Dokumentation
Webprojekt

1 Projektmitglieder

- 1904749
- 6378617

2 Framework-Auswahl und Hilfstools

2.1 Frontend-Framework

Am Anfang sind wir in das Projekt gestartet und haben beide Frameworks verglichen. Da dies sich aber als nicht wirklich einfach herausstellte, haben wir einfach mal mit Next.JS gestartet, da dies im Internet als das „einfachere“ Framework beschrieben wurde. Nach kurzer Zeit hat sich für uns dann aber herausgestellt, dass dies – zumindest für uns – nicht so ist, weshalb wir zu Angular gewechselt haben. Dies stellte sich als eine gute Entscheidung heraus, da Next.JS uns zu unübersichtlich wurde, Angular dies aber durch die Aufteilung des Codes in HTML-, TS- und SCSS-Dateien, vereinfacht.

2.2 Hilfstools

GitHub Copilot:

Wir haben zur Codevervollständigung GitHub Copilot verwendet.

DB-Browser for SQLite

Wir haben zur Einsicht in die DB dieses Tool verwendet.

JetBrains WebStorm

Wir haben als unsere IDE WebStorm verwendet.

Bruno API Client

Als unseren API-Client haben wir Bruno verwendet.

3 Open-Source Pakete

3.1 Backend

Fastify

Mit Fastify lässt sich schnell und einfach eine API auf Node.js Basis aufbauen. Hierbei vereinfacht Fastify die Erstellung von Routen und kümmert sich automatisch um z.B. die HTTP-Codes der Antwort.

Fastify-Cors

CORS ist eine Sicherheitseinschränkung eines Servers, um zu verhindern, dass ungewollte Clients auf zum Beispiel die API zugreifen, um Daten abzurufen. Mit Fastify-Cors lassen sich diese Richtlinien in einem Fastify-Server konfigurieren.

Fastify-Multipart

Diese Erweiterung ermöglicht das Verarbeiten einer Multipart-Form-Request in einem Fastify-Server. Damit lassen sich Dateien und andere Properties auf dem Server empfangen und verarbeiten.

Wir haben diese Erweiterung verwendet, um die Anforderung zu erfüllen, Dateien empfangen zu können.

Better-Sqlite3

Dies ermöglicht die Einfache Einrichtung und Implementierung einer SQLite Datenbank in ein Node.js Projekt.

Wir verwenden eine SQLite Datenbank in unserem Projekt, weshalb dies essentiell für uns war.

FS

Diese Erweiterung ermöglicht die Arbeit mit dem lokalen Dateisystem.

Dies wird benötigt, da wir die empfangenen Dateien abspeichern müssen, und diese lokal gespeichert werden.

JsonWebToken

Ist eine beliebte Bibliothek für die Verwaltung und Verarbeitung von JSON-Web-Tokens (JWTs). Diese werden zur sicheren Übertragung von Informationen zwischen zwei Systemen verwendet, oft zur Authentifizierung.

Wir verwenden diese JWTs, um unsere Authentifizierung sicher zu ermöglichen. Nach dem Login bekommt der User das Token ausgestellt, was die Rolle enthält und somit kann bei jeder Abfrage geprüft werden, ob der Nutzer berechtigt ist, die Abfrage zu tätigen.

3.2 Frontend

Fontawesome

Fontawesome bietet eine große Bibliothek an Icons und vielem weiterem, das sich einfach verwenden lässt, um eine Website zu gestalten.

Wir verwenden diese Erweiterung, um die Icons z.B. beim Login darzustellen.

Bulma

Dieses CSS-Framework bietet moderne, vorgefertigte Komponenten, um eine Webseite zu gestalten. Es basiert auf Flexbox, einem Layoutmodell, das eine flexible und responsive Anordnung von Elementen ermöglicht.

Wir verwenden dieses CSS-Framework, da es schnell und einfach zu implementieren ist und sehr modern. Es ist allgemein auch sehr einfach und intuitiv bei der Programmierung zu verwenden.

NGX-Toastr

Diese Bibliothek ermöglicht sehr schnell und einfach die Implementation von Toast-Nachrichten. Diese werden nur temporär dargestellt und über allem anderen eingeblendet.

Wir verwenden diese Bibliothek, um die Funktion von Toast Nachrichten einfach zu implementieren.

RxJS

Ermöglicht eine einfache Verarbeitung von asynchronen Ereignissen und Datenströme, wie zum Beispiel bei den Requests verwendet wird.

Wir verwenden diese Bibliothek, um unsere Server-Requests zu verarbeiten.

4 Einrichtung und Besonderheiten

4.1 Einrichtung der Software

1. ZIP-Ordner entpacken – enthält folgende Ordner
    a. Bruno (Bruno Collection)
    b. Client (Angular Frontend)
    c. Server (Fastify backend)
2. Ordner als Projekt öffnen
3. In den Ordnern „Client“ und „Server“ den Command ‚npm install‘ ausführen
4. Danach den Server starten mit ‚npm run dev‘
5. Die Client Anwendung starten mit ‚npm start‘
6. Bruno Collection kann in Bruno geöffnet werden und ist damit fertig eingerichtet
7. Die Anwendung ist nun bereit

4.2 Anmeldung

Accountmanager:

Username: Accountmanager

Password: test

Developer

Username: Developer

Password: test

User

Username: User

Password: test

Die Datenbank sollte die notwendigen Daten zur Anmeldung enthalten.

Ansonsten:

- Bruno öffnen und die 2 folgenden Endpunkte ausführen
    - Settings / POST RecreateTables
    - Test / POST LoadTestData
- Datenbank sollte im Backend im Ordner Datenbank sein
    - database / customerpanel.db
    - wird ansonsten auch automatisch erstellt und befüllt

4.3 Nutzung von Bruno:

- Ersteinrichtung (Falls Anmeldung (Nächster Schritt) nicht direkt möglich)
    - Settings / POST RecreateTables
    - Test / POST LoadTestData
- Anmeldung
    - Auth / POST User Authentication
        - meldet Stanardmäßig Accountmanager an
        - Wechsel im Body möglich
- Danach sollten alle Requests ausführbar sein

4.4 Besonderheiten

Recreate Tables

Wir haben eine Funktion eingebaut, um den Entwicklungs- und Einrichtungsprozess zu vereinfachen. Diese Funktion ist über den Endpunkt:

Settings / POST RecreateTables

abrufbar, womit man in der Datenbank alle Tabellen löschen kann und diese werden dann automatisch neu erstellt, sodass man einfacher in der Entwicklung neue Entities hinzufügen kann. Außerdem ist diese Funktion auch im Frontend auffindbar unter den Settings in der Sidebar, damit man beim Testen schneller die Testdaten wieder löschen kann und andere Dinge testen kann. Diese Funktion im Frontend lädt auch automatisch wieder die Testdaten, damit man sich wieder anmelden kann, ohne einen API-Client zu verwenden.

4.2 Dokumente erstellen

Beim Erstellen / Hochladen eines neuen Dokuments erstellt es dieses teilweise doppelt. Dieses Problem konnte nach aktuellem Stand von uns nicht behoben werden.

5 Technische Entscheidungen und Hintergründe

5.1 Login

Wir haben ein Login-System eingebaut, das Automatisch dem Benutzer eine Rolle zuweist anhand des in der Datenbank hinterlegten Benutzereintrags.

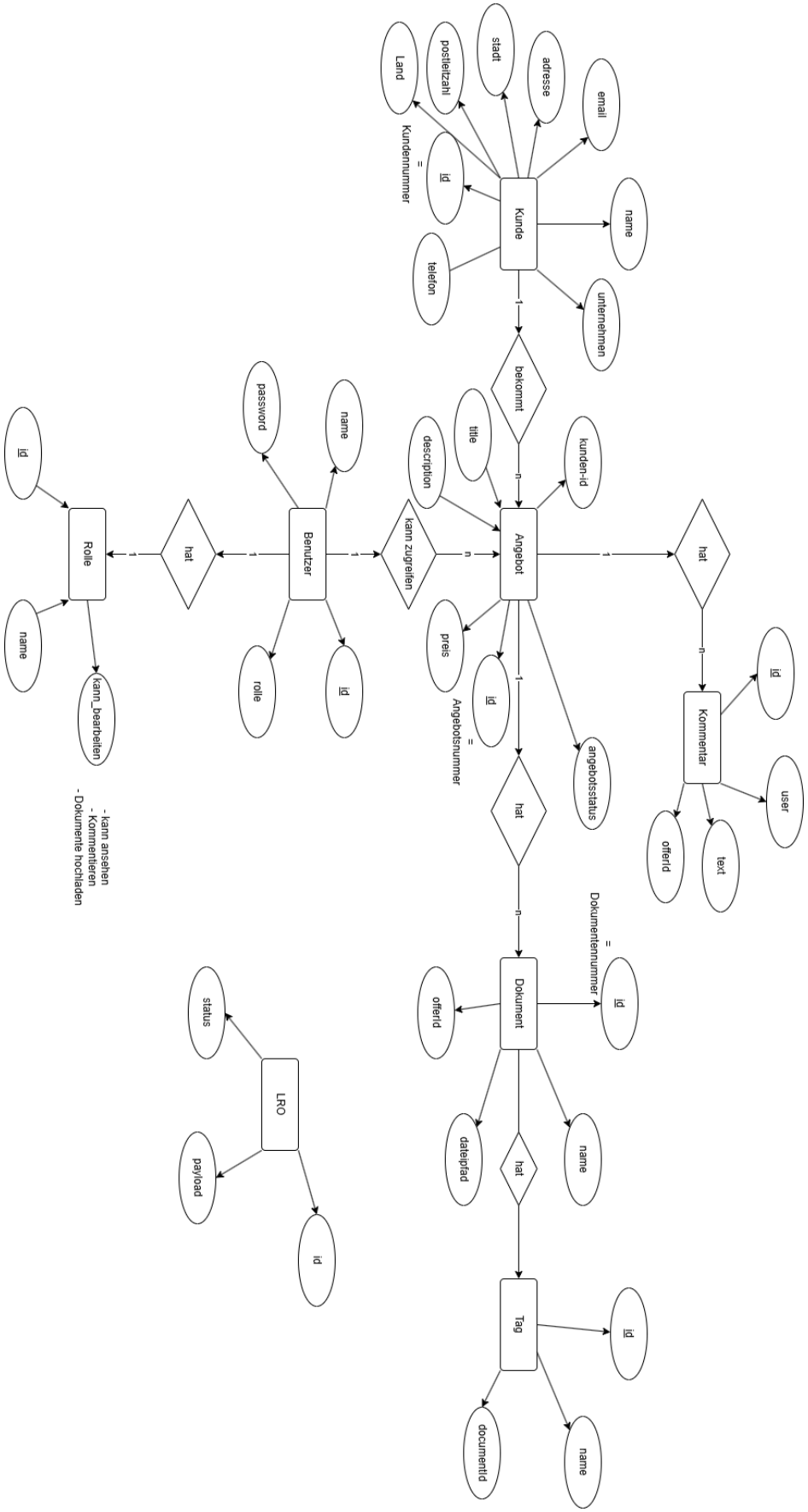
Der Login läuft folgendermaßen ab:

- Der Benutzer meldet sich bei der Plattform mit Benutzername und Passwort an
- Anmeldedaten werden mit der DB abgeglichen und wenn der Nutzer vorhanden ist, wird ein JW-Token generiert.
- Der Token enthält die Rolle des Benutzers, womit später geprüft werden kann, ob dieser Zugriff auf jeweilige Funktionen hat
- Der Token wird als Response zurückgeschickt
- In allen folgenden API Requests wird der Token als Authorization-Header mit angegeben
- Bei den Requests wird im Backend mittels einer Middleware geprüft, ob der Token die entsprechende Rolle enthält und dann die entsprechende Route freigegeben

5.2 Auto-Logout

Der Benutzer wird Automatisch nach 30min abgemeldet, sodass er sich neu anmelden muss.

6 Ressourcen Repräsentation



7 Absicherung der Funktionalitäten

Hinweis: Rollen-, User-, Settings- und Test- Endpunkte werden aktuell nicht überprüft, da dies ein Systemadministrator in einem „Real-World-Beispiel“ machen würde.

Accountmanager:

Kann alles, was Entwickler kann und:

- Kann Kunden verwalten
- Kann Angebote erstellen und löschen
- Kann Angebotsstatus wechseln
- Kann Dokumente ändern

Entwickler:

Kann alles, was Benutzer kann und:

- Kann Angebote ändern
- Kann auf Kommentare erstellen, diese ändern und löschen
- Kann Legacy-Migrationen durchführen
- Kann alle Kunden sehen
- Kann Dokumente erstellen und löschen
- Kann Tags verwalten
- Kann Dokumente nach Tags filtern

Benutzer:

- Kann Dokumente einsehen
- Kann Angebotsdaten einsehen
- Kann spezifischen Kunde einsehen
- Kann Kommentare einsehen

8 Filterung

8.1 Dokumente

- name
- offerId

8.2 Kommentare

- user
- text
- offerId

8.3 Kunden

- name
- email
- phone
- address
- city
- country
- zip
- company

8.4 Angebote

- title
- description
- price
- status
- customerId

8.5 Tags

- name
- documentId

9 Dateien Tags Erweiterung

Erweiterung des Datenmodells / des Backends mit Tags:

- Tabelle zu tables.js hinzugefügt
- Schema erstellt
- Route erstellt
- Logik erstellt
- RecreateTables ausgeführt

10 Ausblick

Funktionen, die man noch hinzufügen / bearbeiten könnte:

- Bug beheben, dass Dokumente doppelt erstellt werden
- Entfernen des RecreateTables aus dem Frontend → Nutzen hauptsächlich in der Entwicklung
- Auto-Logout nur bei Inaktivität, aktuell immer
- Internes Kundenbewertungssystem
    - Zuverlässigkeit und Co. können für die Zukunft festgehalten werden
- Faktor Zeit zu den Angeboten hinzufügen
    - Kalenderübersicht, wann Angebot planmäßig abschließt
- Rechnungssystem implementieren
    - Automatische Erstellung von Rechnungen aus den Angeboten