Hochschule der Medien

Nobelstraße 10

70569 Stuttgart

Sommersemester 2021

Dokumentation

TraveLinked

Modul:	Software-Projekt
Betreuer:	Prof. Walter Kriha
EDV-Nr:	113400

Gruppe:	Vu, Son Hai (sv048)
	Heinrich, Leon (lh108)
	Blumenstock, Silas (sb262)
	Heim, Manuel (mh312)
	Dubiel, Sebastian (sd114)

Studiengang:	Medieninformatik
Abgabeort und –datum:	Stuttgart, 30.07.2021

Inhaltsverzeichnis

1 Einleitung

Unser Projekt „TraveLinked“ wurde im Rahmen des Moduls „Software-Projekt“ im Sommersemester 2021 realisiert. Es handelt sich dabei um eine webbasierte Anwendung, die von Grund auf basierend auf dem MEAN-Stack implementiert wurde.

Ziel und Zweck des Projekts ist die Vertiefung der Kenntnisse im Bereich der Softwareentwicklung.

Der Zeitrahmen beträgt ca. vier Monate. Die fertige Anwendung soll dann am 01.07.2021 an der MediaNight präsentiert werden.

Unsere Gruppe besteht aus fünf MI-Studenten im 6. Semester.

Mit „TraveLinked“ hat der Nutzer die Möglichkeit, seine eigenen individuellen Touren (Individueller Tourismus) zu erstellen, die man mit anderen Benutzern teilen kann.
Eine Tour besteht aus verschiedenen ausgewählten Sehenswürdigkeiten, die man gerne besichtigen möchte.
Das Besondere dabei ist, dass man seine eigenen Sehenswürdigkeiten – sogenannte Geheimtipps – anlegen und diese mit der Community teilen kann.
Zusätzlich kann man einer Tour weitere Aktivitäten („Things to do“) hinzufügen, sowie eigene Bilder, die in einer Fotogalerie angezeigt werden.
Touren können von anderen Benutzern bewertet, gespeichert und kommentiert werden.
Dabei soll auch der soziale Aspekt miteinbezogen werden („Social Tourism“). Das bedeutet, dass man ebenfalls die Möglichkeit hat, sich mit anderen Benutzern zu vernetzen, um sich gegenseitig über bestimmte Touren auszutauschen.
Des Weiteren kann man auch als „Local Guide“ agieren und so Empfehlungen zu seinem Heimatort geben. Für diese Funktionalitäten benötigt man einen eigenen Benutzeraccount.

2 Funktionen

2.1 Grundlegende Funktionen

Die Seite bietet einen Überblick an verschiedenen Touren, die von der Community erstellt wurden. Dabei stehen Touren, die am besten bewertet wurden, an vorderster Stelle auf der Hauptseite. Zudem werden auch aktive Touren angezeigt, das heißt Touren die zu diesem Zeitpunkt stattfinden.

Die Startseite beinhaltet einen großen Hero Banner, der die Aufmerksamkeit des Benutzers erregen soll. In dem Hero Banner befindet sich eine Suchfunktion, mit der man nach Orten auf der ganzen Welt suchen kann. Man kann entweder nach Touren suchen oder nach Local Guides.

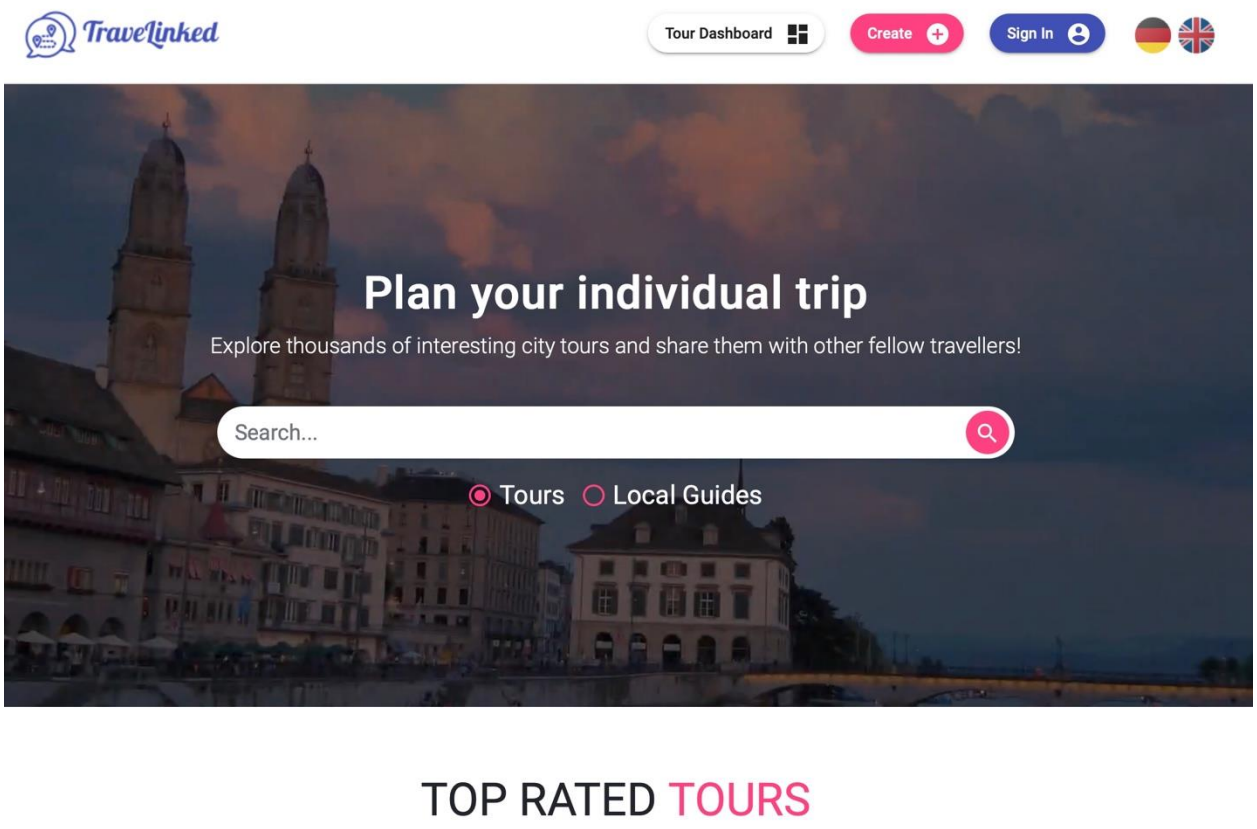

Abbildung 1: Startseite

Die Ergebnisse der Suche sind die Touren im jeweiligen Suchgebiet. Touren die öffentlich sind, können jederzeit eingesehen werden.

Eine Tour kann viele verschiedene Sehenswürdigkeiten beinhalten, die sogenannten Points of Interests. Solche Sehenswürdigkeiten können auch selbst erstellt werden, die dann als Geheimtipps bezeichnet werden.
Die Anwendung profitiert daher von der wachsenden Community, die maßgebend zur Weiterentwicklung des Angebots beiträgt.

Für erweiterte Funktionen benötigt man einen eigenen Account.

2.2 Erweiterte Funktionen

Die Hauptmerkmale dieser Anwendung liegen in der Erstellung, Bearbeitung, sowie dem Folgen und Liken von Touren. Diese Touren können von anderen Benutzern gespeichert bzw. bewertet werden.

Es stehen zahlreiche Sehenswürdigkeiten zur Verfügung, die als Basis für eine Tour dienen. Mit Hilfe der Google API stehen einem eine große Anzahl an verschiedenen Sehenswürdigkeiten auf der ganzen Welt zur Verfügung.
Des Weiteren hat man die Möglichkeit seine eigene individuelle Sehenswürdigkeit zu erstellen, die dann anderen Nutzern zur Verfügung steht.
In der Tour-Übersicht, sieht man seine selektierten Sehenswürdigkeiten, die ebenfalls auf der Karte von Google anzeigt werden. Die Sehenswürdigkeiten lassen sich verbinden, sodass man eine empfohlene Route angezeigt bekommt. Diese Route wird von Google berechnet, wofür der A*-Algorithmus benutzt wird.
Eine Besonderheit unserer Anwendung ist die Vernetzung mit anderen Benutzern. So hat man die Möglichkeit, sich in einem Chat mit anderen Reisenden zu vernetzen. Ebenso kann man private Nachrichten an andere senden. Andere Benutzer können ebenfalls Kommentare zu einer Tour hinterlassen und diese bewerten. Die am besten bewerteten Touren werden als Highlight auf der Startseite angezeigt. Des Weiteren kann man auch anderen Benutzern folgen, was den sozialen Aspekt noch verstärken soll.
Jeder Benutzer hat einen eigenen Benutzeraccount, in dem er seinen Standort festlegen kann. Dieser wird verwendet, wenn man als „Local Guide“ agieren möchte. Das bedeutet, man kann anderen Benutzern Empfehlungen, Tipps oder Hilfe bezüglich seiner Heimatstadt geben, falls Interesse besteht.
Mit einem Benutzeraccount hat man Zugang zu seinem persönlichen Bereich (Tour Dashboard), wo man Einsicht zu seinen selbst erstellten Touren, Sehenswürdigkeiten und bewerteten Touren hat.
Jeder Benutzer hat zudem auch sein eigenes Profil, in dem die wichtigsten Informationen zu sehen sind, wie die Anzahl der Follower, die Touren, sein eigenes Profilfoto, den Heimatstandort und eine kleine Kurzvorstellung.

3 Implementierung

Nachfolgend werden die Implementierungsschritte, sowie die Features, die wir implementiert haben, genauer erläutert.

3.1 Google API

Wir verwenden die Google API zum Laden der Standard-Sehenswürdigkeiten. Google bietet hierfür verschiedene APIs an. Wir benutzen folgende Services: die Google Places API, die Google Maps API und die Google Direction API.

Wir verwenden hauptsächlich die Places API, um Informationen wie Geokoordinaten, Städtenamen und Sehenswürdigkeiten zu laden. [1]
Dazu wird ein HTTP-Request an folgende Adresse geschickt:

```
https://maps.googleapis.com/maps/api/place/
```

Durch Anhängen von Parametern lassen sich somit verschiedene Requests senden. Um beispielsweise Informationen für eine Stadt zu laden, wird dieser Aufruf hier durchgeführt:

```
https://maps.googleapis.com/maps/api/place/textsearch/json?query=Tokyo&key=“APIK“
```

Dies ist eine „Text Search Request“, der als Parameter einen String annimmt, in dem Fall die Stadt „Tokyo“. Als weiterer Parameter ist zwingend der API-Schlüssel anzugeben.
Falls die Suche ein Erfolg war, schickt Google eine Response im JSON Format zurück, in dem Informationen wie Städtenamen, Geokoordinaten usw. enthalten sind.
Durch das Anhängen von spezifischeren Query-Parametern lassen sich somit Sehenswürdigkeiten herausfiltern:

```
https://maps.googleapis.com/maps/api/place/textsearch/json?query=tokyo+point+of+interest
```

Im obigen Beispiel haben wir „point+of+interest+“ angehängt, um die Sehenswürdigkeiten für Tokyo zu erhalten.

[1] https://developers.google.com/maps/documentation/places/web-service/overview

3.2 Tour Logik

Ein angemeldeter Benutzer kann beliebig viele Touren erstellen. Eine Tour besteht aus einer eindeutig identifizierbaren ID, einem Namen, der ID des Benutzers, um die Tour eindeutig einzuordnen, sowie weitere Metadaten wie Tourbeschreibung, Sehenswürdigkeiten, Datum usw.

Eine Tour kann man in der „Tour Creation“ erstellen, wo man verschiedene Daten angeben kann, z.B. wie die Tour heißen soll, der Zeitraum, wann eine Tour stattfinden soll, eine Tourbeschreibung, ein Titelbild, sowie die verschiedenen Sehenswürdigkeiten, die man wahlweise auf der interaktiven Karte auswählen kann oder aus einer Liste.

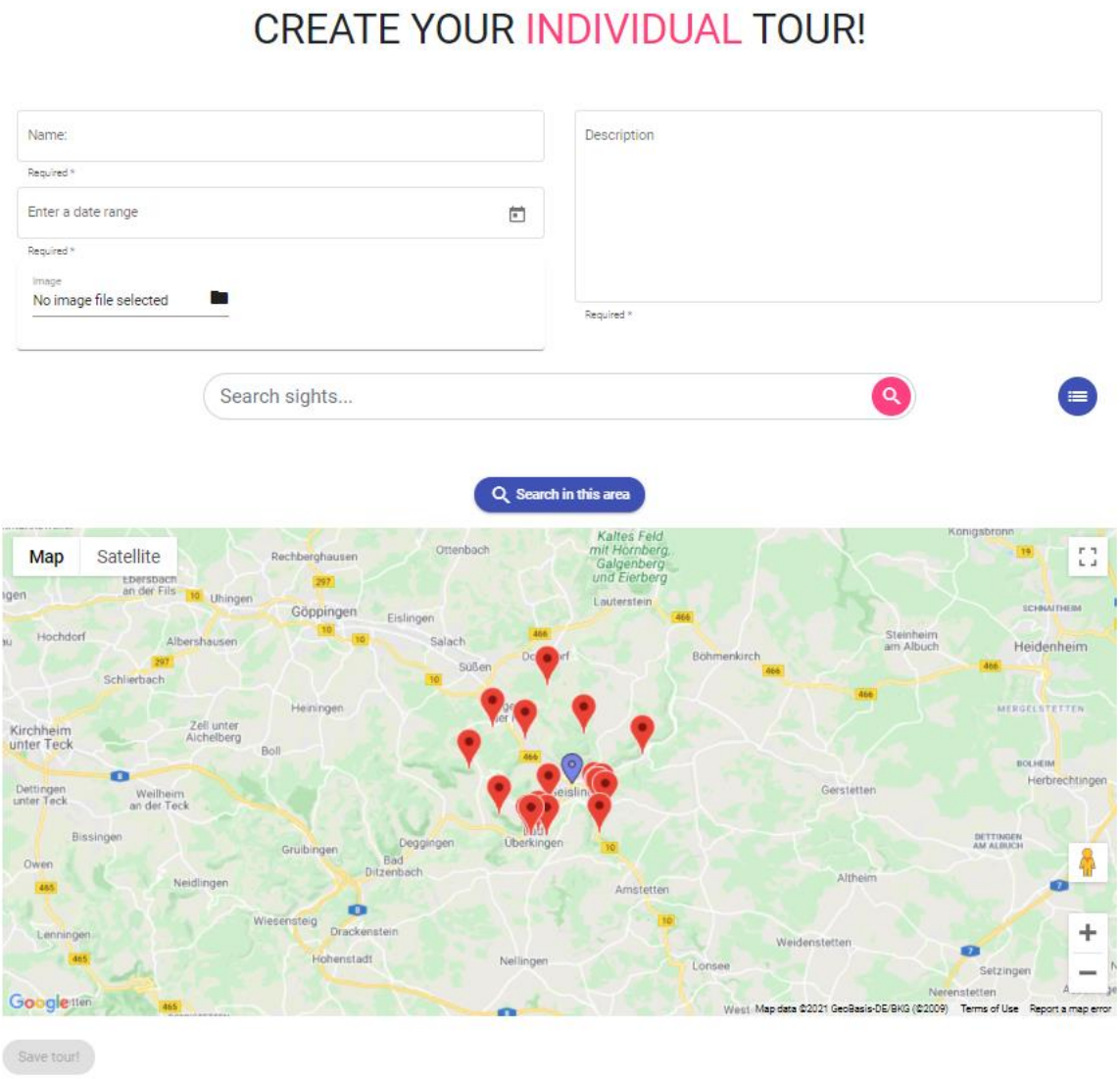

Abbildung 2: Tourerstellung

Beim Anlegen einer Tour, ruft das Frontend unseren Backend Service auf, der wiederum die Daten in unsere MongoDB Datenbank persistiert. Wir haben hierzu verschiedene Endpoints für die Touren angelegt. Das wären zum Beispiel die typischen CRUD-Befehle wie Tour anlegen (createTour), Touren ausgeben (getTourByID, getTopTours, getAllTours usw.), Touren bearbeiten (editTour), sowie Touren löschen (deleteTour).

Auf unserer Startseite befindet sich der Abschnitt „Top Rated Tours“, wo wir die am besten bewerteten Touren anzeigen. Diese holen wir mit der getTopTours-Methode im Frontend, der einen HTTP-Request an das Backend sendet. Das Backend führt wiederum eine Datenbankabfrage aus.
Die Datenbank gibt im Folgenden alle Touren zurück, sortiert nach absteigender Anzahl der Likes, die die Tour erhalten hat.
Unsere Suchfunktion auf der Hauptseite führt ebenso eine Anfrage an das Backend durch. Jede Tour besitzt Koordinaten (Längengrad, Höhengrad) im Format eines GeoJSONs. [2] GeoJSON ist ein Format, um geographische Daten zu repräsentieren.

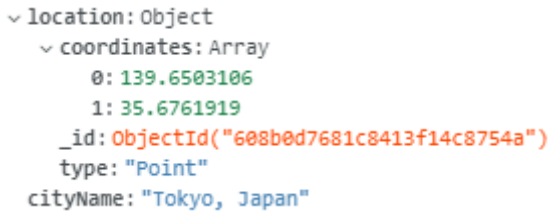

Abbildung 3: Koordinaten einer Tour

Die Suche ruft die Methode `getTourByLocation` auf, die Koordinaten (Longitude, Latitude) erwartet. Die Koordinaten werden an das Backend gesendet und eine Datenbankabfrage durchgeführt wird. Dabei durchsucht sie die Datenbank nach den entsprechenden Koordinaten in einem bestimmten Radius und gibt alle Touren zurück, die sich in diesem Radius befinden.

Des Weiteren befindet sich auf unserer Startseite noch der Abschnitt „Active Tours“, wo Touren angezeigt werden, die zum aktuellen Zeitraum aktiv sind. Jede Tour hat ein Start- und Enddatum. Mit Hilfe einer Datenbankabfrage, können wir alle Touren zurückgeben, wo die aktuelle Zeit in diese Spanne fällt.

[2] https://geojson.org/

```
const getActiveTours = async (req, res) => {
   try {
      const activeTours = await TourDB.Tour.find({
         startDate: { $lt: moment().format() },
         endDate: { $gte: moment().format() },
      }).exec();
      res.status(200).send(activeTours);
   } catch (error) {
      return res.status(500).send({ errorMessage: error });
   }
};
```

Die Abfrage schaut in der Datenbank alle Touren an, wo das Startdatum kleiner und das Enddatum größer als das aktuelle Datum ist.

3.3 Route-Guards

Um unsere Seiten vor unerlaubtem Zugriff zu schützen, verwenden wir sogenannte Route-Guards, welche ein Feature von Angular sind. Route-Guards können im app-routing.module.ts File bestimmten Unterseiten der Webanwendung zugeteilt werden, wodurch sichergestellt wird, dass auf diese Seiten nur eingeloggte Nutzer Zugriff haben. Somit wird sichergestellt, wenn man eine Seite auf direktem Wege über die URL aufruft, dass der Zugriff für unbefugte Nutzer nicht sichtbar ist.

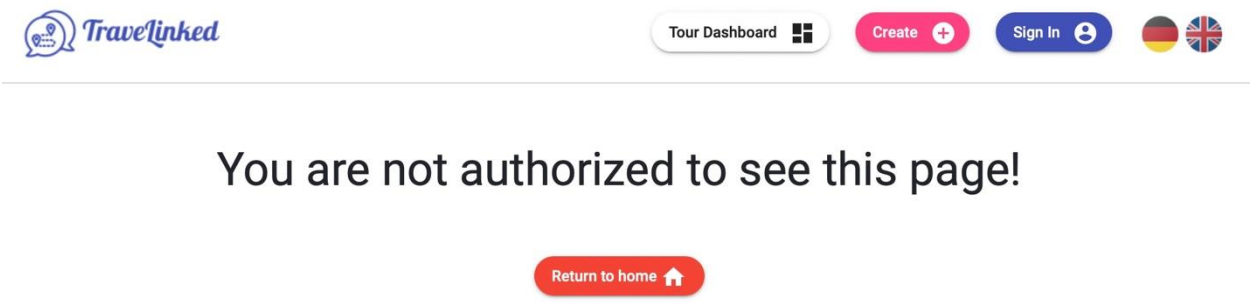

Abbildung 4: Fehlende Berechtigung

3.4 Mobile Responsiveness

Unsere Anwendung ist ebenfalls für die mobile Ansicht angepasst. Hierzu verwenden wir Bootstrap, ein CSS-Framework von der Firma Twitter. Mit Bootstrap kann man schnell und einfach responsive Anwendungen erstellen. [3] Zudem bietet Bootstrap einheitliche Komponenten wie Buttons, Formulare, Tabellen usw. Wir haben hauptsächlich das Grid-System von Bootstrap verwendet. Damit wird die Anwendung individuell angepasst, je nachdem wie groß die Fenstergröße ist. Dies wird mit verschiedenen Breakpoints gesetzt. Das Grid-System besteht aus zwölf Spalten. Anhand der Breakpoints wird der Inhalt dementsprechend auf die Spalten angepasst. Für die Tour-Ansicht auf der Startseite haben wir beispielsweise auf der Desktop-Ansicht drei Touren in einer Zeile. Jede Tour beansprucht vier Spalten. Ab einer gewissen Größe, wenn die Fenstergröße einen bestimmten Breakpoint erreicht, beansprucht eine Tour dann die vollen zwölf Spalten. Mit `class=”row”` wird die Zeile definiert und mit `class=”col”` die Spalte.

```
<div class="row tours-view" *ngIf="!isLoading">
      <div class="col-md-4" *ngFor="let tour of activeTours.slice(0, 3)">
```

Dies bedeutet, dass seine Tour zunächst wie erwähnt vier Spalten beansprucht und ab einer gewissen Größe dann zwölf Spalten. Eine Übersicht zu den Breakpoints befindet sich auf der offiziellen Bootstrap Dokumentation. [4]

[3] https://getbootstrap.com/
[4] https://getbootstrap.com/docs/5.0/layout/grid/#grid-options

3.5 Chat Funktionen

Um eine Kommunikation mit anderen Benutzern zu ermöglichen, haben wir Websockets verwendet. Websockets ermöglichen eine bidirektionale Verbindung zwischen unserer Webanwendung und dem Webserver und basieren auf dem TCP-Protokoll.

Wir verwenden hierfür die socket.io Library. [5] Socket.io ermöglicht eine bidirektionale und eventbasierte Kommunikation in Echtzeit. Socket.io verwendet Räume (Chatrooms). Das bedeutet, jedes Mal, wenn ein neuer Benutzer einen Raum betritt, wird eine Socket-Verbindung aufgebaut. Wenn der Benutzer eine Nachricht im Frontend versendet, sendet (emitted) er gleichzeitig ein Event. Das Event, eine Nachricht zu versenden, sieht bei uns so aus:

```
socket.emit('message', data)
```

Der erste Parameter ist der Name des Events und der zweite Parameter die Daten, die versendet werden sollen. Auf solch ein Event kann socket.io im Backend reagieren. Dazu muss er auf das Event „message“ lauschen:

```
socket.on('message', (data) => {
      //-> emit new message
      io.in(data.room).emit('new message', {
         user: data.user,
         message: data.message,
      });
   });
```

Im obigen Code-Beispiel, reagiert socket.io auf das Event „message“, falls solch ein Event emitted wurde. Er nimmt die Daten entgegen und kann sie weiterverarbeiten. In dem Fall emitted er ebenfalls ein Event („new message“) mit den entsprechenden Daten. Daraufhin kann das Frontend auf dieses Event reagieren und die neue Nachricht in dem entsprechenden Raum in der UI anzeigen.

Um private Nachrichten zu versenden, verwenden wir die ID der Benutzer, um eindeutige Räume zu erstellen. Jede private Nachricht wird bei uns gespeichert, um so das Laden des Chatverlaufs zu ermöglichen.

[5] https://socket.io/

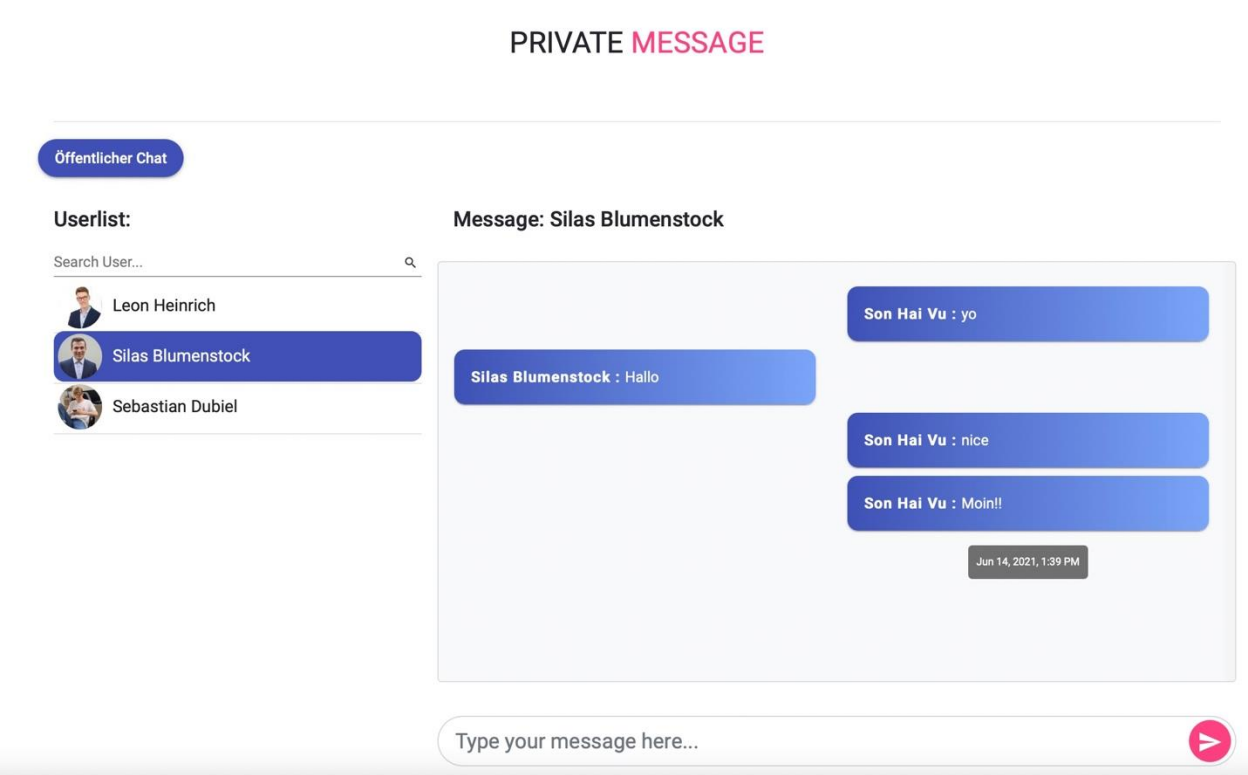

Abbildung 5: Privatnachrichten

Im obigen Bild (Abbildung 5) ist die Ansicht für Privatnachrichten zu sehen. Links kann man nach Benutzern suchen. Benutzer, mit denen man bereits Kontakt hatte, werden in der Liste gespeichert. Die Liste wird nach Datum, d.h. absteigend nach den aktuellsten Nachrichten, sortiert.

Die Benutzer haben zudem die Möglichkeit, über die Profilansicht eines anderen Benutzers, den jeweiligen Benutzer zu kontaktieren. Der Empfänger sieht in seinen Nachrichten ebenfalls eine Benachrichtigung, dass er eine ungelesene Nachricht hat.

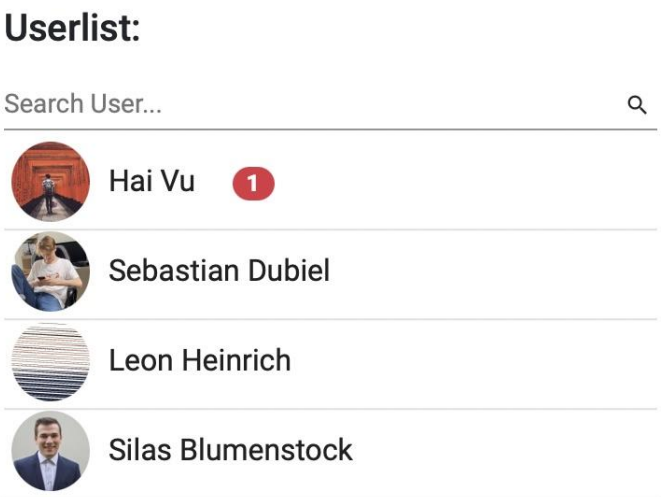

Abbildung 6: Liste der bereits kontaktierten Benutzer mit einer ungelesenen Nachricht

3.6 Mehrsprachigkeit

TraveLinked unterstützt Mehrsprachigkeit, so hat der Benutzer derzeit die Möglichkeit, die Seite entweder auf Deutsch oder auf Englisch anzeigen zu lassen. Wir verwenden hierfür eine spezielle Angular Library, die sogenannte „ngx-translate“. [6] Mit dieser Erweiterung ist es möglich, die Anwendung in mehreren Sprachen auszugeben. Dazu müssen die Begriffe vorerst in einer JSON-Datei definiert werden. Für jede Sprache wird die entsprechende JSON-Datei angelegt. In unserem Fall, für die Sprachen Deutsch und Englisch, sind die Wörter in der en.json und in der de.json Datei definiert. Die zu übersetzenden Begriffe sind letztendlich einzelne Strings, die in einem Key-Value-Pair definiert sind. Für die Überschrift auf der Startseite wird folgender Wert verwendet: "TITLE": "Plan your individual tour". In der HTML kann man auf diesen Key zugreifen und mit der „translate pipe“ diesen String in der ausgewählten Sprache anzeigen. Die Syntax, um auf diesen String zuzugreifen, sieht demnach so aus: {{'TITLE' | translate}}.

Somit lässt sich bei Bedarf weitere Sprachen hinzufügen, da die Keys für jede Sprache identisch ist und man lediglich nur den Value anpassen muss. Die Sprachdateien liegen im i18n Ordner. Die Bezeichnung i18n ist die Abkürzung für die Internationalisierung in der Softwareentwicklung, welche den Zweck hat, ein Programm so zu gestalten, dass es leicht an andere Sprachen und Kulturen angepasst werden kann.

[6] https://github.com/ngx-translate/core

3.7 Clean Code

Wir haben viel Wert auf einen sauberen Code gelegt. Für die Einhaltung von Clean Code haben wir einige Guidelines festlegt, wie der Code auszusehen hat. Mit Hilfe von verschiedenen Tools wird sichergestellt, dass der Code einen guten Stil hat. Wir verwenden hierfür Prettier, was unseren Code ordentlich formatiert und den Code mit Semikolons abschließt, auch wenn dies in JavaScript nicht zwingend notwendig ist, haben wir darauf geachtet. Mit Hilfe eines Pre-Commit, wird sichergestellt, dass der Code automatisch formatiert wird, bevor er dann tatsächlich comittet wird. Des Weiteren haben wir ein Skript definiert, welches durch alle Files durchgeht und sie entsprechend formatiert.

Ebenso achten wir auf die Performance, Best Practices und auf den SEO-Wert. Unser Code ist daher mit verschiedenen Metadaten gekennzeichnet. Gemäß dem W3C-Standard besitzen unsere HTML-Elemente zusätzliche Attribute für die Accessibility. Das wären zum Beispiel die alt-Attribute beim img-Element oder die aria-label-Attribute bei Buttons, wenn sie keinen inhaltlichen Text besitzen. Dies ist wichtig für Screenreader und steigert so die User Experience, insbesondere bei eingeschränkten Benutzern.

Des Weiteren sind unsere Dateien wie Bilder, Videos usw. komprimiert, um deren Speicherverbrauch zu reduzieren.

Nachfolgend der Lighthouse Report von Google bezüglich der Performance unserer Website:

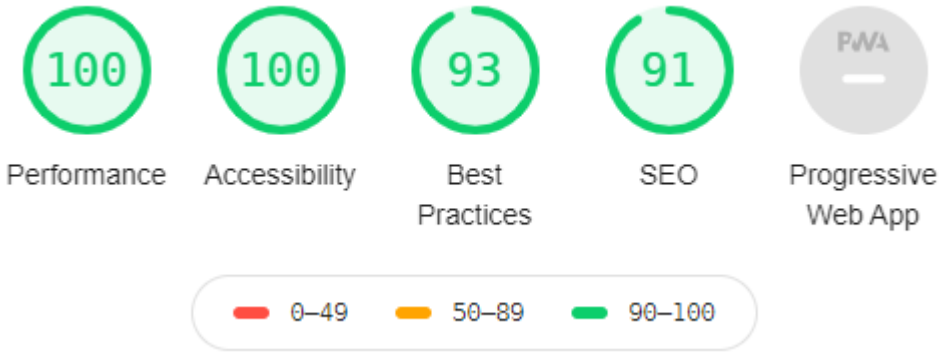

Abbildung 7: Google Lighthouse Report

3.8 HTTP-Interceptor

Zur Behandlung von Ausnahmen bei HTTP-Requests, insbesondere bei abgelaufenen Tokens wurde ein HTTP-Interceptor implementiert. Ein HTTP-Interceptor ist eine, im Angular-Framework enthaltene Funktionalität, die durch die Implementierung des HTTP-Interceptor-Interface eingebunden werden kann. Dieser fängt HTTP-Requests ab und ermöglicht eine Modifikation eines Requests, sowie das Abfangen von Antworten auf Requests. Das Abfangen eines Requests wird durch das Überschreiben der intercept()-Methode umgesetzt. Mit Hilfe der catchError()-Funktion aus RxJS kann ein Fehler bei einem Requests abgefangen werden, Fehler mit Fehlercode 401 und entsprechender Fehlermeldung sind Fehler, die durch abgelaufenen JSON Web-Tokens entstanden sind, für diese wurde eine Refresh-Token Funktionalität implementiert. Im Abschnitt 4.3 wird detaillierter auf die Aktualisierung des Tokens, sowie auf die generelle Verwendung von JSON Web-Tokens in unserer Anwendung eingegangen. Ein erneuerter Token kann gespeichert werden und dem Header eines Requests hinzugefügt werden, wobei er in unserer Applikation einer Kopie des fehlgeschlagenen Requests hinzugefügt wurde. Durch die Verwendung eines HttpHandlers in der intercept()-Methode kann ein HTTP-Request in einen Stream aus HttpEvents transformiert werden. Dieser Stream ermöglicht durch die Nutzung der handle()-Methode eine Ausführung weiterer Requests, somit kann der kopierte, fehlgeschlagene Request mit einem aktualisierten Header im Token ausgeführt werden. Der HTTP-Interceptor wird also zur Fehlerbehandlung bei abgelaufenen Tokens eingesetzt, wobei der fehlgeschlagene Request automatisch mit erneuertem Token erneut ausgeführt wird.
Ein maßgeblicher Vorteil der Nutzung eines HTTP-Interceptors ist es, dass eine Ausnahmebehandlung an einer zentralen Stelle stattfindet und diese somit nicht an jeder Stelle, an der ein Request ausgeführt wird nötig ist.
In zukünftigen Entwicklungsschritten soll der Interceptor auch für die generelle Fehlerbehandlung durch die Verwendung einer ErrorHandling-Klasse genutzt werden, außerdem soll der Interceptor für die Modifikation des Standard-Headers aller Requests verwendet werden.

```
intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
   return next.handle(req).pipe(
      catchError((err) => {
         if (err.status === 401 && err.error.errorMessage === 'Token has been expired') {
            return this.handleRefresh(req, next);      Son Hai Vu, 3 months ago • Ran pre
         } else {
            return throwError(err);
         }
      })
   );
}
```

Abbildung 8: Implementation der intercept-Methode um Fehler durch abgelaufene Tokens zu behandeln

4 Security

4.1 Authentifikation

In der Anwendung hat man die Möglichkeit, seinen eigenen Benutzeraccount anzulegen. Dazu muss man sich zunächst mit Namen, E-Mail und Passwort registrieren. Ein neu angelegter Account hat zunächst den Status „Pending“, das bedeutet, dass der Account zunächst freigeschaltet werden muss. Nach der Registrierung erhält der Benutzer eine E-Mail, wo er aufgefordert wird, seinen Account zu aktivieren. Die genaue Funktionsweise wird im nächsten Abschnitt erläutert. Für die Registrierung muss der Benutzer außerdem ein sicheres Passwort angeben, das eine bestimmte Länge aufweisen und aus einer Kombination von Zahlen und Zeichen bestehen muss.

Dies verringert das Risiko durch Brute-Force Angriffe, bei denen ein Passwort durch Erraten geknackt werden kann.

4.2 Passwort Hashing

Das Speichern von Passwörtern im Klartext stellt eine große Sicherheitslücke dar. Die Passwörter werden in unserer Datenbank daher nicht als Klartext gespeichert, sondern verschlüsselt. Wir verwenden hierfür bcrypt, was eine kryptographische Hashfunktion ist, die für das Speichern und Hashen von Passwörtern zuständig ist. Bcrypt verwendet eine sogenannte Salt, was in der Kryptographie als eine zufällig gewählte Zeichenfolge bezeichnet wird. Diese Zeichenfolge wird an einen gegebenen Klartext vor der Eingabe in eine Hashfunktion angehängt, um so die Entropie zu erhöhen. In der Kryptographie stellt die Entropie ein Maß für die „Unordnung“ in Texten dar. Eine Hashfunktion ist eine Einwegfunktion, das bedeutet, dass in dem Fall ein gehashtes Passwort nicht mehr umzukehren ist, also dessen Klartext nicht mehr wiederherzustellen ist. Mit bcrypt können wir die Anzahl der Runden festlegen, wie oft er den Salt Prozess durchführen soll. Generell kann man sagen, je höher, desto sicherer, was jedoch auch zu einer längeren Laufzeit führt. Das gehashte Passwort könnte dann folgendermaßen aussehen: "$2b$10$11Q7kOSVOSz0GwjET4/NhePP896vSsKFI4rR5RRYCwEnFQk1scSN2".

4.3 Nutzung von JSON Web-Tokens

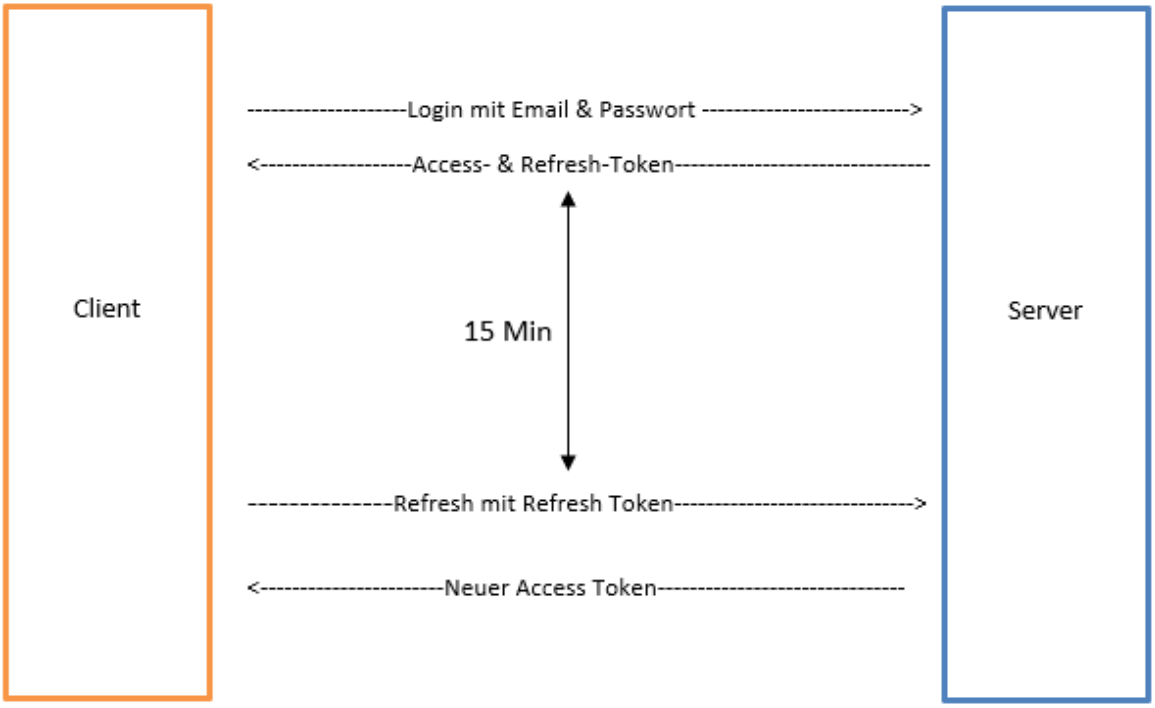

Abbildung 9: Authentifizierung mit JSON-Web Tokens

Zur Authentifizierung haben wir das in Abbildung 9 abgebildete Prinzip verwendet. Um auf gesicherte Daten eines Nutzers (z.B. Touren) zugreifen zu können, ist es nötig, den verwendeten Zugriff zu authentifizieren. Damit nicht bei jedem Zugriff eine Angabe von E-Mail und Passwort nötig ist und diese nicht im Client gespeichert werden müssen, haben wir das Prinzip der Token-Authentifizierung genutzt. Dabei enthält ein Nutzer nach dem Login eine Antwort, die einen Access-, sowie ein Refresh-Token beinhaltet. Mit Hilfe des Access Tokens kann der User auf für ihn freigegebene, geschützte Daten zugreifen, wobei keine Angabe von E-Mail und Passwort mehr nötig ist.

Um das Sicherheitsrisiko durch gestohlene Access Tokens zu verringern, haben Access-Tokens eine maximale Gültigkeitsdauer von 15 Minuten. Danach muss ein neuer Access Token angefordert werden, wobei ein Request mit Angabe des Refresh-Tokens nötig ist. Die konkret genutzten Tokens sind bei uns JSON-Web Tokens. Diese sind im JSON-Format aufgebaut und bestehen aus Header, Payload und Signature. Der Header beinhaltet z.B. Informationen über verwendete Algorithmen der Verschlüsselung, im Payload können beliebige Daten, z.B. User Informationen enthalten sein. Die Signatur dient zur Überprüfung der Gültigkeit eines Tokens und verhindert, dass dieser verändert werden kann. Tokens werden durch das Base64-Verfahren

codiert und verschickt. Der Vorteil der Nutzung von Tokens ist, dass bereits im Token Informationen über die Session und den User enthalten sein können, somit ist es nicht nötig, offene Sessions auf dem Server zu speichern. Das verwendete Prinzip unterstützt somit auch eine Auslagerung der Funktionalität zur Authentifizierung auf einen weiteren Server. Außerdem wird vereinfacht, dass ein Nutzer mehrfach mit unterschiedlichen Geräten verbunden sein kann.

4.4 E-Mail-Bestätigung

Ein Account hat entweder den Status „Active“ oder „Pending“. Nach jeder Registrierung befindet sich ein neu erstellter Account zunächst im Zustand „Pending“. Das bedeutet, dass der Account zwar schon existiert, aber noch nicht aktiv ist. Dazu muss der Benutzer seinen Account über einen per Mail zugesendeten Link freischalten. Bei der Registrierung, wird ein zufällig erstellter Bestätigungscode in der Form eines JWT Token erstellt.

```
const token = jwt.sign({ email: req.body.email }, config.mail_secret);

   var user = new UserDB.User({
      fullName: req.body.fullName,
      email: req.body.email,
      hashedPassword: UserDB.User.hashPassword(req.body.password),
      confirmationCode: token,
   });
```

Anschließend bekommt der Benutzer eine Mail mit einem Link, der als Query-Parameter diesen Code beinhaltet. Klickt der Benutzer auf den Link, wird er auf die entsprechende Seite im Frontend weitergeleitet, der dann einen Request an das Backend sendet.

Das Backend sucht dann in der Datenbank den Benutzer mit dem angeforderten Code. Stimmt dieser Code überein, wird dann der Status des Benutzers in der Datenbank auf „Active“ gesetzt und der Benutzer kann sich dann mit seinem Passwort einloggen. Für das Senden der E-Mails verwenden wir das npm package nodemailer, der mit einem Gmail Konto verknüpft ist (city-tours.hdm@gmail.com). Nodemailer verwendet SMTP für das Senden der E-Mails. [7]

[7] https://nodemailer.com/

5 Eingesetzte Technologien

Die Technologie, die wir im Projekt angewendet haben, ist der sogenannte MEAN-Stack (MongoDB, Express.js, Angular, Node.js).

5.1 MongoDBs

Wir verwenden MongoDB als Datenbank. MongoDB ist eine dokumentenbasierte Datenbank, die sehr effizient bei Abfragen ist. [8] Besonders im JavaScript Umfeld, ist sie sehr mächtig. MongoDB basiert auf NoSQL und ist im Vergleich zu klassischen SQL-Datenbanken weniger streng im Hinblick auf Normalisierungen und Beziehungen. Da unsere Technologien auf JavaScript bzw. TypeScript basiert, ist die Verwendung von MongoDB vorteilhaft, da MongoDB wie erwähnt dokumentenbasiert ist und mit Objekten arbeitet. MongoDB verwendet Schemen (ähnlich zu Entitäten aus SQL) und speichert die Einträge als Objekte in Dokumenten ab. Die Dokumente werden als BSON-Datei abgespeichert, welches vom Aufbau her ähnlich ist, wie eine JSON-Datei, d.h. sie verwendet Key-Value-Paare. Ein weiterer Vorteil von MongoDB ist, dass die Datenbank Open-Source ist. Ebenso bietet MongoDB eine Cloud an, sowie eine webbasierte Oberfläche, welche wir in der Entwicklungsphase verwendet haben.

Eine Abfrage vom Backend an die Datenbank lässt sich ebenfalls mit wenigen Code-Zeilen ausführen. Um beispielsweise einen User über die ID abzufragen ist folgende Abfrage nötig:

```
const getUserById = async (req, res) => {
   try {
      const user = await UserDB.User.findById({ _id: req.params.userid }).exec();
      res.status(200).send(user);
   } catch (err) {
      return res.status(500).send({ errorMessage: error });
   }
};
```

Die Abfrage sucht in der Datenbank mittels der von MongoDB zur Verfügung gestellte Methode `findById()`, ob es einen Eintrag in der User-Datenbank mit der angefragten ID existiert. Die ID wird hierbei über den URL-Parameter übergeben und an die Methode `findById()` übergeben.

[8] https://www.mongodb.com/

5.2 Express.js / Node.js

Das Backend läuft unter der JavaScript-Laufzeitumgebung Node.js. Mit Node.js haben wir Zugriff auf den npm-Packagemanager, welcher eine große Anzahl an verschiedenen Libraries anbietet. Wir verwenden Node.js im Zusammenhang mit Express.js, um einen Webserver zu betreiben. Durch Express.js lassen sich beispielsweise Middlewares einbauen, die wir unter anderem in unseren geschützten Routen eingebaut haben. Wir verwenden dazu eine eigens geschriebene Authentifizierungs-Middleware, um zu überprüfen, ob jemand berechtigt ist, den Endpoint aufzurufen.

Unser Backend haben wir aufgeteilt nach Funktionalitäten. Wir verwenden Controller, um HTTP-Requests zu behandeln. Sie sind zuständig für die Requests und Response. Die Anwendungslogik haben wir ausgelagert in Services. Für die Routen und Models wurde ebenfalls ein eigenes Verzeichnis angelegt. Die allgemeine Server-Logik liegt in der server.js-Datei. Umgebungsvariablen, wie der Port, die API-Schlüssel, sowie andere vertrauliche Daten sind in einer .env-Datei abgelegt, welche in der Regel in der .gitignore-Datei aufgeführt wird.

Um das Backend zu starten wird der Befehl `npm start` aufgerufen, welches den Server in unserem Projekt auf Port 3000 startet.

5.3 Angular

Angular ist ein von Google entwickeltes Frontend Framework, basierend auf TypeScript. [9] TypeScript ist eine Programmiersprache, die von Microsoft entwickelt wurde und ist eine Ergänzung zu JavaScript. [10] Zusätzlich zu den Funktionen in JavaScript, ist TypeScript spezifischer. TypeScript verwendet unter anderem Interfaces, Typisierung, Methodensignatur, generische Programmierung. Jeder Code der in JavaScript programmiert ist, ist mit TypeScript kompatibel. Letztendlich wird jeder TypeScript Code am Ende des Tages zu einem JavaScript Code kompiliert, da die Browser nur JavaScript verstehen. Durch TypeScript wird eine starke Typisierung ermöglicht, was zu weniger Laufzeitfehler führt.

Angular eignet sich hervorragend für Single-Page-Applikationen und verwendet das MVVM-Pattern. Die Hauptkonzepte in Angular sind Components, Directives, Templates, Module und Dependency Injection. Das Besondere dabei ist, dass der Code in kleinere Komponenten aufgeteilt wird, sodass es einfacher ist, den Code zu warten. Beispielsweise kann man jedes Feature bzw. jede Funktionalität auf der Website in Modulen bzw. Komponenten aufteilen. Ähnliche Technologien wären React.js oder Vue.js. Alle drei Frameworks sind sehr populär, haben ihre Vorteile und eignen sich zum Entwickeln komplexer Anwendungen. Angular gilt als Framework mit der höchsten Komplexität, eignet sich aber durch seine Struktur auch sehr zum Erlernen von Frontend-Architekturen und wird sehr gerne für Enterprise-Anwendungen verwendet.

Aufgrund des Single-Page-Konzepts wird ein ständiges Nachladen der Seite mittels HTTP-Request vermieden, was zu einer deutlichen Steigerung der Performance führt. Es wird lediglich nur eine HTML-Datei geladen, anstatt mehrere wie es bei einer klassischen Webanwendung der Fall ist.

Um das Frontend zu starten wird der Befehl `ng serve` verwendet. Das Frontend läuft dann auf dem Port 4200.

[9] https://angular.io/docs
[10] https://www.typescriptlang.org/

5.4 Cypress E2E-Tests

Cypress ist ein JavaScript End to End Testing Framework für automatisierte Webtests. [11] Ähnliche Frameworks sind Selenium oder Protractor. Es ist Open Source und kostenlos. Mit E2E Test werden Testszenarien durchgespielt, die von Anfang bis zum Ende durchlaufen, sie werden auch Full-Stack-Tests genannt. Dabei geht man von der Sicht des Endbenutzers aus. Mit Cypress möchten wir die Tests automatisieren. Die Tests laufen auf dem Cypress Test Runner im Electron Browser, der die Chromium Technologie verwendet. Die Tests sind ähnlich wie die Espresso Tests in Android, welche vor allem die Benutzeroberfläche testen, also aus Sicht des Benutzers. Dabei simuliert der Test Runner Klicks im Browser auf die zu testenden Elemente wie Buttons, Input-Felder, User-Feedback usw.

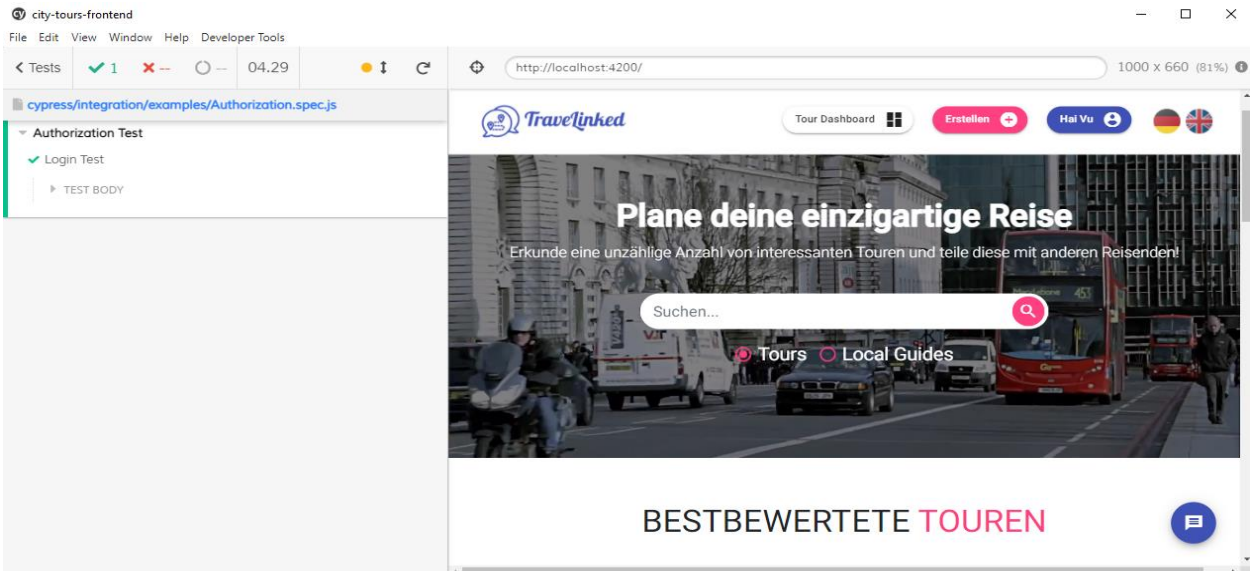

Abbildung 10: Cypress Test Runner

Im obigen Bild (Abbildung 10) sieht man den Cypress Test Runner. Cypress führt die Tests in einem einzigartigen interaktiven Runner aus, der es ermöglicht, die Befehle während der Ausführung zu sehen und gleichzeitig, die zu testende Anwendung zu betrachten. Auf der linken Seite sind die Testfälle aufgelistet. Cypress durchläuft alle Testfälle von oben nach unten. Auf der rechten Seite sieht man die zu testende Website. Während die Tests durchlaufen, kann man auf der Benutzeroberfläche nachvollziehen, welche Schritte durchgeführt werden. Wenn ein Test erfolgreich durchläuft, wird dieser grün markiert, bei einem Fehler dann dementsprechend rot. Sollte ein Test nicht erfolgreich durchgelaufen sein, wird Cypress nochmals versuchen den gleichen Test auszuführen. Man kann in den Einstellungen definieren, wie oft Cypress einen fehlgeschlagenen Test wiederholen soll.

[11] https://www.cypress.io/

6 Arbeitsmethodik

Die Arbeitsmethodik die wir angewandt haben, hatte Elemente der agilen Softwaremethodik. Das bedeutet, dass wir wöchentliche Meetings durchgeführt haben, um uns auszutauschen und den aktuellen Stand festzuhalten. Zur Dokumentation haben wir das Projektmanagement-Tool JIRA herangezogen. Zunächst haben wir uns zusammengesetzt und die Anforderungen gesammelt, was das Projekt beinhalten soll. Diese haben wir gegliedert in Must-have, Should-have und Nice-to-have. Diese Anforderungen haben wir dann letztendlich in unser Backlog auf JIRA hinzugefügt. Unseren Backlog haben wir dann nach und nach während des Projekts erweitert, was der agilen Softwaremethodik entspricht. So konnten wir die Aufgaben gut einteilen und jeder wusste, was seine Aufgabe war. Die ganzen Anforderungen haben wir wiederum in kleineren Tasks untergliedert. Das bedeutet, dass man in der Regel eine größere Anforderung bzw. ein größeres Feature zu implementieren hat, sogenannte User Storys, die dann aus mehreren kleineren Tasks bestehen. Eine User Story könnte beispielsweise die Login Authentifizierung sein, welche sich in verschiedenen Tasks aufteilen lässt, z.B. in „Implementierung von Passwort Hashing“, „Absenden einer Bestätigungsmail“ usw. Ein Task, der gerade im Bearbeitungsprozess ist, d.h. von einem Mitglied derzeit bearbeitet wird, befindet sich dann im Kanban Board in dem Zustand „in Progress“. Wenn dieser Task dann abgeschlossen ist, bekommt er den Zustand „Done“. Während des Projektverlaufes traten zudem immer wieder verschiedene Bugs auf, die sofort im Backlog hinzugefügt werden konnten, um diese Fehler auch allen anderen Teammitgliedern mitzuteilen.

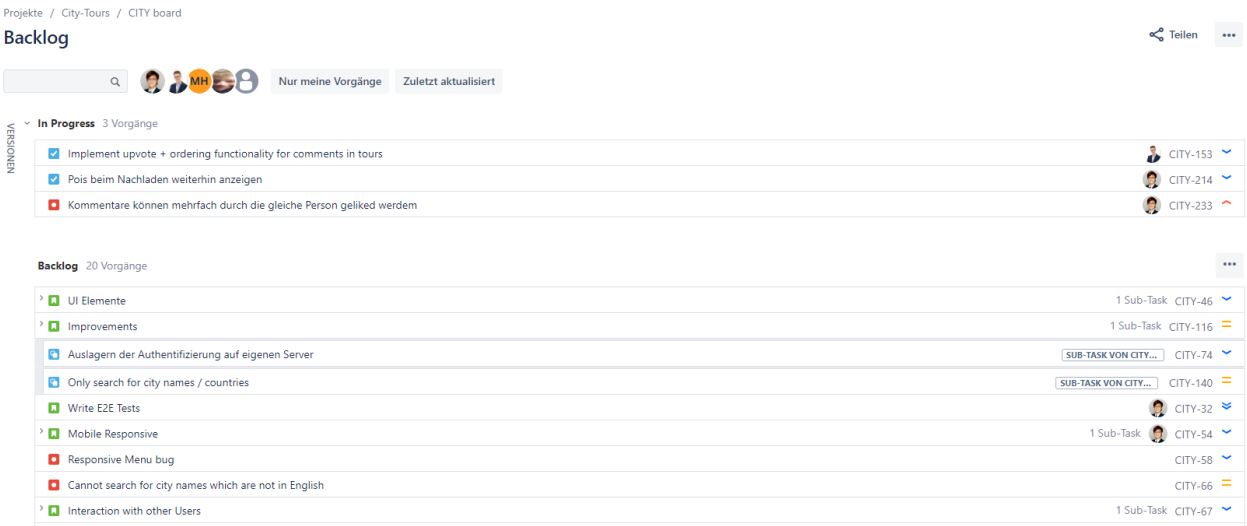

Abbildung 11: Backlog

7 Fazit

Durch das Projekt wurden Kenntnisse in der Softwareentwicklung weiter vertieft. Wir konnten unser erworbenes Wissen hier praktisch umsetzen. Dazu gehören Planung und Konzeption, sowie ein vernünftiges Zeit- und Ressourcenmanagement. Wir haben versucht, so gut es geht, das Projekt unter realen Bedingungen durchzuführen, wie es in der Praxis üblich ist. So haben wir viel Wert auf sauberen Code, Projektmanagement und Design Patterns gelegt. Auch haben wir Pair Programming betrieben, was sich ebenfalls als sehr effizient bewiesen hat.

Es klappte nicht alles sofort auf Anhieb und vieles musste selbständig recherchiert werden, dafür ist jedoch die Lernkurve sehr steil und man hat durch das Projekt viel Neues gelernt.
Zwischendurch gab es immer wieder Schwierigkeiten und Blocker, die den Workflow behinderten. Beispielsweise die Implementierung der Chat-Funktion war schwieriger als zuerst angenommen. Das Hosten der Anwendung auf dem Server hatte ebenfalls einige Tage in Anspruch genommen.
Das Arbeiten im Team erfordert eine gute Koordination. So hatten wir hin und wieder Merge-Konflikte gehabt und andere Meinungsverschiedenheiten, die aber zum Glück immer gelöst werden konnten. Zudem hängen einige Features von anderen Features ab, die wiederum von anderen Teammitgliedern bearbeitet werden. Aus diesem Grund musste man auf das Teammitglied warten bzw. man schaut sich gemeinsam den Code an und hilft sich gegenseitig.

Das Projekt hatte fast alle Bereiche der Webentwicklung abgedeckt (Fullstack). Dazu gehören die Implementierungen im Backend und im Frontend, sowie die dazugehörigen E2E-Tests und Unit-Tests und natürlich auch das Verfassen der Dokumentation.
Von der Entwicklung bis zum Deployment war alles dabei.
Das Projekt hat uns gezeigt, dass wir in der Lage waren, innerhalb einer gegebenen Zeit, eine fertige Anwendung zu entwickeln.

Zusammenfassend kann man sagen, dass so ein Software-Projekt eine sehr große und wertvolle Erfahrung war. Vor allem aber hat es auch sehr viel Spaß gemacht und man ist am Ende doch sehr stolz diese Anwendung auf der MediaNight präsentieren zu dürfen.
Vielleicht bietet sich noch die Gelegenheit an, unsere Anwendung in Zukunft zu vermarkten. Wir sehen da auf jeden Fall Potential.