eigenes Protokoll verwenden, sowohl über das Internet als auch lokal. Die Einstellungen auf Windows-Geräten zum Beispiel können mit dem MS-SETTINGS-Protokoll angesprochen werden. Die Seite für Windows-Updates kann direkt mit [ms-settings:windowsupdate](#)² geöffnet werden.

2.1.2.1 Getäuscht durch HTTPS

Im Zusammenhang mit HTTPS gibt es im Internet und in der Gesellschaft sehr gefährliches Halbwissen. Obwohl das «s» für sicher steht, heisst dies nicht, dass eine Webseite auch sicher ist. Dies bedeutet nur, dass die Verbindung vom Client zum Server verschlüsselt ist. Jede und jeder kann heutzutage ein sogenanntes SSL-Zertifikat («Secure Socket Layer») erhalten und somit eine verschlüsselte Verbindung zum eigenen Server ermöglichen. Auch wenn eine Webseite also ein Schloss-Symbol in der Adressleiste hat, kann sie von einem Betrüger (*engl.* «scammer») stammen, der an Daten der Benutzer gelangen will. Wer ein wenig sicherer gehen will, kann in den gängigen Browsern via Klick auf das Schloss-Symbol weitere Informationen über das verwendete Zertifikat erhalten, z. B. wann, von wem und für wen es ausgestellt wurde.

2.1.2.2 Verunsichert durch HTTP

HTTP ist seit dem Aufkommen von HTTPS aufgrund seiner mangelnden Verschlüsselung aus der Mode gekommen. Da die Verbindung zu einer Webseite über HTTP nicht verschlüsselt ist, kann die Anfrage auf dem Weg zum Server über eine Man-in-the-middle-Attacke abgefangen und mitgelesen werden. Bei HTTPS ist dies dank der Verschlüsselung nicht mehr der Fall. Passwörter und sensible Daten sollten deshalb nie über HTTP

² <https://docs.microsoft.com/en-us/windows/uwp/launch-resume/launch-settings-app> (Abgerufen am 23.10.2022)

übertragen werden. Scheinbar seriöse Webseiten über HTTP sollten daher mit viel Misstrauen betrachtet werden.

2.1.3 Domains und URLs

Wenn eine Webseite geöffnet werden soll, welche nicht auf demselben Gerät gespeichert ist, muss die Anfrage an den zuständigen Server gelangen. Hier kommt nun das World-Wide-Web (WWW; Internet) ins Spiel. Wer eine Webseite öffnen will, hat meistens etwas folgender Art: <https://www.youtube.com/watch?v=dQw4w9WgXcQ>.

Dies ist eine sogenannte URL («Uniform Resource Locator») und besteht aus verschiedenen Teilen. Der erste Teil, [https](https://), steht, wie wir bereits wissen, für das Protokoll. Der zweite Teil, www.youtube.com, ist die Domain und der restliche Teil ist der Pfad.

2.1.3.1 Domain

Die Domain ist der mehr oder weniger wichtigste Teil einer URL, denn Sie bestimmt indirekt den Zielservers. Sie besteht aus folgenden Teilen und wird von rechts nach links immer spezifischer:

- «Top-level Domain» (TLD)
- «Second-level Domain» (SLD)
- «Third-level Domain», «Fourth-level Domain» usw.

Die TLD-Endung gibt meistens Informationen darüber, in welchem Land sich eine Webseite «befindet» oder zu welcher Kategorie eine Webseite gehört. (z. B. [.ch](https://www.ch.ch), [.com](https://www.com.com), [.gov](https://www.gov.gov)) Bis auf einige Ausnahmen (z. B. [.swiss](https://www.swiss.ch) ³) haben TLDs grundsätzlich keine bestimmten Richtlinien, wer diese benutzen darf.

Die SLD ist der eigentliche Hauptteil der Domain und somit auch der Teil, der von Firmen und Privatpersonen gemietet wird. In unserem Beispiel wäre dies [youtube.com](https://www.youtube.com/). Bei mehrteiligen Domainendungen (z. B. [.co.uk](https://www.co.uk)) ist dementsprechend die «Third-level Domain» der Hauptteil und der Rest um eine Stufe verschoben. Die SLD ist auch der Teil,

³ <https://www.hostpoint.ch/domains/swiss.html> (Abgerufen am 23.10.2022)

welcher in nicht ganz vertrauenswürdigen E-Mails unbedingt immer beachtet werden muss, denn youtube.com.example.com hätte mit youtube.com nichts zu tun. Weitere Subdomains gehören jeweils zur entsprechenden Hauptdomain und sind auch im Besitz deren Inhaber. Für Webseiten wird üblicherweise die Subdomain [www.](https://www.youtube.com), also in unserem Beispiel www.youtube.com verwendet. Dies ist jedoch nicht zwingend so und wird eher als Erkennungsmerkmal verwendet.

2.1.3.2 Pfad

Der Pfad, also in unserem Beispiel [/watch?v=dQw4w9WgXcQ](https://www.youtube.com/watch?v=dQw4w9WgXcQ), gibt dem Server an, auf welche Ressource zugegriffen werden will. Bei einer sicheren Verbindung wird der Pfad auch verschlüsselt und bleibt somit für Zwischenstellen verborgen.

2.1.3.3 Anfrage

Um herauszufinden, welcher Server hinter einer Domain steckt, fragt das anfragende Gerät bei sogenannten DNS («Domain Name System») Servern nach. Diese geben dann eine IP-Adresse als Antwort. Diese identifiziert ein Gerät in einem Netzwerk eindeutig. An diese IP-Adresse wird schliesslich die HTTP(S) request gesendet.

[localhost](https://en.wikipedia.org/wiki/Localhost) ist ausserdem eine Pseudo-Domain, welche immer auf die spezielle IP-Adresse 127.0.0.1 zeigt, welche wiederum auf das aktuelle Gerät zurückverweist.

Informationen über die Funktionsweise von IP-Adressen können Wikipedia ⁴ entnommen werden.

2.1.4 Dynamische und statische Webseiten

Bei Webseiten gibt es eine wichtige Unterscheidung: Statische und dynamische Webseiten. Statische Webseiten ändern ihren Inhalt grundsätzlich nicht. Ein Client greift dabei fast direkt auf die Dateien zu, ohne dass ein Server diese zuerst modifiziert. Sie sind auch für alle Besucher identisch (Beispiel: Dokumentation eines Computerprogrammes). Ganz anders funktionieren dynamische Seiten: Hier modifiziert der Server die Webseite für jeden Benutzer individuell. Auch kann sich der Inhalt von Aufruf zu Aufruf unterscheiden. Jede Webseite, welchen Benutzern das Einloggen oder Interagieren mit

⁴ <https://de.wikipedia.org/wiki/IP-Adresse> (Abgerufen am 23.10.2022)

anderen Benutzern ermöglicht, ist somit sicher eine dynamische Webseite. Nur weil eine Webseite aber interaktiv ist, bedeutet dies nicht automatisch, dass sie auch dynamisch ist.

2.2 NATIVE APPS UND WEB-APPS

Native Applikationen und Web-Applikationen sind beides Begriffe, welche man möglicherweise schon mal gehört hat. Doch was bedeuten sie und wozu dienen sie?

2.2.1 Native App

Native Apps sind auf ein bestimmtes Betriebssystem zugeschnitten und funktionieren exakt so nicht in anderen Betriebssystemen. Diese Apps können auf diverse Funktionen des Betriebssystems zugreifen, sich zum Beispiel mit Widgets in diverse Teile des Betriebssystems integrieren und können auf den lokalen Speicher zugreifen. Sie werden grundsätzlich über den für das Betriebssystem eigenen Store installiert (z. B. App Store, Google Play Store oder Microsoft Store) oder in einem für das Betriebssystem (*engl.* «operating system»; OS) bekannten Format (z. B. *.msi* und *.exe* für Windows, *.apk* für Android) aus dem Internet heruntergeladen und installiert. Native Apps können sowohl ausschliesslich auf dem lokalen Gerät laufen als auch mit einem Server kommunizieren.

2.2.2 Web-App

Eine Web-App (oder Webapp) ist eine Webseite, welche mit diversen Funktionen Interaktionen ermöglicht. Meistens werden die Daten (z. B. Speicherstand) auf einem Server gespeichert und verarbeitet und können dadurch von verschiedenen Geräten aus zugegriffen werden. Eine solche App erfordert zwar keine Installation, jedoch eine Internetverbindung. Authentifizierungs- und Trackinginformationen werden, sofern relevant, in sogenannten Cookies im Browser gespeichert. Web-Apps können, wenn sie responsiv (*engl.* «responsive»; reaktionsfähig) sind, auf Smartphones und PCs mit unterschiedlichen Bildschirmgrössen verwendet werden.

2.2.3 Die Probleme

Während für viele Funktionen (z. B. NFC, Netzwerkzugriff) native Apps zwingend benötigt werden, ist nicht für alle Betriebssysteme eine separate App entwickeln zu müssen für die meisten Entwickler ein massgebliches Argument für Web-Apps. Dies spart erheblich Aufwand und Kosten und ermöglicht ein viel grösseres Zielpublikum. Jedoch fühlen sich

Web-Apps nicht wirklich wie Apps an. Man befindet sich immer noch im Browser und sieht die ganze Zeit die Adressleiste sowie diverse andere Steuerelemente. Nicht nur benötigen diese auf kleinen Bildschirmen viel Platz, sondern sie können auch zu versehentlicher Bedienung führen oder Inhalte der Web-App überdecken.

2.2.4 PWAs - die Kombination

Aus diesen Gründen wurden PWAs («Progressive Web Apps») ins Leben gerufen. Diese sind, wie der Name erwarten lässt, immer noch Web-Apps. Jedoch haben sie – besonders für Smartphones – einige wichtige Erweiterungen. PWAs können zum Home-Bildschirm hinzugefügt werden. Dort sehen sie wie jede andere App aus und auf den ersten Blick gibt es keinen Unterschied, denn sie starten – sofern richtig konfiguriert – im Vollbildmodus ohne die oben erwähnten Browser-Navigations-Buttons. Je nach Betriebssystem und Konfiguration können PWAs sogar in der Lage sein, offline zu funktionieren.

Der einzige Unterschied im Erstellen von normalen Web-Apps und PWAs ist, sofern nicht noch zusätzliche Funktionen wie einen Offline-Modus eingebaut werden sollen, eine einzige Konfigurationsdatei.

2.3 DER AUFBAU EINER WEBSEITE

Eine Webseite braucht zum Funktionieren sowohl ein Front- als auch ein Backend.

2.3.1 Backend

Unter dem Backend wird alles verstanden, was auf dem Server abläuft, respektive ausserhalb des Einflussbereichs des Clients passiert. Während die Aufgaben des Backends bei statischen Seiten nur die Auslieferung von Dateien beinhalten, spielt das Backend bei dynamischen Seiten eine grosse Rolle. Hier findet die Authentifizierung des Clients statt, es werden Berechnungen durchgeführt und die Seite wird für den Client vorbereitet. Nach der Verarbeitung einer *request* sendet der Server schliesslich eine *response* als Antwort, welche dann vom Browser interpretiert wird.

2.3.2 Frontend

Das Frontend beinhaltet alles, was auf der Clientseite abläuft. Vom Aussehen der Webseite bis zur Benutzerinteraktion ist hier alles dabei. Im Frontend gibt es drei Programmier-

und Auszeichnungssprachen, welche für Webseiten fast unumgänglich sind: HTML, CSS und JavaScript. Diese drei Sprachen werden in allen gängigen Browsern vollumfänglich unterstützt. Somit kann (fast) garantiert werden, dass Webseiten auf allen Geräten gleicher Grösse auch gleich aussehen.

2.3.2.1 HTML - das Gerüst

HTML («HyperText Markup Language») ist nicht direkt eine Programmiersprache, sondern eine Auszeichnungssprache (*engl.* «markup language») ⁵. Es gibt sie schon seit 1992 ⁶, also fast seit dem «Beginn» des Internets wie wir es heute kennen im Jahr 1990 ⁷.

HTML-Code besteht aus vielen durch sogenannte Tags (*engl.* «tag»; Etikett) definierten Elementen. HTML-Tags können an den für XML-artige Sprachen typischen kleiner als (<) und grösser als (>) Zeichen erkannt werden. Ein Element besteht in der Regel aus einem öffnenden (`<tag ...>`) und einem schliessenden (`</tag>`) Tag oder in gewissen Fällen aus einem selbstschliessenden Tag (`<tag ... />`) sowie optionalen Attributen wie zum Beispiel einer Klasse (`<div class="container mb-2">`) oder einer ID (`<h1 id="maintitle"></h1>`).



```
beispiel.html
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <meta charset="utf-8">
5     <title>HTML-Beispiel</title>
6   </head>
7   <body>
8     <h1 id="maintitle">HTML-Beispiel</h1>
9     <p class="info">Das ist ein HTML-Beispiel.</p>
10  </body>
11 </html>
```

Abbildung 1: Beispiel einer HTML-Datei

Abbildung 1 zeigt, wie eine sehr einfach HTML-Datei aussehen könnte.

⁵ <https://de.wikipedia.com/wiki/Auszeichnungssprache> (Abgerufen am 23.10.2022)

⁶ https://de.wikipedia.com/wiki/Hypertext_Markup_Language (Abgerufen am 23.10.2022)

⁷ <https://de.wikipedia.com/wiki/Internet> (Abgerufen am 23.10.2022)

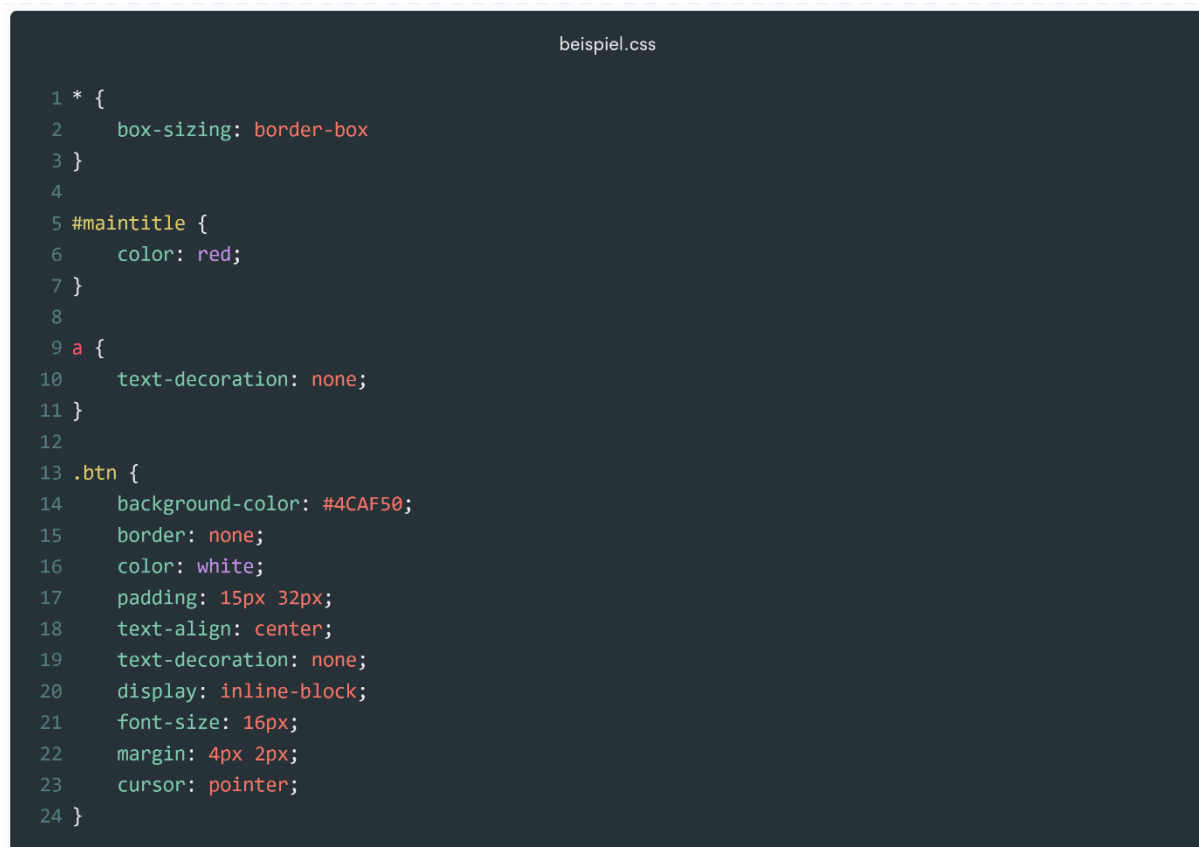
Mit HTML wird die Hierarchie der Elemente einer Webseite sowie einige deren Eigenschaften definiert. HTML definiert aber nur den Inhalt, Elemente können damit nicht positioniert oder gefärbt werden.

Weitere Informationen über HTML sowie Details zum obigen Beispiel befinden sich im Anhang A.1.

2.3.2.2 CSS - die Fassade

Da blosse HTML-Seiten kaum Möglichkeiten zur schönen Darstellung bieten, wurde CSS («Cascaded Style Sheet») erfunden und 1996 publiziert⁸. Mit CSS ist es möglich, Elemente anders darzustellen, als sie normalerweise aussehen würden. Dies beinhaltet Farben, Grössen, Positionierung und sogar Animationen.

Abbildung 2 zeigt, wie eine einfache CSS-Datei aussehen könnte.



```
beispiel.css

1 * {
2   box-sizing: border-box
3 }
4
5 #maintitle {
6   color: red;
7 }
8
9 a {
10  text-decoration: none;
11 }
12
13 .btn {
14   background-color: #4CAF50;
15   border: none;
16   color: white;
17   padding: 15px 32px;
18   text-align: center;
19   text-decoration: none;
20   display: inline-block;
21   font-size: 16px;
22   margin: 4px 2px;
23   cursor: pointer;
24 }
```

Abbildung 2: Beispiel einer CSS-Datei

⁸ https://de.wikipedia.org/wiki/Cascading_Style_Sheets (Abgerufen am 23.10.2022)

CSS besteht aus verschiedenen Anweisungsgruppen, welche die Darstellung von Elementen verändern. Zu jeder Anweisungsgruppe gehört ein Selektor. Dieser gibt an, auf welche Elemente eine Anweisungsgruppe zutrifft. Selektoren können auch auf verschiedene Arten kombiniert werden. Unter anderem können Elemente auch anhand ihrer Vorfahren (in der Hierarchie höherstehende Elemente) oder Nachbarelemente ausgewählt werden.

In den Anweisungsgruppen (den geschweiften Klammern nach einem Selektor) folgen schliesslich durch Semikolon getrennte Anweisungen.

Es gibt drei verschiedene Arten, wie CSS in HTML integriert werden kann:

1. Direkt beim jeweiligen Element («inline»):

```
<div style="color: red"></div>
```

2. Eingebettet im HTML:

```
<style>div { color: red; }</style>
```

3. Mit Verlinkung (im *head* -Tag) zu einer separaten Datei/URL:

```
<link rel="stylesheet" href="./mystyle.css">
```

Alle drei Versionen haben ihre Vor- und Nachteile, aber die dritte Version wird grundsätzlich vorgezogen, da der HTML-Code somit aufgeräumter erscheint.

Mit CSS können auch bereits kleine interaktive Dinge umgesetzt werden, wie z. B. ein Drop-down-Menü oder eine Farbänderung beim «Drüberfahren» (*engl.* «hover»). Für komplexere Interaktionen kann CSS jedoch nicht mehr verwendet werden.

Weiterführende Informationen über CSS befinden sich im Anhang A.2.

2.3.2.3 JavaScript - die Technik

Mit der Programmiersprache JavaScript (kurz: JS) kann die Funktionalität einer Webseite massiv gesteigert werden. Mit JavaScript können Berechnungen durchgeführt, HTTP(S)-Abfragen im Hintergrund gemacht sowie HTML-Elemente verändert und erstellt werden. Nebst all den typischen Funktionen von Programmiersprachen kann JavaScript auch auf Interaktion des Benutzers mit der Webseite reagieren und dementsprechend Aktionen ausführen. Wie CSS kann auch JavaScript auf drei verschiedene Arten in HTML eingebunden werden:

1. Direkt bei einem Element («inline»):

```
<button onclick="alert('Hi!')>Klick mich!</button>
```

(wird bei entsprechenden vom Benutzer ausgelösten Aktionen ausgeführt)

2. Eingebettet im HTML:

```
<script>...</script>
```

3. Mit Verlinkung (im *head* -Tag) zu einer separaten Datei/URL:

```
<script src="myscript.js"></script>
```

Abbildung 3 zeigt, wie eine einfache JavaScript-Datei aussehen könnte.



```
beispiel.js

1 function alertCurrentTime() {
2   let _dt = new Date();
3   let _date = _dt.toLocaleString("de-CH", {
4     day: "numeric",
5     month: "long",
6     year: "numeric",
7   });
8   let _time = _dt.toLocaleString("de-CH", {
9     hour: "numeric",
10    minute: "numeric",
11  });
12  alert(`Heute ist der ${_date}. Es ist ${_time} Uhr!`);
13 }
14
15 window.addEventListener('load', alertCurrentTime);
```

Abbildung 3: Beispiel einer JavaScript-Datei

Dieses Beispiel würde beim Laden der Seite ein Pop-Up mit der aktuellen Zeit anzeigen.

Es ist zu beachten, dass JavaScript vereinzelte Ähnlichkeiten zur Programmiersprache Java besitzt, mit dieser jedoch bis auf den Namen keineswegs in Verbindung steht!

Weiterführende Informationen über JavaScript befinden sich im Anhang A.3.

3 DURCHFÜHRUNG

Mit dem Grundverständnis einer Webseite und den dazu benötigten Sprachen sind nun die wichtigsten Voraussetzungen erfüllt, um das Projekt zu starten.

3.1 DER APP-NAME

Als Name für die Web-App wurde «Teamized» ausgewählt. Dieser Name ist eine Mischung aus den englischen Wörtern «team», «optimized» und «amazed». Er soll ungefähr aussagen, dass das Arbeiten im Team einfacher umzusetzen ist als man denkt. Da die Wortendung «-ized» in etwa mit «-isiert» übersetzt werden kann, könnte dieser Name auch etwas wie «teamisiert» («zum Team gemacht») bedeuten.

Bemerkung: Zu Beginn des Projektes wurde der Platzhaltername «OrgaTask» verwendet. Dieser ist vereinzelt noch im Code zu finden.

3.2 APP-SPEZIFIKATIONEN DEFINIEREN

Die Grundfunktion der App soll darin bestehen, dass Teams erstellt werden können, Personen zu Teams hinzugefügt werden können und zwischen Teams hin- und her gewechselt werden kann. Zusätzlich sollen folgende Seiten Teil der App sein:

- **Arbeitszeit:** Hier sollten Benutzer ihre Arbeitszeit aufzeichnen können und Graphen davon ansehen können.
- **Kalender:** Hier soll es möglich sein, dass Teams gemeinsame Kalender verwalten und auf ihren eigenen Geräten abonnieren können.
- **To-do-Listen:** Diese Seite soll Teams das gemeinsame Verwalten von To-do-Listen ermöglichen.

Die App selbst soll aus Kacheln verschiedener Grössen bestehen, welche je nach Bildschirmgrösse nebeneinander oder untereinander angeordnet sind. Das Wechseln zwischen Seiten soll für den Benutzer ohne Verzögerung ablaufen. Falls doch noch Dinge geladen werden müssen, soll ihm eine Ladeanimation angezeigt werden.

3.3 AUSWÄHLEN DER SOFTWARE & TOOLS

Aufgrund der im Theorieteil beschriebenen Vor- und Nachteile eignet sich eine PWA für dieses Projekt am besten. Somit können Desktop-Benutzer die Web-App als normale Webseite nutzen und Mobile-Benutzer können die Web-App wie eine native App zum Home-Bildschirm hinzufügen.

3.3.1 Backend

Für das Backend gibt es viele sogenannte Frameworks (*engl.* «framework»; Gerüst). Diese übernehmen die grundlegende Verarbeitung von requests und responses, welche bei allen Webseiten grundsätzlich gleich verläuft und ermöglichen somit die Fokussierung auf die für die gewünschte App besonderen Dinge. Auch werden diese Frameworks von Programmierern aus der ganzen Welt unterhalten und somit werden regelmässig neue Sicherheitslücken geschlossen und Funktionen erweitert.

Einige bekannte Backend-Frameworks und die Programmiersprache, in welchen sie geschrieben sind, sind Flask (Python), Django (Python), Express.js (JavaScript), Meteor.js (JavaScript), ASP.NET (.NET), Ruby on Rails (Ruby), CakePHP (PHP) sowie Laravel (PHP). All diese Frameworks haben ihre Vor- und Nachteile. Für dieses Projekt stellte sich jedoch Django als geeignetstes Framework heraus. Es besitzt hervorragende Tools zur Datenbankverwaltung, besitzt integrierte Authentifizierungsmöglichkeiten und dank der Programmiersprache Python ist es eher einfach zu verstehen. Für dieses spezifische Projekt ist es zusätzlich aufgrund bereits bestehender Infrastruktur und Vorkenntnissen sehr gut geeignet.

3.3.2 Frontend

Auch für das Frontend gibt es Frameworks. Es ist zwar möglich, Webseite ohne Frontend-Framework zu erstellen und den Seitenaufbau auf das Backend zu verschieben oder mit JavaScript direkt umzusetzen, jedoch können Frontend-Frameworks sehr hilfreich beim Erstellen von interaktiven Apps sein, besonders wenn sie sich wie eine App anfühlen sollen und nicht bei jedem Seitenwechsel die Seite neu laden sollen. Die bekanntesten Namen hierbei sind React, Vue und Angular. Aus persönlichen Interessen wurde für dieses Projekt React ausgewählt.

Hinweis: Angular ist der Nachfolger von AngularJS, welches seit Anfang 2022 nicht mehr unterhalten wird ⁹.

Mithilfe von Babel.js, welches die JavaScript-Dateien «übersetzen» kann, ist es möglich, HTML-ähnlichen Code (JavaScript XML, kurz JSX) ¹⁰ direkt in die JavaScript-Dateien zu schreiben, was React perfekt ergänzt.

Ausserdem wurden für dieses Projekt folgende weitere Bibliotheken gewählt: JQuery, FontAwesome, Bootstrap, Sweetalert2 sowie Recharts. JQuery ist eine Ergänzung zu JavaScript und vereinfacht das Lesen und Ändern von HTML-Elementen, FontAwesome fügt eine ganze Bibliothek mit diversen Icons hinzu, welche auf der Webseite eingesetzt werden können, Bootstrap bietet eine Sammlung von Standard-CSS-Klassen fürs Designen der Seite, SweetAlert2 ist eine Bibliothek für Pop-up-Alerts und -Formulare und Recharts ist eine Bibliothek für Diagramme und Grafiken.

3.3.3 Hosting

Damit die Webseite durchgehend erreichbar ist, muss ein Server 24/7 laufen und im Internet erreichbar sein. Dazu gibt es verschiedene Dienste, welche das Mieten diverser Server-Ressourcen ermöglichen, sogenannte «platform as a service» (PaaS) oder «infrastructure as a service» (IaaS). Heroku ¹¹ ist eine solche Plattform. Für kleine Projekte bietet sie sogar ein kostenloses Angebot an, bei welchem sich der Server jedoch jeweils nach 30 Minuten ausschaltet und erst bei der nächsten Verbindung wieder einschaltet, was zu Verzögerungen kommen kann. Da diese komplett kostenlose Option keine Bestätigung der Identität erforderte, wurde sie leider sehr häufig missbraucht und Heroku hat sich entschieden, diese Angebote zu ersetzen. ¹² Bereits ab \$7 USD pro Monat gibt es jedoch Angebote für einen dauerhaft laufenden Server. ¹³ Studenten erhalten bei Bestätigung ihres Studentenstatus eine Gutschrift im Wert von \$156 USD und können

⁹ <https://blog.angular.io/discontinued-long-term-support-for-angularjs-cc066b82e65a> (Abgerufen am 23.10.2022)

¹⁰ <https://reactjs.org/docs/introducing-jsx.html> (Abgerufen am 23.10.2022)

¹¹ <https://www.heroku.com/about> (Abgerufen am 23.10.2022)

¹² <https://blog.heroku.com/next-chapter> (Abgerufen am 23.10.2022)

¹³ <https://www.heroku.com/dynos> (Abgerufen am 23.10.2022)

während einem Jahr die kostenpflichtigen Dienste somit kostenlos entdecken und neues lernen.^{14 15} Es gibt auch andere PaaS und IaaS Dienste, aber Heroku ist nach persönlicher Erfahrung sehr zuverlässig und gut dokumentiert, weshalb es sich für dieses Projekt perfekt eignet.

3.3.4 Programme zur Entwicklung

Um Programmcode gut bearbeiten zu können, braucht es einen guten Editor (Textbearbeitungsprogramm). Optimalerweise bietet dieser nebst Syntax-Highlighting (farbige Markierung von verschiedenen Aspekten des Codes) auch automatische Vervollständigung und intelligente Code-Informationen an. Der wohl am weitesten verbreitete Editor heutzutage ist Visual Studio Code (kurz VS Code) von Microsoft¹⁶. Er bietet bereits von sich aus viele Funktionen an und durch Erweiterungen kann er beliebig mit Funktionen erweitert werden. Für dieses Projekt ist er perfekt geeignet.

Um den Änderungsverlauf über verschiedene Versionen des Projekts zu erhalten, wird die die Open-Source-Versionsverwaltungssoftware Git¹⁷ zusammen mit der dazugehörenden Onlineplattform GitHub¹⁸ verwendet. Der von GitHub entwickelte Copilot¹⁹ bringt ausserdem von künstlicher Intelligenz unterstützte automatische Vervollständigung in VS Code. Dieser fördert besonders bei repetitiven Dingen die Produktivität.

3.4 ENTWICKLUNGSUMGEBUNG UND PROJEKT VORBEREITEN

Bevor mit dem Programmieren begonnen werden kann, müssen zuerst diverse Vorkehrungen getroffen werden.

¹⁴ <https://www.heroku.com/github-students> (Abgerufen am 23.10.2022)

¹⁵ <https://blog.heroku.com/github-student-developer-program> (Abgerufen am 23.10.2022)

¹⁶ <https://code.visualstudio.com/> (Abgerufen am 23.10.2022)

¹⁷ <https://www.git-scm.com/> (Abgerufen am 23.10.2022)

¹⁸ <https://github.com/> (Abgerufen am 23.10.2022)

¹⁹ <https://github.com/features/copilot/> (Abgerufen am 23.10.2022)

3.4.1 Installationen

Folgende Programme mussten für dieses Projekt heruntergeladen und installiert werden: Git ²⁰, VS Code ²¹ (oder ein alternativer Editor), npx mit npm und Node.js (für Babel.js) ²² sowie Python ²³. Ausserdem musste noch Django installiert werden:

```
python -m pip install -U django
```

3.4.2 Ordnerstruktur

Um Ordnung zu behalten, benötigt es eine logische und der Norm entsprechende Ordnerstruktur. Im Hauptordner (*django-teamized*) befinden sich nebst der *README.md* Datei (standardisierte Benennung für die Datei, welche Informationen und Anleitungen von einem Programm/Projekt beinhaltet) diverse Dateien, welche für Python wichtig sind. Dazu gehören *requirements.txt* (bestimmt, welche Versionen von Python Bibliotheken installiert werden *sollten*), *setup.py* und *setup.cfg* (bestimmen diverse Attribute des Python Moduls und welche Bibliotheken installiert werden *müssen*), *MANIFEST.in* (bestimmt alle Dateien/Ordner, welche zum Python Modul gehören), *_version.txt* (die aktuelle Version des Moduls) sowie *LICENSE* (die Lizenz, mit welcher der Quellcode veröffentlicht wird). Dank all diesen Dateien ist es möglich, die Django-App in verschiedene Django-Projekte zu integrieren. Es ist nun eine sogenannte «reusable Django app». Auch die *.gitignore*-Datei (welche besagt, welche Dateien die Software Git ignorieren soll) und relevante Dateien für GitHub im *.github*-Ordner sind hier.

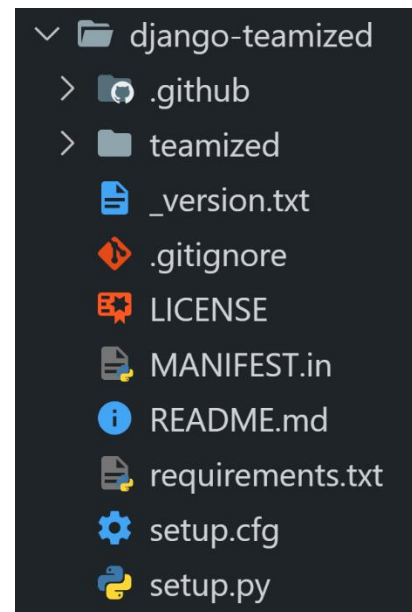


Abbildung 4: Ordnerstruktur von «django-teamized» (in VS Code)

²⁰ <https://git-scm.com/book/en/v2/Getting-Started-Installing-Git> (Abgerufen am 23.10.2022)

²¹ <https://code.visualstudio.com/Download> (Abgerufen am 23.10.2022)

²² <https://docs.npmjs.com/downloading-and-installing-node-js-and-npm> (Abgerufen am 23.10.2022)

²³ <https://www.python.org/downloads/> (Abgerufen am 23.10.2022)

Der Ordner mit dem eigentlichen Code ist der *teamized* Unterordner. Dieser wurde mit `python -m django startapp teamized` generiert und enthält bereits von Beginn an wichtige Dateien für Django, besonders *apps.py*, *models.py*, *views.py*, *urls.py* und *admin.py*. In *apps.py* werden grundlegende Einstellungen der Django-App getroffen, besonders aber der Name der App. Dieser wird in Django vor allem im Bereich Datenbank benötigt. In *models.py* werden Datenbankmodelle definiert. Dies sind Python-Klassen, welche einerseits die Spalten der Datenbanktabelle definieren sowie Funktionen beinhalten, welche auf Objekte dieser Klasse angewendet werden können. Die Kommunikation mit der Datenbank in der *SQL*-Sprache wird komplett von Django übernommen. Bei der Benutzung von Django muss nur mit Python-Objekten gearbeitet werden. Dies ermöglicht es auch, unabhängig von der Datenbank und deren Software zu denken, denn

Django «übersetzt» die Anweisungen für die jeweils leicht verschiedenen Datenbankmanagementsysteme (DBMS). In *views.py* werden Funktionen definiert, welche beim Aufrufen einer URL mit Angabe der *request* aufgerufen werden. Der Rückgabewert dieser Funktionen wird als *response* zum Client zurückgesendet. Welche URL oder welches URL-Muster zu welcher *view*-Funktion gehört, wird in *urls.py* definiert. Django besitzt ausserdem ein sehr nützliches Admin-Tool. In *admin.py* wird dazu definiert, welche Modelle und Felder im Admin-Tool angezeigt werden sollen. Dieses Tool ist besonders für Testzwecke ein wichtiges Tool, für den normalen Benutzer ist dieses nicht zugänglich. Die anderen Python-Dateien in diesem Ordner sind dazu da, Code etwas zu ordnen. Für Django haben sie keine direkte Bedeutung.

Mit dem `python manage.py makemigrations`-Befehl aus dem Django-Projekt erstellt Django automatisch Dateien in *migrations*, welche die Änderungen an den Datenbank-Modellen beinhalten. Mit `python manage.py migrate` werden alle noch nicht angewandten Änderungen in der Datenbank angewandt. Dies ist besonders praktisch bei mehreren Konfigurationen mit verschiedenen Datenbanken. Für das Erstellen eines

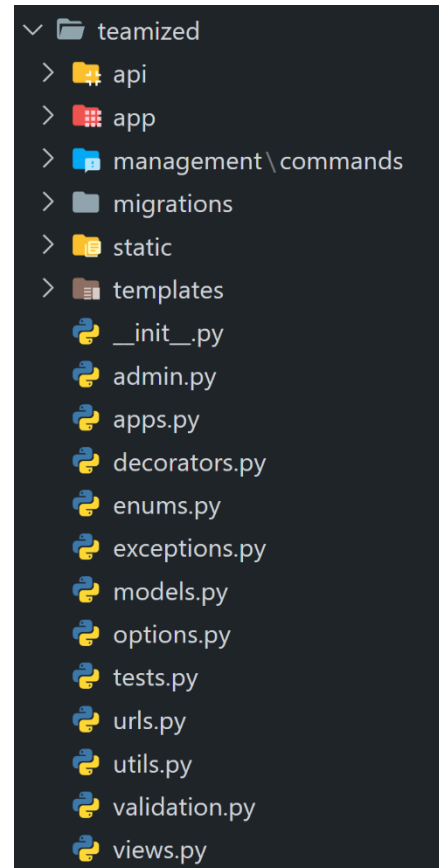


Abbildung 5: Ordnerstruktur von «teamized» (in VS Code)

Django-Projekts und somit der `manage.py`-Datei wird hier auf die sehr anfängerfreundliche Django-Dokumentation verwiesen.²⁴ Der `python manage.py`-Befehl kann ausserdem durch Dateien im `management/commands`-Ordner von jeder installierten App erweitert werden.

Der `templates`-Ordner beinhaltet HTML-Vorlagen, welche von view-Funktionen zusammengebaut und individuell gefüllt werden. In `static` befinden sich statische (unveränderliche) Dateien wie zum Beispiel Bilder, CSS-Dateien und JavaScript-Dateien. Damit das Frontend mit dem Backend kommunizieren kann, benötigt es eine API («application programming interface»). Um eine klare Trennung zwischen Endpunkten, welche vom Benutzer direkt abgerufen werden können und Endpunkten, welche nur im Hintergrund abgerufen werden sollten zu schaffen, werden API-Endpunkte in den `api`-Unterordner verschoben. Die dort liegenden Dateien werden von den Hauptdateien entsprechend importiert. Jetzt folgt noch der wichtigste Unterordner: `app`. Dies ist der Platz für die React-App

Bevor es weitergeht, wird diese neue Django-App noch in das bereits bestehendes Django-Projekt eingebaut. Dazu wird `"teamized"` zur `INSTALLED_APPS`-Einstellung hinzugefügt und mit `path('teamized/', include('teamized.urls'))` in die projektweiten `urlpatterns` eingefügt.

3.4.3 JSX-Kompilierung mit Babel.js

Wie bereits im Theorieteil erwähnt, muss der JSX-Quellcode zuerst durch babel.js in normales JavaScript umgewandelt werden. Dazu existiert im `app`-Ordner die `package.json`-Datei. Diese beinhaltet eine Liste der benötigten Bibliotheken sowie zwei Befehlsdefinitionen. Mit `npm install` können die benötigten Bibliotheken installiert werden und mit den erstellten Befehlen `npm run build` und `npm run build-live` kann der ganze `src`-Unterordner (kurz für engl. «source»: Quelle) kompiliert (d.h. umgewandelt / übersetzt) und in den `static`-Ordner kopiert werden. Der erste Befehl verkleinert die Dateien nebenbei, was für die Produktionsumgebung geeignet ist, da dies Speicherplatz spart. Der zweite Befehl wartet zusätzlich auf Änderungen an den

²⁴ <https://docs.djangoproject.com/en/4.1/intro/tutorial01/> (Abgerufen am 23.10.2022)

Quelldateien und übersetzt diese fortlaufend. Somit ist er zur Entwicklung sehr praktisch, da Änderungen direkt beim nächsten Reload der Seite angewendet werden können, ohne jedes Mal manuell einen Befehl auszuführen.

3.5 CODING

Nun ist alles soweit vorbereitet und die Entwicklung der App kann beginnen. Aus Platzgründen wird hier auf längere Codebeispiele verzichtet. Der ganze Quellcode ist auf GitHub unter <https://github.com/rafaelurben/django-teamized> verfügbar.

3.5.1 Umsetzung Authentifizierung

Django selbst besitzt bereits integrierte Authentifizierungsmöglichkeiten via Benutzername und Passwort. Nicht jeder möchte jedoch ein Passwort verwenden. Deshalb wurden zum Login dieser App auch Online-Dienste wie Google, Discord und GitHub als mögliche Anmeldeoptionen integriert. Dabei ist der jeweilige Dienst für das Authentifizieren des Benutzers verantwortlich und der Benutzer kann sich mit nur einem oder zwei Klicks in der App einloggen, ohne je ein Passwort zu erstellen. Dies wird über das sogenannte OAuth 2.0 Protokoll ermöglicht.

Die Accountverwaltung inkl. Login ist jedoch aus Platzgründen nicht direkt Teil dieser Arbeit. Der verwendete Quellcode ist aber ebenfalls öffentlich auf GitHub unter <https://github.com/rafaelurben/django-account> einsehbar.

3.5.2 Umsetzung Backend

Als erstes werden die HTML-Vorlagen benötigt. Die App selbst besteht nur aus zwei Seiten: Einer Übersichtsseite mit diversen Informationen und der App-Seite. Zusätzlich folgen noch eine Error- und Debug-Seite. All diese Seiten sollen dasselbe Grundlayout mit einer Navigationsleiste auf der oberen Seite haben. Dies wird in der *base.html*-Vorlage definiert. Dort werden auch die zuvor erwähnten CSS- und JavaScript-Bibliotheken geladen. Mithilfe der Django-Template-Sprache (Abbildung 6) können in dieser Vorlage (Platzhalter-)Blöcke verwendet werden, welche von weiteren Vorlagen ersetzt oder erweitert werden können, ohne den Rest der Datei mehrfach speichern zu müssen. Genau dies passiert auch hier: In *app.html* wird zusätzlich ein Behälter für eine Seitenleiste

platziert sowie natürlich die Frontend-React-App importiert. Ausserdem wird die Navigationsleiste um einige Elemente erweitert.

```
teamized/templates/teamized/404.html

1 {% extends 'teamized/base.html' %}
2
3 {% block title %}404{% endblock %}
4
5 {% block content %}
6     Diese Seite wurde leider nicht gefunden.
7 {% endblock content %}
```

Abbildung 6: Beispiel der Django Template-Sprache

Die Navigationsleiste enthält neben dem App-Namen einen Link zur Accountverwaltung und der Startseite sowie einen Knopf zum Ausloggen. Diese wichtigen Menüpunkte sind durchgehend sichtbar. Auf mobilen Geräten werden sie in einem Dropdown zusammengefasst. Bevor auf die App-Seite zugegriffen werden kann, muss sich ein Benutzer zuerst registrieren oder einloggen.

```
teamized/views.py

1 from django.shortcuts import render
2
3 def home(request):
4     "Show the home page"
5
6     return render(request, 'teamized/home.html')
```

Abbildung 7: Beispiel einer Django view-Funktion

Diese Vorlagen müssen nun nur noch in einer view-Funktion (Abbildung 7) gerendert und zurückgegeben und diese wiederum in *urls.py* referenziert werden und schon kann die Navigationsleiste nach Starten des Servers ²⁵ betrachtet werden (Abbildung 8).

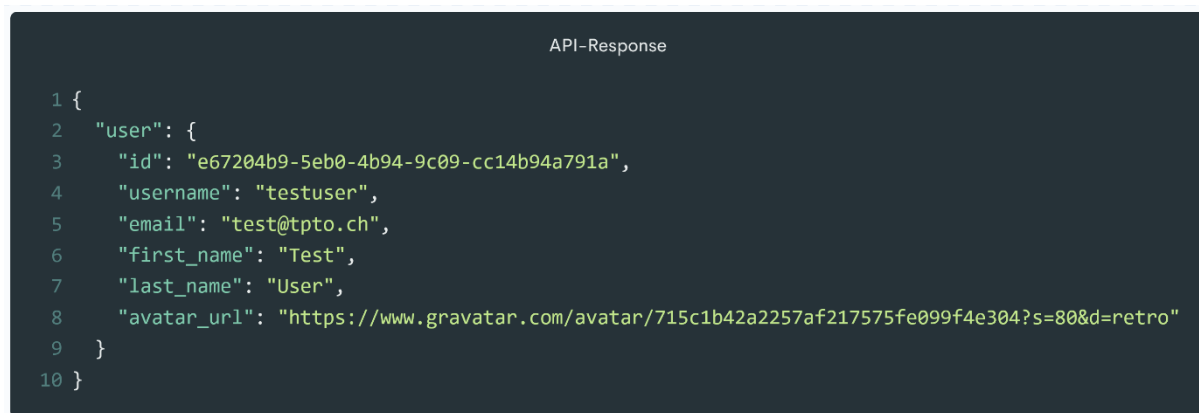


Abbildung 8: Navigationsleiste (Screenshot)

Weitere Screenshots der Web-App befinden sich im Anhang B.

²⁵ <https://docs.djangoproject.com/en/4.1/ref/django-admin/> (Abgerufen am 23.10.2022)

Damit die App auch etwas kann, muss sie zuerst Dinge speichern können. Hierbei kommen Datenbankmodelle zum Einsatz. Für die Grundfunktionalitäten werden die Modelle *User*, *Team*, *Member* und *Invite* verwendet, wobei *Member* die Mitgliedschaft eines *Users* in einem *Team* darstellt. Damit die App auch mit diesen Modellen interagieren kann, gehören zu jedem Modell entsprechende API-Endpunkte zum Abrufen, Hinzufügen, Bearbeiten sowie Löschen von Objekten. Die Endpunkte sind alle vor unbefugten Benutzern geschützt. Ein Benutzer kann also nicht einfach die Mitglieder eines anderen Teams abfragen. Die Endpunkte geben ihre Antwort im JSON («JavaScript object notation») Format zurück. Dies sieht in etwa so aus:



```
API-Response
1 {
2   "user": {
3     "id": "e67204b9-5eb0-4b94-9c09-cc14b94a791a",
4     "username": "testuser",
5     "email": "test@tp.to.ch",
6     "first_name": "Test",
7     "last_name": "User",
8     "avatar_url": "https://www.gravatar.com/avatar/715c1b42a2257af217575fe099f4e304?s=80&d=retro"
9   }
10 }
```

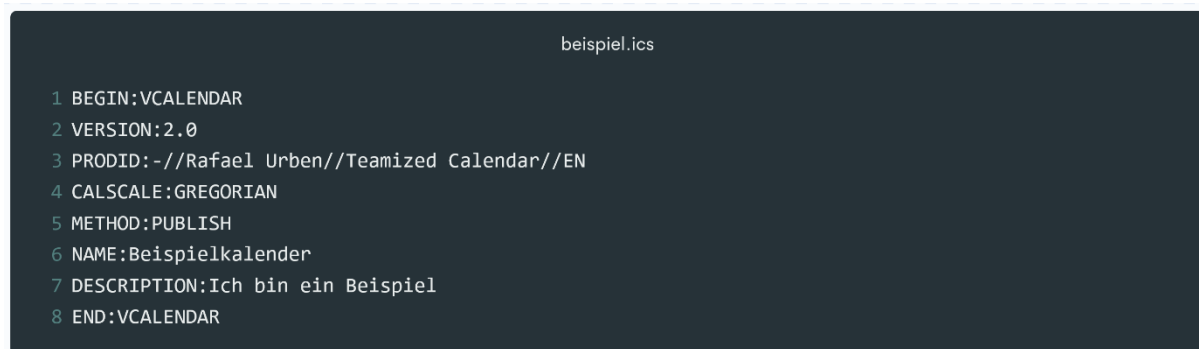
Abbildung 9: Beispiel einer API-Response im JSON-Format

Wie auch in Abbildung 9 ersichtlich, wird das Profilbild bei Gravatar ²⁶ abgerufen. Dies ist ein kostenloser Dienst, bei welchem Benutzer ihre E-Mail-Adresse mit einem Profilbild verknüpfen können, welches dann auf allen Seiten verwendet wird, welche diesen Dienst unterstützen. Die E-Mail-Adresse selbst wird von der App jedoch nie an den Dienst weitergegeben, sondern nur ein Hash davon. Eine Hash-Funktion ist eine nach heutigem Stand unumkehrbare Funktion, welche bei gleicher Eingabe immer das gleiche Ergebnis liefert. Gravatar speichert den Hash aller E-Mail-Adressen und weiss somit trotzdem, welches Profilbild zu welchem Benutzer gehört.

Für alle weiteren Funktionen der App (Arbeitszeit, Kalender, To-do-Listen) ist das Prinzip ähnlich. Einzig der Kalender erfordert einen erwähnenswerten zusätzlichen Aufwand im Backend: Da der Kalender auch abonniert werden können soll, benötigt dieser eine

²⁶ <https://gravatar.com/> (Abgerufen am 23.10.2022)

zusätzliche Schnittstelle. Dazu wurde für diese App das iCalendar-Protokoll ²⁷ ausgewählt. Dieses ist mit den gängigsten Kalender-Applikationen wie Apple Calendar, Google Calendar sowie Microsoft 365 kompatibel. Da dieses Protokoll nur mit reinem Text arbeitet, ist es auch relativ einfach zu implementieren. Ein leerer Kalender sieht in diesem Protokoll so aus:

The image shows a dark-themed code editor window titled 'beispiel.ics'. It contains the following text:

```
1 BEGIN:VCALENDAR
2 VERSION:2.0
3 PRODID:-//Rafael Urben//Teamized Calendar//EN
4 CALSCALE:GREGORIAN
5 METHOD:PUBLISH
6 NAME:Beispielkalender
7 DESCRIPTION:Ich bin ein Beispiel
8 END:VCALENDAR
```

Abbildung 10: Beispiel einer .ics-Datei für das iCalendar-Protokoll

Für die Umsetzung wurde nebst ein paar zusätzlichen Methoden für das Zusammensetzen der einzelnen Textelemente in den *Calendar* und *CalendarEvent* Modellen ein zusätzlicher öffentlicher (ohne Authentifizierung zugänglicher) Endpunkt erstellt. Um den Kalender darüber abzurufen, muss aber die exakte URL bekannt sein. Somit ist der Kalender gegen Fremdzugriffe weiterhin geschützt.

3.5.3 Umsetzung Frontend

Die Entwicklung des Frontends ist schon etwas komplexer. Begonnen wird dazu in der *app.jsx*-Datei im *app/src*-Ordner, welche auch im HTML referenziert wird. Darin werden erst einmal einige globale Variablen initiiert. Auch die Logik zum «soft reload», einem Aktualisieren der Inhalte, ohne die Seite neu zu laden, befindet sich hier. Von hier aus werden alle wichtigen Prozesse gestartet und weitere Dateien geladen. Diese befinden sich in zwei danebenliegenden Unterordner: *components* und *utils*.

In *components* befinden sich alle verwendeten React-Komponenten. Dies sind einzelne (wiederverwendbare) Teile der Webseite oder ganze Seiten. Für den Aufbau der einzelnen Seiten helfen die Komponenten in *dashboard.jsx*. Mit diesen wird das typische

²⁷ <https://icalendar.org/> (Abgerufen am 23.10.2022)

Kachellayout der Seiten erstellt und dafür gesorgt, dass alle Seiten automatisch nach dem gleichen Stil aufgebaut sind. Aber die wichtigste Komponente ist der *PageLoader* aus *pageLoader.jsx*. Er ist die höchste React-Komponente der Seite und lädt die Seitenkomponente der zurzeit ausgewählten Seite. Beim ersten Laden sowie Aktualisieren der Seite zeigt er ausserdem eine Ladeanimation an. Doch auch diese Komponente allein bringt nichts; sie muss zuerst geladen werden. Da sich die ganze App auf nur einer Seite abspielt, braucht es dazu aber zuerst eine Navigationslogik.

Für die Navigationslogik gibt es Tools wie React Router ²⁸, jedoch wurde für diese App eine eigene Lösung entwickelt. Diese lebt in *utils/navigation.js*. Sie basiert auf der History-API des Browsers, mit welcher der Seitenverlauf und die angezeigte URL manipuliert werden kann, ohne jedoch die Seite wirklich zu wechseln. Das aktuell ausgewählte Team sowie die aktuell ausgewählte Seite werden als sogenannte Suchparameter in der URL gespeichert. Diese sehen in etwa so aus: *.../app?p=calendars&t=8b7bd0d5-9193-4603-97c5-4a2698991ad5*. Beim erneuten Öffnen dieser Seite oder beim Zurücknavigieren im Browserverlauf können diese Angaben wieder aus der URL extrahiert und somit die entsprechende Seite angezeigt werden. Auch das Rendering der Hauptkomponenten geschieht in dieser Datei: Die *AppMenubar*, die *AppSidebar* und der *PageLoader* werden mit der *render()*-Funktion in deren entsprechende HTML-Container geladen. Wird die *render()*-Funktion erneut aufgerufen, wird React von nun an nur noch die geänderten Komponenten tatsächlich ändern. Somit können viele Ressourcen gespart werden.

Womit auch viele Ressourcen gespart werden können, ist mit Caching. Caching bedeutet, dass Daten nicht jedes Mal vom Server abgerufen werden, sondern lokal zwischengespeichert werden. Auch hier wurde eine eigene Lösung umgesetzt. Die Liste aller Teams, bei welchen der Benutzer Mitglied ist, wird in einer globalen Variable gespeichert. Sobald eine Seite aufgerufen wird, welche weitere Daten erfordert (zum Beispiel die Mitgliederliste), werden diese bei der API abgefragt und unter dem entsprechenden Team gespeichert. Wenn diese Seite nun erneut aufgerufen wird,

²⁸ <https://reactrouter.com/en/main> (Abgerufen am 23.10.2022)

können die lokal gespeicherten Daten verwendet werden und der Weg über den Server entfällt. Somit wird ein verzögerungsfreier Seitenwechsel möglich.

Für die Kommunikation mit der API gibt es die *utils/api.js*-Hilfsmittel. Diese legen die Basis dafür, wie Anfragen an den Server gesendet werden. Diese finden in sämtlichen anderen Dateien im *utils*-Ordner Anwendung. Für die meisten Datenbankmodelle sind die benötigten Endpunkte und somit die benötigten Funktionen sehr ähnlich. Objekte können aufgelistet, erstellt, geändert oder gelöscht werden. Zu jeder Aktion gehört meistens eine zweite Funktion, welche diese Aktion mithilfe von Sweetalert2 in ein schönes benutzerfreundliches Pop-up verpackt, entweder als Formular oder Bestätigung. Die Datei *todo.js* enthält zum Beispiel folgende Funktionen:

```
teamized/app/src/utils/todo.js

1 export async function getToDoLists(teamId) {...}
2 export async function createToDoList(teamId, name, description, color) {...}
3 export async function createToDoListPopup(team) {...}
4 export async function editToDoList(teamId, todolistId, name, description, color) {...}
5 export async function editToDoListPopup(team, todolist) {...}
6 export async function deleteToDoList(teamId, todolistId) {...}
7 export async function deleteToDoListPopup(team, todolist) {...}
8 export async function createToDoListItem(teamId, todolistId, name, description) {...}
9 export async function editToDoListItem(teamId, todolistId, itemId, name, description, done) {...}
10 export async function editToDoListItemPopup(team, todolist, item) {...}
11 export async function viewToDoListItemPopup(team, todolist, item) {...}
12 export async function deleteToDoListItem(teamId, todolistId, itemId) {...}
13 export async function deleteToDoListItemPopup(team, todolist, item) {...}
```

Abbildung 11: Beispielcode – Funktionen des To-do-Listen-Moduls

Dies sind alle Funktionen im Bereich To-do-Listen, welche nötig sind, um sowohl mit dem Backend als auch mit dem Benutzer zu kommunizieren. In den Funktionen, welche Änderungen an der Datenbank vornehmen, wird der lokale Cache ebenfalls angepasst bzw. ergänzt.

Um das Ganze nun auch bedienbar zu machen, gibt es die Seitenkomponenten. Innerhalb einer Seitenkomponente werden die oben erwähnten Dashboard-Komponenten verwendet. Darin folgen die Inhalte der einzelnen Kacheln, welche je nach Komplexität selbst wieder eigene Komponenten sind. Viele einzelne, dafür sehr spezifische Funktionen zu verwenden wird in der Informatik als übliche Praxis angesehen. Deshalb werden den Komponenten immer nur diejenigen Daten weitergegeben, welche diese benötigen, um ihren Zweck zu erfüllen.

Mit der Erstellung der Seitenkomponenten ist die Entwicklung einer ersten Version der App nun so weit abgeschlossen.

3.6 TESTING

Was wie bei jedem Projekt natürlich nicht fehlen darf, ist das Testen. Aufgrund der vielen verschiedenen Funktionen und begrenzter Zeit eignet sich automatisiertes Testen für dieses Projekt eher nicht. Hier liegt der Mensch gegenüber dem Computer im Vorteil: Er kann viel mehr Dinge auf einmal beachten und sieht auch Fehler, nach welchen er nicht direkt sucht. Es führt also leider nichts drum herum, alle Funktionen in diversen Fenstergrößen und mit verschiedenen (auch unlogischen) Eingaben zu testen. Als weiterer Vorteil können somit nicht nur Fehler, sondern auch inkonsequente Darstellungen oder Unklarheiten bei der Bedienung gefunden werden.

Der einfachste Weg, um Fehler zu finden, ist aber das intensive Benutzen der App.

Zur Behebung der Fehler im Frontend sind die Entwicklertools im Browser sehr hilfreich. Diese können in allen gängigen Browsern mit F12 jederzeit aktiviert werden und bieten diverse Funktionen, besonders hilfreich sind aber der Inspektor, mit welchem alle Elemente auf der Seite und deren Attribute angesehen werden können, sowie die Konsole, in welcher Fehlermeldungen ersichtlich werden.

Nach dem Beheben einiger Fehler und diversen kleinen Anpassungen ist die App nun betriebsbereit.

4 ENDPRODUKT

Das Endprodukt ist verfügbar unter <https://app.rafaelurben.ch/teamized/>. Screenshots der Web-App befinden sich im Anhang BAnhang B: Screenshots der Web-App.

5 DISKUSSION

Die finale App erfüllt alle gegebenen Ziele. Der Funktionsumfang ist zwar nicht besonders riesig, aber die definierten Basisfunktionen sind alle vorhanden. Auch bei längerem Testen trifft man nur sehr selten auf Bugs, und wenn dann auf sehr spezifische. Dank wiederholter Verbesserungen übertrifft auch das Design die ursprünglichen Erwartungen und dank Caching ist die App auch relativ schnell, bzw. hat nur kleine Ladezeiten. Qualität über Quantität hat sich in allen Fällen bewährt. Bei der mobilen Darstellung einiger Tabellen müsste unter Umständen jedoch noch eine bessere Lösung gefunden werden.

5.1 AUSBLICK

Die Struktur der App ist so aufgebaut, dass neue Funktionen in Zukunft sehr einfach ergänzt werden können. Eine Funktion, welche definitiv noch in diese App gehört, ist eine Arbeitszeitübersicht des ganzen Teams sowie eine teamübergreifende Arbeitszeitansicht für Benutzer. Auch wären «Projekte» eine mögliche Ergänzung. Damit könnte der Fortschritt eines ganzen Projektes überwacht werden, möglicherweise mit direkter Anbindung an eine To-do-Liste.

6 VERZEICHNISSE

6.1 ABBILDUNGEN

| | |
|--|----|
| Abbildung 1: Beispiel einer HTML-Datei..... | 12 |
| Abbildung 2: Beispiel einer CSS-Datei..... | 13 |
| Abbildung 3: Beispiel einer JavaScript-Datei..... | 15 |
| Abbildung 4: Ordnerstruktur von «django-teamized» (in VS Code) | 20 |
| Abbildung 5: Ordnerstruktur von «teamized» (in VS Code) | 21 |
| Abbildung 6: Beispiel der Django Template-Sprache..... | 24 |
| Abbildung 7: Beispiel einer Django view-Funktion | 24 |
| Abbildung 8: Navigationsleiste (Screenshot) | 24 |
| Abbildung 9: Beispiel einer API-Response im JSON-Format | 25 |
| Abbildung 10: Beispiel einer .ics-Datei für das iCalendar-Protokoll..... | 26 |
| Abbildung 11: Beispielcode – Funktionen des To-do-Listen-Moduls | 28 |
| Abbildung 12: Ergebnis des HTML-Beispiels | 33 |
| Abbildung 13: Startseite (Screenshot) | 37 |
| Abbildung 14: App – Startseite (Screenshot) | 37 |
| Abbildung 15: App – Teams (Screenshot) | 38 |
| Abbildung 16: App – Team (Screenshot) | 38 |
| Abbildung 17: App – Arbeitszeit (Screenshot) | 39 |
| Abbildung 18: App – Kalender (Screenshot) | 39 |
| Abbildung 19: App – To-do-Listen (Screenshot) | 40 |
| Abbildung 20: App – Ansicht auf Mobilgeräten (Screenshots)..... | 40 |

6.2 TABELLEN

| | |
|--|----|
| Tabelle 1: Liste von HTML-Tags | 34 |
| Tabelle 2: CSS-Selektortypen..... | 35 |
| Tabelle 3: Liste von CSS-Elementen | 35 |

7 EIGENSTÄNDIGKEITSERKLÄRUNG

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Mir ist bekannt, was ein Plagiat ist und ich bin mir der Konsequenzen eines Teil- oder Vollplagiates bewusst.

X

Rafael Urben
Autor

8 SCHLUSSWORT

Mit der Web-App bin ich persönlich sehr zufrieden. Ich hätte nicht erwartet, dass ich mich mit React so schnell anfreunden könnte, aber nun möchte ich es nicht mehr so schnell weggeben. Es war sehr interessant, im Bereich Frontend mal etwas ganz neues zu bauen, nur durch die Hilfe des Internets. Die Zeitplanung hat auch einigermaßen gut geklappt, für die schriftliche Arbeit hätte ich jedoch etwas besser planen und ein wenig mehr Zeit einplanen sollen.

Gerne möchte ich mich ausserdem bei all denen bedanken, welche meine App getestet und mit ihrem Feedback zu diesem Projekt beigetragen haben. Ein besonderer Dank geht an meinen Betreuer, Lukas Gerber.

ANHANG A: SPRACHEN – ERWEITERTE DETAILS

A.1 HTML

Die erste Zeile eines HTML-Dokuments, `<!DOCTYPE html>`, gibt an, dass es sich beim Dokument um HTML handelt. Somit kann auch ohne Dateinamen der Inhalt von anderen XML-artigen Sprachen unterschieden werden. Innerhalb des darauffolgenden `<html>`-Tags gibt es zwei wichtige Elemente: `head` und `body`. Im `head`-Tag werden Dinge festgelegt, welche nicht direkt auf der Seite sichtbar sind. Dazu gehört unter anderem der Seitentitel und das Favicon, welche im Browser-Tab angezeigt werden, Verlinkungen zu dazugehörenden Dateien sowie Metadaten für SEO («Search Engine Optimization»), also zum Beispiel eine Seitenbeschreibung für in die Google-Suche. Im `body`-Tag befindet sich alle Elemente, welche auf der Seite angezeigt werden und unter Umständen (weitere) `<script>`-Tags mit JavaScript-Code oder Verlinkungen zu JavaScript-Dateien.

Zeilenumschläge und die Einrückung mit Leerschlägen oder Tabulatoren sind in HTML optional, aber aufgrund der Leserlichkeit werden Sie nur selten weggelassen.

Das Beispiel aus Abbildung 1 würde in einem Browser wie folgt aussehen:



Abbildung 12: Ergebnis des HTML-Beispiels

Es besteht nur aus einem Titel (`h1`) und einem Paragraphen (`p`).

Moderne Browser können viele Fehler (wie zum Beispiel ungeschlossene Tags oder fehlende `html`, `head` oder `body`-Tags) im HTML-Code korrigieren, jedoch sollte man sich darauf nie verlassen und von Beginn an keine Tags weglassen und alle Tags korrekt schliessen.

Tabelle 1 zeigt einige weitere wichtige HTML-Tags und deren Bedeutung:

| Tag | Voller Name | Bedeutung | Beispiel |
|---------------------|-----------------|--------------------------|---|
| <code>h1-h6</code> | heading | Überschrift | <code><h1>Willkommen</h1></code> |
| <code>div</code> | division | Bereich/Abschnitt | <code><div></div></code> |
| <code>p</code> | paragraph | Paragraf (Textabschnitt) | <code><p>Guten Tag!</p></code> |
| <code>span</code> | span | Inline-Textbehälter | Ich heisse <code>John!</code> |
| <code>b</code> | bold | Fett | Ich bin <code>fett</code> . |
| <code>i</code> | italics | Kursiv | Ich bin <code><i>kursiv</i></code> . |
| <code>a</code> | anchor | Hyperlink | <code>Link</code> |
| <code>form</code> | form | Formular | <code><form method="POST">...</form></code> |
| <code>input</code> | input | Eingabefeld | <code><input type="text" name="first_name" /></code> |
| <code>button</code> | button | Knopf | <code><button onclick="alert('Hi!')">Hallo</button></code> |
| <code>ul</code> | unordered list | Ungeordnete Liste | <code>ÄpfelBirnen</code> |
| <code>ol</code> | ordered list | Geordnete Liste | <code>JohannFritz</code> |
| <code>li</code> | list item | Listenelement | <code>Frank</code> |
| <code>br</code> | break | Zeilenumschlag | <code> </code> |
| <code>hr</code> | horizontal rule | Horizontale Linie | <code><hr /></code> |
| <code>style</code> | style | CSS-Code | <code><style>h1 { color: red; }</style></code> |
| <code>script</code> | script | JavaScript-Code/Datei | <code><script src="main.js"></script></code> |
| <code>link</code> | link | Verwandte Datei | <code><link rel="stylesheet" href="style.css"></code> |
| <code>meta</code> | meta | Metadaten | <code><meta name="robots" content="noindex"></code> |
| <code>table</code> | table | Tabelle | <code><table></code> |
| <code>tr</code> | table row | Tabellenzeile | <code><tr><th>Spalte A</th><th>Sp. B</th></tr></code> |
| <code>th</code> | table heading | Tabellenzelle (Titel) | <code><tr><th>10</th><th>20</th></tr></code> |
| <code>td</code> | table data | Tabellenzelle | <code><td></td></td></td></tr></table></code> |

Tabelle 1: Liste von HTML-Tags

Dies sind alles Elemente, welche in HTML häufig anzutreffen sind. Auch für dieses Projekte werden sie fast alle verwendet.

Zusätzliche Informationen über HTML können Wikipedia ²⁹ entnommen werden.

²⁹ https://de.wikipedia.com/wiki/Hypertext_Markup_Language (Abgerufen am 23.10.2022)

A.2 CSS

Tabelle 2 zeigt einige häufig benötigte Selektortypen:

| Selektortyp | Anwendungsbeispiel | Trifft zu auf |
|--------------|-------------------------------------|---|
| Universell | <code>* {...}</code> | Alle Elemente |
| Via Typ | <code>li {...}</code> | Alle Listenelemente (<code>li</code>) |
| Via Klasse | <code>.container {...}</code> | Alle Elemente mit der Klasse <code>container</code> |
| Via Attribut | <code>input[type=date] {...}</code> | Alle Datumsauswahlfelder |
| Via ID | <code>#maintitle {...}</code> | Das Element mit der ID <code>maintitle</code> |

Tabelle 2: CSS-Selektortypen

Die ID wird in CSS mit `#...` angesprochen. Dabei handelt es sich um dasselbe wie bei Links mit `#...`, denn diese verlinken ebenfalls auf das Element mit der entsprechenden ID und ermöglichen dadurch das Hin- und Herspringen zwischen Abschnitten auf einer Webseite. (Der `#...`-Teil einer URL wird auch grundsätzlich nie an den Server übertragen.)

Tabelle 3 zeigt einige der bekanntesten CSS-Anweisungen:

| Anweisung | Bedeutung | Beispiel |
|-------------------------------|---------------------------------------|---------------------------------------|
| <code>position</code> | Position | <code>position: absolute;</code> |
| <code>width</code> | Breite | <code>width: 100%;</code> |
| <code>height</code> | Höhe | <code>height: 300px;</code> |
| <code>margin</code> | Abstand ausserhalb des Elementrahmens | <code>margin: 2px 3px 1px 4px;</code> |
| <code>border</code> | Rahmen | <code>border: 2px solid black;</code> |
| <code>padding</code> | Abstand innerhalb des Elementrahmens | <code>padding: 4px 2px;</code> |
| <code>color</code> | (Text-)Farbe | <code>color: #0000ff;</code> |
| <code>background-color</code> | Hintergrundfarbe | <code>background-color: white;</code> |

Tabelle 3: Liste von CSS-Elementen

Mit medienspezifischen Anweisungen (z. B. `@media print {...}` oder `@media (min-width: 992px) {...}`) können für bestimmte Anzeigegeräte oder -grössen verschiedene Anweisungen gegeben werden. Dies ermöglicht sogenanntes «responsive webdesign», womit sich die Seite je nach Grösse des Browserfensters variabel anpasst.

Wenn mehrere Selektoren mit einer gleichartigen Anweisung (z. B. *color*) auf ein Element zutreffen, dann gilt die Anweisung des spezifischeren ³⁰ Selektors. Somit ist es möglich, sowohl grundsätzliche Anweisungen an alle Elemente als auch spezifischere Anweisungen an spezielle Elemente zu geben.

Auch bei CSS sind Zeilenumschläge und Einrücken freiwillig.

Zusätzliche Informationen über CSS können Wikipedia ³¹ entnommen werden.

A.3 JAVASCRIPT

JavaScript ist eine sogenannte «interpretierte» Programmiersprache. Dies bedeutet, dass sie nicht zuerst in Maschinensprache übersetzt (kompiliert) werden muss, sondern dass sie vom Browser direkt als Text verarbeitet werden kann. Je nach Browser wird ein Teil des Codes auch während der Ausführung kompiliert («just-in-time compilation»), was die Ausführung verschnellern kann.

Anders als in anderen Programmiersprachen wird

Wie bei HTML und CSS sind auch bei JavaScript Zeilenumschläge und das Einrücken freiwillig. Dies ermöglicht auch das Verkleinern von Dateien.

Weiterführende Informationen über die Programmiersprache JavaScript können Wikipedia ³² oder Mozillas Entwicklerportal ³³ (Englisch) entnommen werden.

³⁰ <https://developer.mozilla.org/en-US/docs/Web/CSS/Specificity> (Abgerufen am 23.10.2022)

³¹ https://de.wikipedia.org/wiki/Cascading_Style_Sheet (Abgerufen am 23.10.2022)

³² <https://en.wikipedia.org/wiki/JavaScript> (Abgerufen am 23.10.2022)

³³ <https://developer.mozilla.org/en-US/docs/Web/javascript> (Abgerufen am 23.10.2022)

ANHANG B: SCREENSHOTS DER WEB-APP

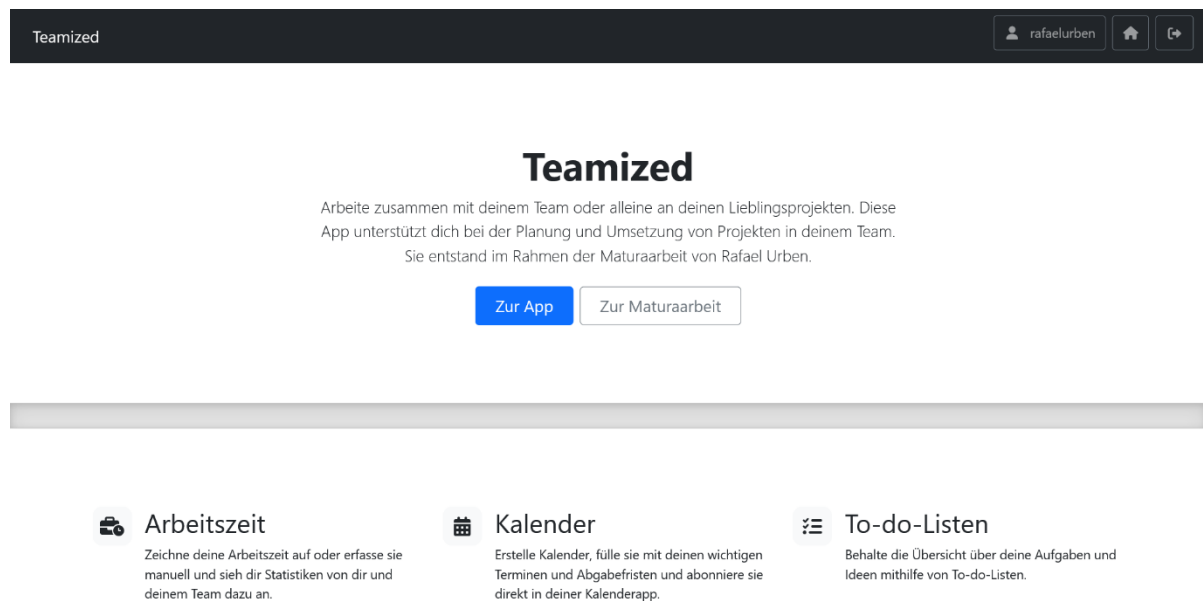


Abbildung 13: Startseite (Screenshot)

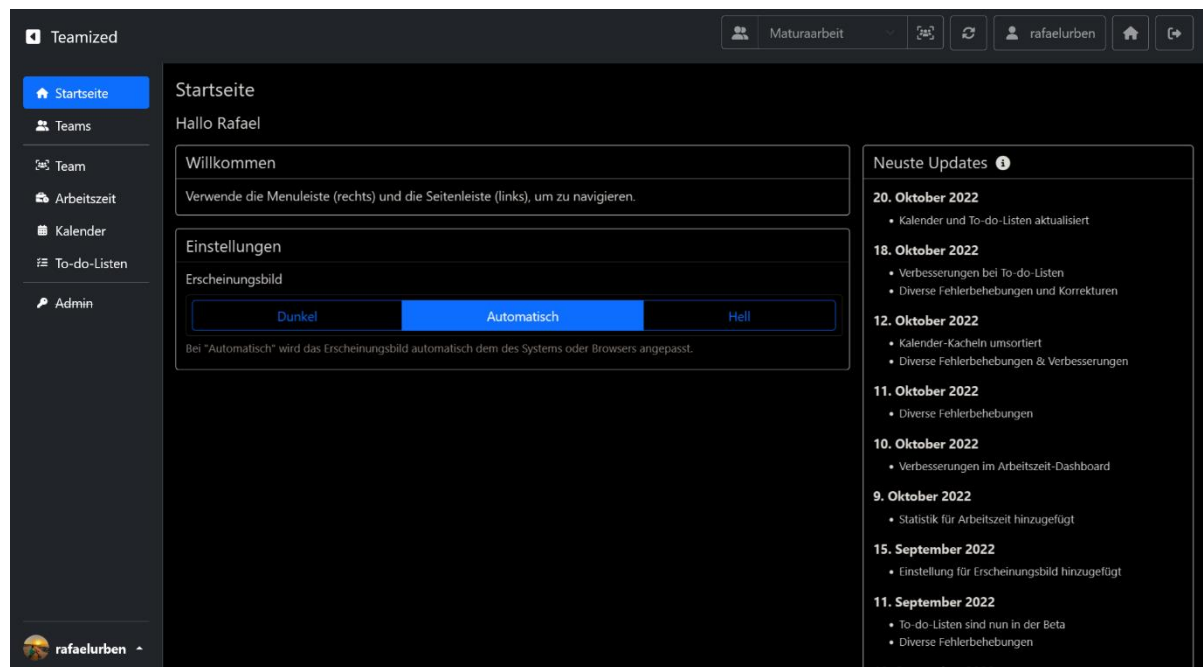


Abbildung 14: App – Startseite (Screenshot)

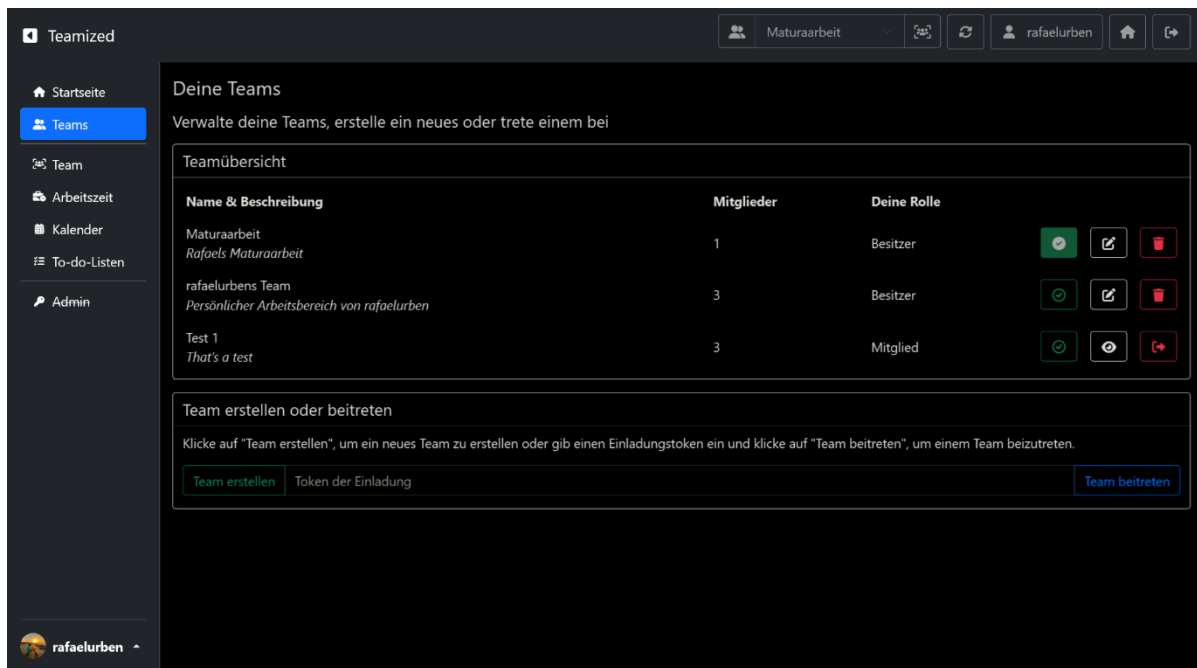


Abbildung 15: App – Teams (Screenshot)

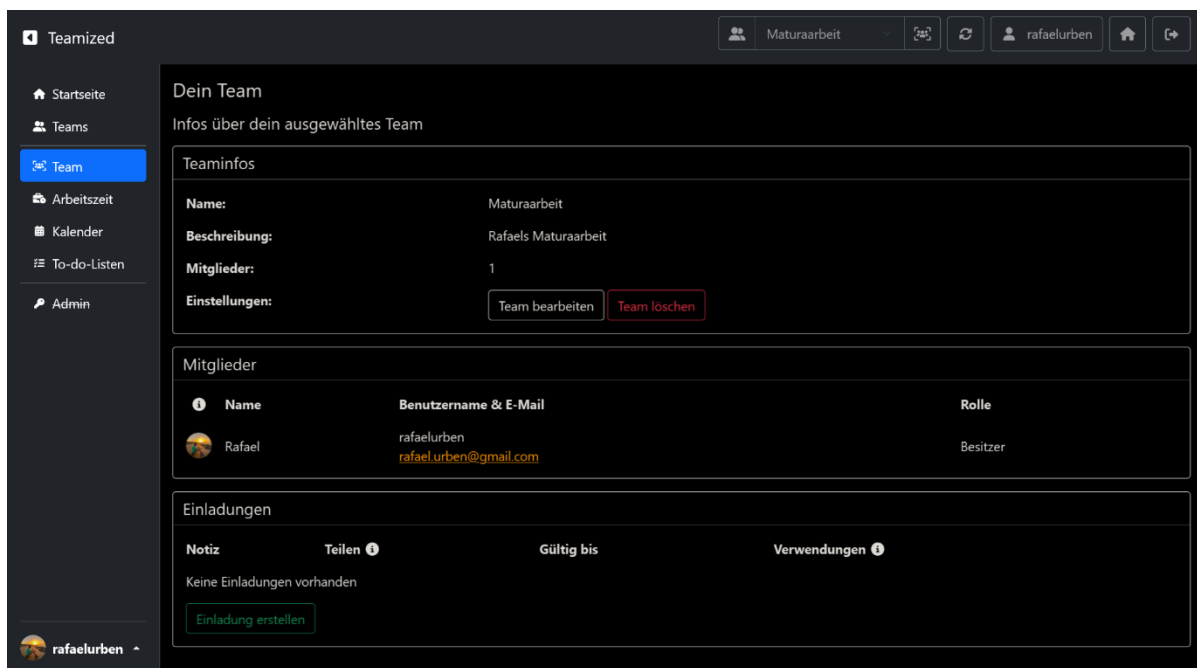


Abbildung 16: App – Team (Screenshot)

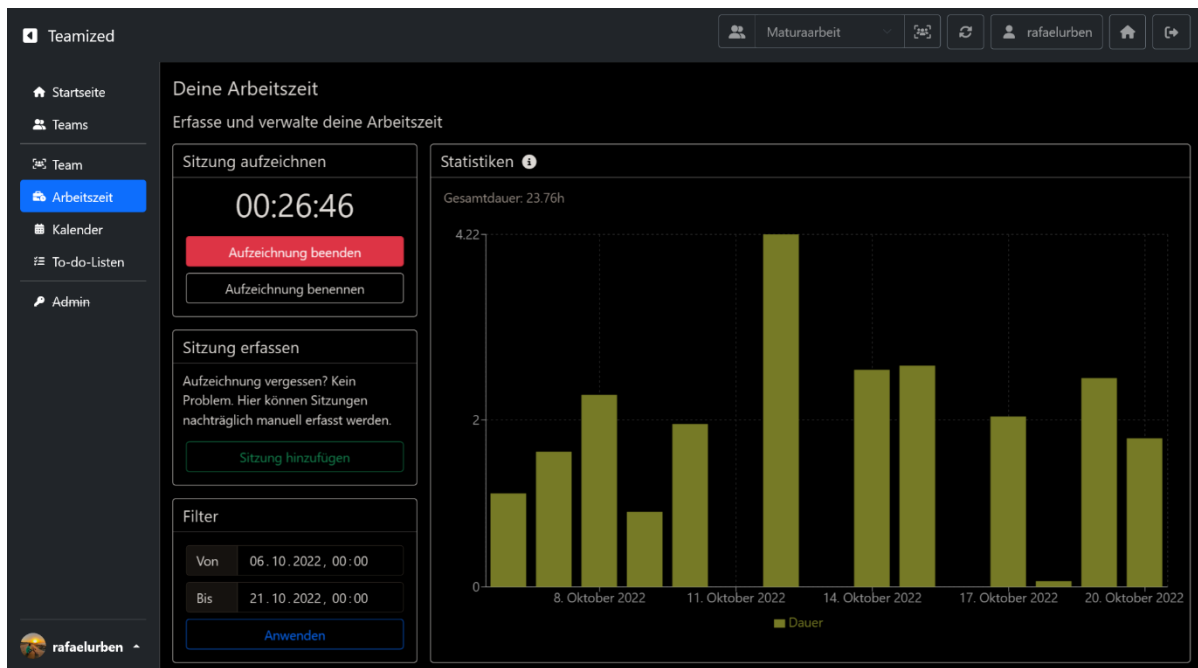


Abbildung 17: App – Arbeitszeit (Screenshot)

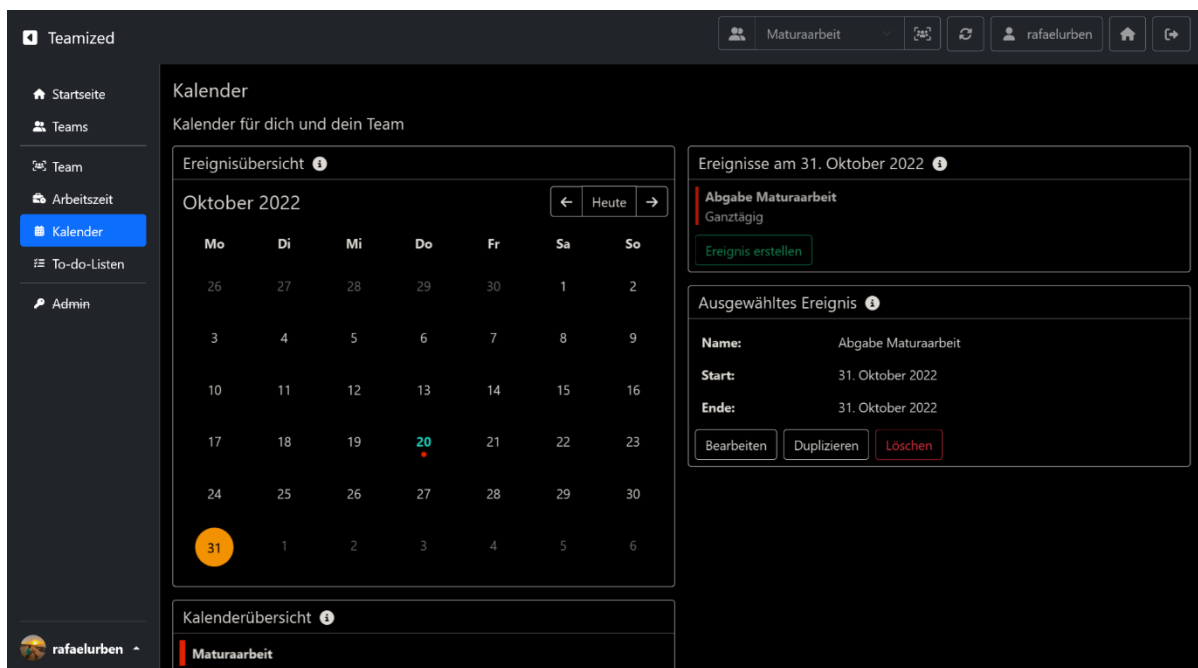


Abbildung 18: App – Kalender (Screenshot)

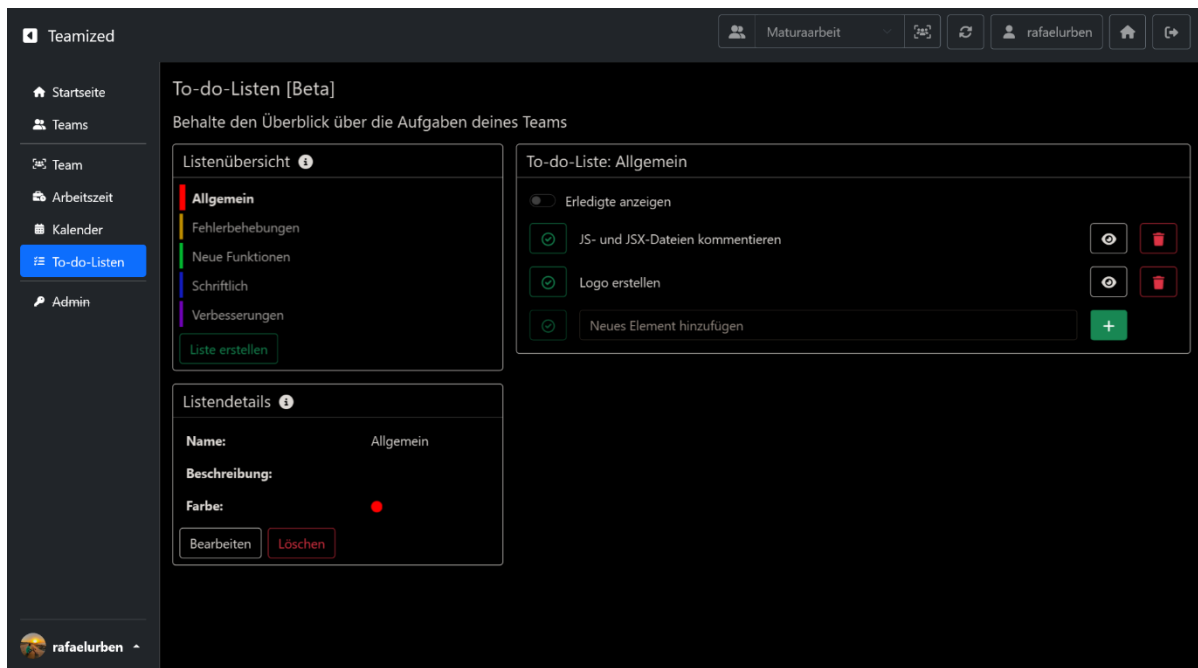


Abbildung 19: App – To-do-Listen (Screenshot)

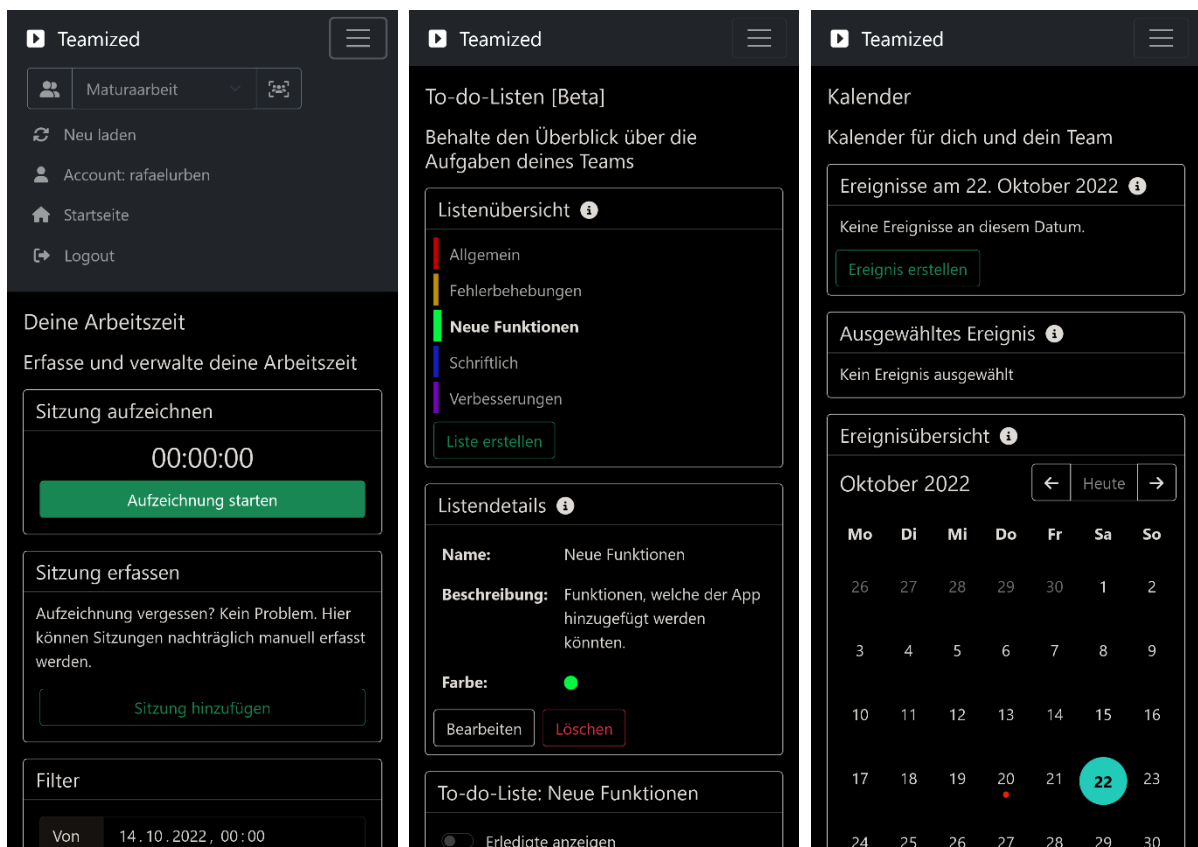


Abbildung 20: App – Ansicht auf Mobilgeräten (Screenshots)

(Bild links mit geöffneter Navigationsleiste)