



Dokumentation Tannder

Konzept

UI/UX Konzept

Arbeitspakete:

Client-Arbeitspakete

Server-Arbeitspakete

Dokumentation der Beiträge

Umsetzung

Beschreibung der Environments

Beschreibung des Clients

Login

Navigation

Upload von Bäumen

Voting

Rangliste (Treederboard)

Karte

Beschreibung des Servers

Main

Interfaces

TreeRod

Forest

EloLogic

TreeCognition

Anleitung zum Starten der Anwendung

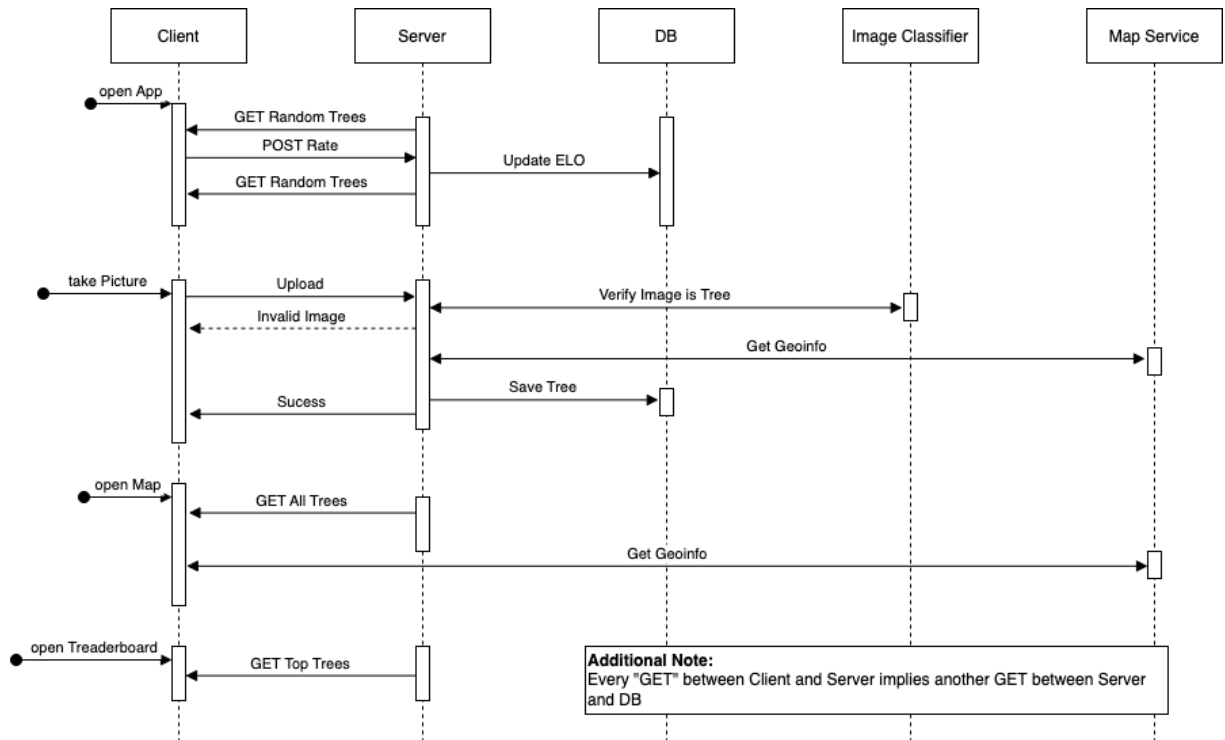
Vorraussetzung: Google Cloud Account

Run (Production)

Run (Development)

Konzept

Ziel des Projektes war es eine App zu entwickeln, mithilfe derer die Benutzer/-innen Bäume miteinander vergleichen können. Dafür sollen beim Öffnen der App jeweils zwei Bäume mit ihren Bildern angezeigt werden. Die Nutzenden können danach entscheiden welcher der "bessere" Baum ist und diesen auswählen. Das Hauptkriterium für die Bewertung soll hier ein Foto jedes Baumes sein. Jedem der Bäume soll ein Elo-Rating zugeordnet werden. Dieses soll dann bei jedem Baum-Vergleich angepasst werden. Alle Benutzenden können selber neue Bäume hochladen indem sie die, auch im Web ausführbare, Kamera-Funktion der App nutzen. In der App sollen außerdem die zehn Bäume mit dem höchsten Elo-Wert angezeigt werden. Noch dazu soll eine Karte implementiert werden, in der der Standort der Bäume angezeigt wird.

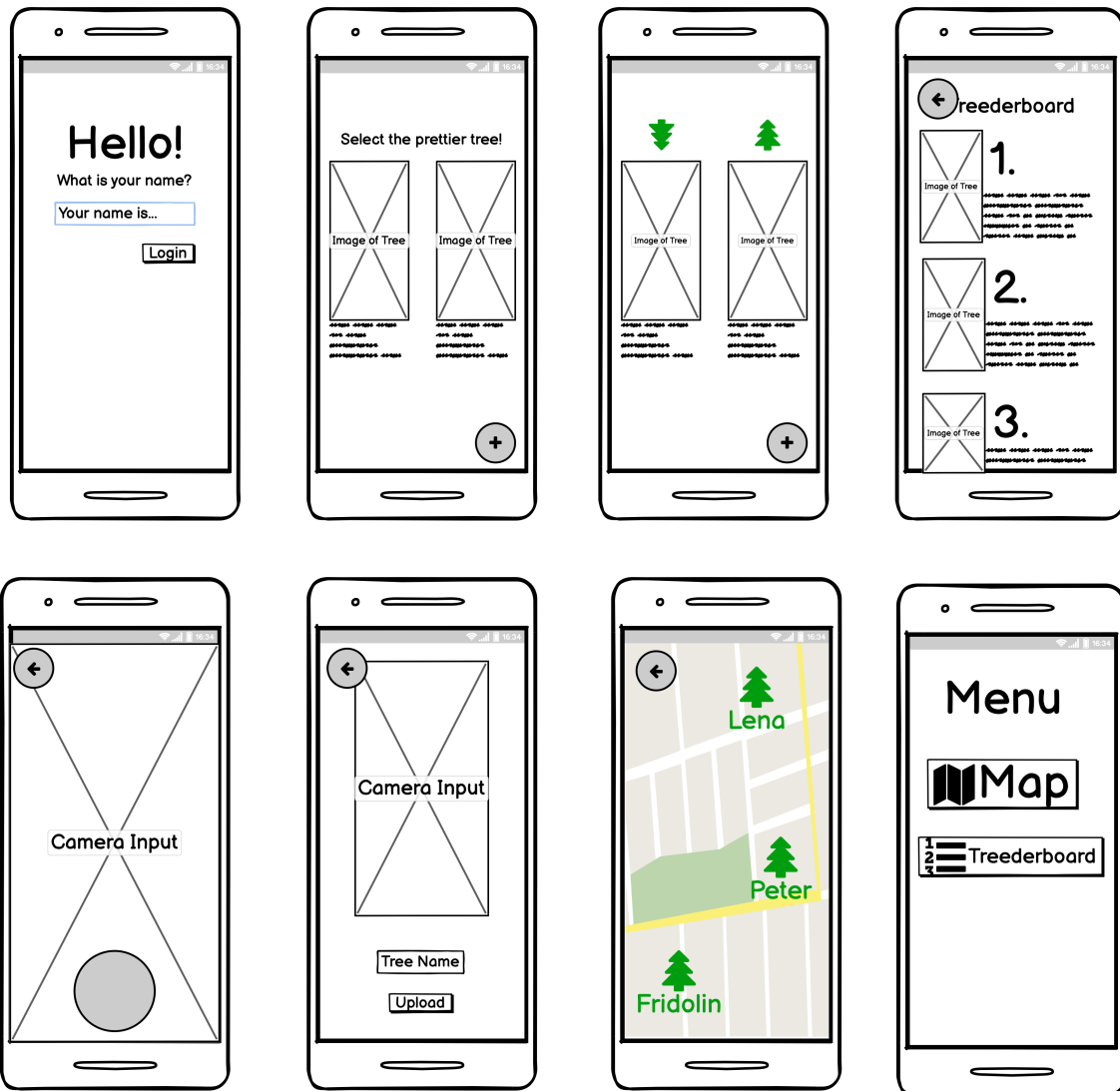


Flowchart der Application: 5 Verschiedene Services interagieren miteinander um die verschiedenen Funktionalitäten abzudecken.

Um diese Funktionalitäten umzusetzen sollen einige externe Services eingebunden werden. Der erste Service ist eine MongoDB als Datenbank um die Bäume speichern zu können. Die Karte funktioniert über eine LeafLet/OpenStreetMap-Schnittstelle. Bei dem Upload der Bäume wird überprüft ob es sich tatsächlich um einen Baum handelt, was durch den Image Classification Service von Google überprüft wird. Außerdem wird das Kamera Plugin von Cordova genutzt, welches durch PWA-Kamera-Fähigkeiten ergänzt wird.

UI/UX Konzept

Up tree: animation move in from bottom
Down tree: falling tree animation



Mockup des User-Interfaces

Zuerst soll der/die Nutzende seinen Namen über eine Login-Page eingeben können. Daraufhin wird er auf die "Voting-Page" weitergeleitet wo ihm zwei Bäume gezeigt werden. Daraufhin kann er den schöneren Baum auswählen woraufhin eine Animation erscheint. Daraufhin werden zwei neue Bäume präsentiert. Er hat jederzeit die Möglichkeit zur "Karten-Ansicht" über ein Menü zu gelangen. Dort wird ihm eine Karte angezeigt, auf welcher alle hochgeladenen Bäume gekennzeichnet sind. Weiterhin kann er sich die besten 10 Bäume auf dem "Treederboard" anschauen, ebenfalls über das Menü erreichbar. Auch über das Menü erreichbar ist eine Seite zum Hochladen eines Baumes.

Arbeitspakete:

Um die Arbeitspakete sinnvoll einzuteilen wurde das Entwicklungs-Team in zwei Teams aufgeteilt. Eines dieser Teams arbeitet exklusiv am `client` und das andere exklusiv am `server`. Das Server-Team besteht aus Ole Offerdinger und Philipp Wussow und das Client-Team besteht aus Tobias Hölzer und Amon Schneider. Innerhalb dieser Teams gibt es noch eine detailliertere Aufteilung der Arbeit. Wenn die Arbeitspakete einen Verantwortlichen haben, so wurde dieser Teil von dieser Person alleine geschrieben. Wenn zwei Verantwortliche eingetragen sind wurde dieser Teil von beidem, meistens im Rahmen von Pair-Programming geschrieben.

Client-Arbeitspakete

- Login-Page - Tobias Hölzer
- Voting-Page - Tobias Hölzer und Amon Schneider
- Map-Page - Amon Schneider
- Menu - Amon Schneider
- Leaderboard-Modal - Tobias Hölzer
- Upload - Amon Schneider und Tobias Hölzer
- UX - Tobias Hölzer und Amon Schneider
- Api Service - Tobias Hölzer

Server-Arbeitspakete

- Express-Server (TreeRod) - Ole Ofterdinger und Philipp Wussow
- Datenbank Anbindung (Forest) - Philipp Wussow und Ole Ofterdinger
- ELO-Service (EloLogic) - Philipp Wussow
- Image Recognition Service (TreeCognition) - Ole Ofterdinger

Dokumentation der Beiträge

GitHub Commits sind in Git Dokumentiert. Da einige von uns mit verschiedenen Computern gearbeitet haben und das Projekt auf GitHub erstellt wurde und nur auf GitLab gemirrored sind die Namen und Mail-Adressen nicht die gleichen.

clacc4 ist Amon Schneider

ShirkahnW ist Philipp Wussow

Der Graph mit allen Beiträgen ist zu finden unter:

<https://git.dhbw-stuttgart.de/software-engineering-wwi2019f/assignment-gruppe-4/-/network/main>

Umsetzung

Im Folgenden wird erklärt, wie die App zu benutzen und implementiert ist. Weiterhin wird erklärt wie eine lokale Version der Application gestartet werden kann.

Die Applikation ist geteilt in einen Ionic-Client, einen Express-Server und eine Mongo-Datenbank. Der Produkt-Lifecycle der Application besteht aus zwei Stages: Production und Development. Um die App in Production auszuführen wird Docker Compose genutzt.

Beschreibung der Environments

Entwickelt wurde die App teamübergreifend in einem Visual Studio Code Workspace, wodurch durch Extensions für Linting und Formatting Code-Qualität sichergestellt werden konnte. Über VSCode Remote Workspaces können auch Technologien wie WSL2 auf Windows zum Einsatz kommen.

Bei der Entwicklung werden die verschiedenen Services "nativ" gestartet - also ohne explizite Vernetzung über eigene Docker Networks, Volumes oder ähnliches. So wird der Ionic-Client auf Port 8100 ausgeführt und der Express-Server auf 3000. Der Express-Server nutzt in der Development Stage Cors.

In Production wird hingegen Docker Compose genutzt. Hierbei wird der Ionic-Client und der Express-Server kompiliert und optimiert. Der Ionic-Client wird daraufhin über den Express-Server "geserved", da es sich hierbei um Static-Files handelt. Daher ist auch die Nutzung von Cors nicht nötig.

Beschreibung des Clients

Der Client der Tannder-Applikation wurde auf der Basis von Ionic und Angular Frameworks entwickelt. Der Fokus bei der Entwicklung und Prioritäten hinsichtlich der Nutzererfahrung lagen auf der Einbindung von Wiederverwendbaren Komponenten und Modals. Dies und angestrebte geringe Ladezeiten sorgen für sehr flexible Nutzungserfahrung der App mit einer klar ersichtlichen Navigation.

Über die klaren Services hinaus werden interne Angular-Elemente wie ein Storage verwendet.

Auch Ionic-Komponenten und Symbole werden für die Ausgestaltung der Nutzeroberfläche verwendet.

Login



Wenn der Nutzer die App öffnet, wird geprüft, ob der Nutzer bereits einen Namen festgelegt hat. Falls dies der Fall ist, wird der Benutzer auf die Bewertungs- bzw. Abstimmungsseite weitergeleitet. Falls der Nutzer jedoch noch keinen Namen festgelegt, wird der Benutzer hingegen auf die Anmeldeseite weitergeleitet.

Die Anmeldeseite besteht aus einer Texteingabe und einem Anmeldeknopf. Der Benutzer kann dort einen Namen oder Alias eingeben und den Login-Button drücken. Diese Aktion speichert den Benutzernamen im lokalen Speichersystem und ermöglicht eine Nutzung der Applikation über die Anmeldeseite hinaus.

Wenn der Benutzer somit nun die App erneut aufruft, wird der Benutzername im lokalen Speicher gefunden und der Benutzer wird zur App weitergeleitet.

Navigation


Grundsätzlich ist die Navigation der Applikation bewusst sehr intuitiv gestaltet. Das Menü besteht in einem sogenannten "Floating Action Button", ein auf jeder Seite präsentierter Knopf. Beim Drücken dieses Knopfes wird das Menü durch die jeweiligen NavigationOptionen ergänzt. So kann über diese Menüpunkte stets jede Seite und Funktionalität von jeder Seite aus erreicht werden. Optionen sind hier der Menüpunkt zum Hinzufügen eines neuen Baumes, der Menüpunkt zum Anzeigen der Rangliste und der Bewertungsseite, sowie ein Menüknopf zum Anzeigen der Baumkarte.

Dieses gesamte Navigations- bzw. Menüelement wird als ein globaler Komponent auf allen Seiten wiederverwendet und bietet somit eine sehr leichtgewichtige und minimalistische Option zur Navigation.

Upload von Bäumen

Upload

CLOSE



Preview

Name your Tree!

Günter

SUBMIT

Der Benutzer kann die Kamera über das im Menü enthaltene Plus-Symbol öffnen. Diese Aktion wiederum aktiviert die Kamera und versucht auf den Standort des Nutzers zuzugreifen. Dafür wird der Nutzer nach der Erlaubnis für die Verwendung dieser

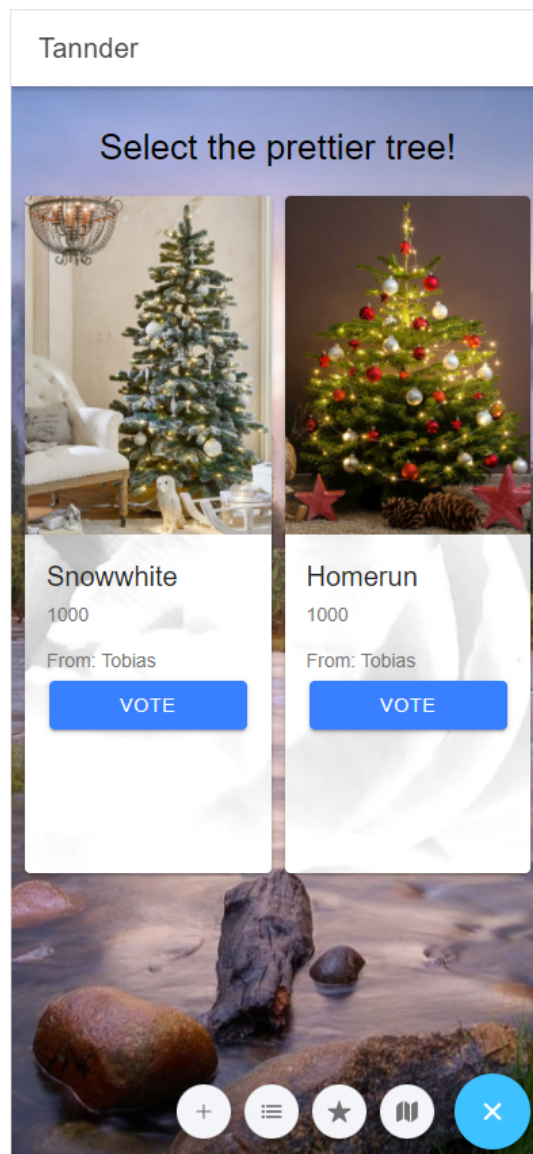
Funktionalitäten gefragt. Erfolgt keine Zustimmung, kann trotzdem ein lokal gespeichertes Bild hochgeladen werden. Nachdem der Benutzer jedoch im Normalfall ein Foto von seinem Baum gemacht hat, muss er einen Spitznamen für den Baum eingeben, woraufhin er das Foto abschicken kann. Bei einem erfolgreichen Upload, schließt sich das Modal, welches das Formular und die Vorschau des hochzuladenen Bildes enthält, automatisch.

Ein hinzugefügter Baum wird durch eine POST-Request an das Backend übermittelt. Speziell der Pfad `/trees/upload` wird hier mit einem Promise verknüpft und stellt entweder eine, Erfolg bestätigende, Antwort oder Fehlercodes bereit.

Kann ein Baum nicht hinzugefügt werden, wird auf Nutzerebene ein Fehlerdialog angezeigt.

Generell sind jedoch eine Standortfreigabe und Kamerafreigabe Voraussetzung für das Hinzufügen eines Baumes.

Voting



Auf der Seite der Applikation, welche für die Bewertungs-Funktionalität vorgesehen ist, kann der Benutzer einen von zwei Bäumen auswählen, die per `/trees/random` aus dem `server` abgerufen wurden. Die Bilder der Bäume lassen sich per Klick vergrößern, um sie besser bewerten zu können. Nach seiner Auswahl durch das Tippen auf eine der zwei gegenübergestellten Vorschaukarten von Bäumen, startet eine Animation. Zugleich wird im Hintergrund die getroffene Auswahl, bzw. die Metadaten

des bevorzugten Baums an den Pfad `trees/vote` der Backend-API gesendet und zwei neue Bäume werden bei `/trees/random` abgefragt, sowie erneut präsentiert. Auf diese Weise können kontinuierlich Bäume bewertet werden, wobei der Großteil der der hiermit verbundenen Logik ins Backend verlagert wurde.

Rangliste (Treederboard)

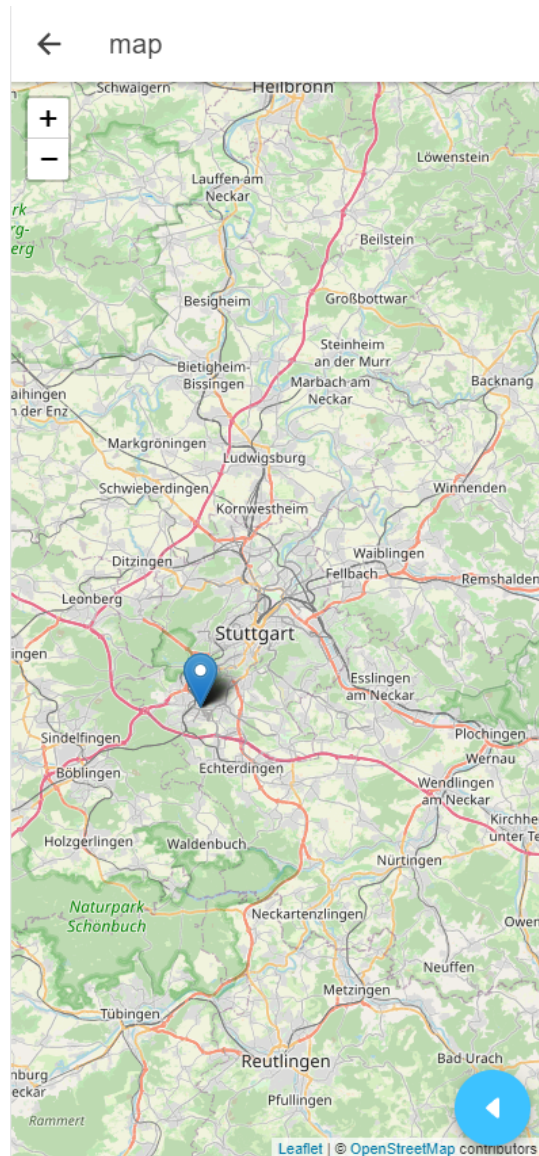


Der Benutzer hat auch die Möglichkeit, die Top 10 der global beliebtesten Bäume zu sehen. Präsentiert werden diese 10 Bäume über eine Rangliste, die über ein zugehörigen Button im Menü erreichbar ist. Dabei beginnt unsere Rangliste bei 0, da dies die einzig richtige Art und Weise ist eine Liste zu nummerieren.

Präsentiert wird hier ein Modal, welches Ionic-Cards für die 10, laut Backend-API, beliebtesten Bäume, anzeigt. In jeder dieser Karten ist jeweils eine Platzierungsnummer, ein Bild des Baumes, ein Punktestand und einen Baum-Namen enthalten.

Um über die Rangliste, trotz der Anforderung an detaillierte Informationen, eine positive Nutzungserfahrung mit minimalen Ladezeiten zu gewährleisten, wird die Backend-API über den Pfad `/trees/many?max=10` aufgerufen und somit 10 Detailinformationen zu Bäumen anstatt vieler genereller Informationen erhalten.

Karte



Wenn der Benutzer auf das Kartensymbol des Menüs klickt, wird ihm eine OpenStreetMap angezeigt, die mit Hilfe der JavaScript-Bibliothek Leaflet und ihren Kartenkacheln erstellt wurde. Die Karte verfügt über verschiedene Bildlafebene und ermöglicht es dem Benutzer, die Karte zu vergrößern oder zu verkleinern. Auf diese Weise kann der Benutzer die Standorte der in der Datenbank gespeicherten Bäume weltweit erkunden, die über `/trees/many` abgerufen werden. Die Bäume selbst werden durch Markierungen dargestellt, die an verschiedenen Koordinaten und somit in verschiedenen Regionen der Karte platziert sind.

Die OpenStreetMap sowie das Framework Leaflet erfordern zwar eine aufwändigere Einbindung, bringen jedoch hingegen Mapbox oder ähnlichen Services mehr Möglichkeiten zur offenen Nutzung und Flexibilität mit sich.

So können beispielsweise die Markierungen, welche Bäume repräsentieren, individuell durch Symbole angepasst werden und auf Klicks mit erneuten API-Aufrufen reagieren.

Die API-Abfrage erfolgt hier aufgrund des Skalierbarkeitsaspektes über den Pfad `trees/single/<ID>`, über welchen minimale Informationen und keine Base64-Strings zurückgegeben werden. Beim Klick auf einzelne Markierungen könnten nämlich immer noch zusätzliche Details abgerufen werden.

Beschreibung des Servers

"Das Backend ist eine REST-Schnittstelle zur Datenbank. Hier werden alle Daten gespeichert, die vom Client an den Server gesendet werden. Die Daten werden in einer MongoDB-Datenbank gespeichert. Die Datenbank wird mit dem Docker-Container gestartet. Die Datenbank wird mit dem Docker-Container gestartet, der auf dem Server läuft."

Der obere Text wurde von GitHub Copilot geschrieben. Er klingt zwar komisch aber daran ist alles richtig. Leider passiert danach nichts weiter als dass immer wieder erwähnt wird, dass der Docker Container auf dem Server läuft. Der restliche Text ist also eigene Arbeit.

Für das Backend wird das Express-Framework verwendet. In Anlehnung an das MVC Pattern dient der Router als View lediglich zur Definition von Schnittstellen und Überprüfung von Eingaben. Die gesamte Geschäftslogik ist in den Services definiert und die Schemas sind das Model.

Elementar sind auch die Wortspiele bei der Benennung, weil Software, die nicht ihren Metaphern treu bleibt nicht ernstgenommen werden würde (Siehe Flask, SQLAlchemy, BeautifulSoup etc.).

TreeRod → Express Router

Forest → TreeStorageService (DB Anbindung)

TreeCognition → Tree Recognition

Treexamples → Example Images of Trees

Main

Main dient lediglich zum Start des Servers und zur Definition einiger Einstellungen. Unter Anderem wird dort die Body-Größe angepasst um Bilder über Base64 zu übertragen.

Die Entscheidung dafür Base64 zu nutzen, statt einen Buffer zu übertragen, ist darauf begründet, dass dies in der Umsetzung einfacher ist und dies auch für den TreecognitionService und in der Datenbank verwendet wird. Die Alternativen wie z.B. Mulder sind zu umfangreich für die simple Anwendung. Zudem müssen die Bilder in allen Software Teil nie in ein anderes Format konvertiert werden.

Interfaces

Zur Kommunikation werden drei verschiedene Interfaces verwendet.

GeoInfo dient lediglich als Interface um zwei Geokoordinaten zu übertragen.

Die Bäume unterscheiden sich in zwei Typen. Bäume die vom Nutzer an den Server gesendet werden sind ein `NewTree`. Bäume die vom Server an den Client gesendet werden sind ein `Tree`. Sie unterscheiden sich dadurch dass ein `NewTree` keine ID und kein Elo-Rating enthält und ein `Tree` kein Bild enthält, da dies separat übertragen wird.

TreeRod

TreeRod definiert die Routen und ruft die entsprechenden Services auf. Wichtiger Teil von TreeRod ist die Validierung der Eingaben. Da die API offen ist sollen so fehlerhafte oder schädliche Anfragen verhindert werden. Im Falle einer solchen Fehlerhaften werden die entsprechenden HTTP Error Codes zurückgesendet. Da einige Fehler nicht in die Spezifikationen von HTTP passen wurden für einige Fehler eigene Fehlercodes zur besseren Differenzierung genutzt:

- 579 Beim Laden von Bäumen: Es sind weniger als 2 Bäume in der Datenbank
- 469 Beim Hochladen eines Baumes: Der Baum ist kein echter Baum und wurde vom TreeCognitionService abgelehnt

Für die Kommunikation werden 6 Routen definiert.

Um einen Baum hinzuzufügen werden wird `POST trees/upload` benutzt.

Um 2 zufällige Bäume vom Server für einen Vote zu laden wird `GET trees/random` genutzt.

Um für einen der beiden Bäume abzustimmen und das Elo-Rating auszulösen wird `POST trees/vote` benutzt. Im Body wird die ID des Gewinner- und des Verlierer-Baumes übergeben. Damit werden in der EloLogic die neuen Elo Werte berechnet und in die Datenbank gespeichert.

Um einen spezifischen Baum mit einer bereits bekannten ID zu erhalten wird `GET trees/single/<ID>` genutzt.

Um alle Bäume für das Leaderboard zu bekommen wird `GET trees/many?max=<amount>` verwendet. Der Parameter `max` ist optional und wenn er nicht mitgegeben wird, werden alle Bäume ausgegeben.

Die Anfrage `GET trees/image/<id>` dient als Hilfsanfrage für die anderen Anfragen um die Bilder der Bäume zu laden, weil besonders in `GET /trees/many/` zu viele Daten übertragen werden würden. Außerdem überträgt sie die Bäume direkt als Buffer damit sie einfacher in Web Apps einzubinden sind.

Forest

Forest bindet die Datenbank an den Server an. Diese App nutzt eine MongoDB die mithilfe von Mongoose angesteuert wird. Um die Bäume abspeichern zu können, werden alle Informationen aus den Tree-Interfaces in ein gemeinsames Tree-Schema zusammengefasst. Dies sieht folgendermaßen aus:

```
userName: String,
treeName: String,
eloRating: Number,
geo: { lat: Number, lon: Number },
image: String,
```

Um alle von TreeRod spezifizierten Funktionen ausführen zu können, werden verschiedene Funktionen implementiert:

Eine Funktion sorgt dafür, dass Bäume in die Datenbank gespeichert werden können. Dabei wird jedem neuen Baum ein Standart Elo-Rating von 1000 mitgegeben.

Eine weitere Funktion sorgt dafür, dass zwei zufällige Bäume aus der Datenbank zurückgegeben werden. Dafür wird `$sample` genutzt. Es wird ein Error geworfen, wenn es nicht genug Bäume in der Datenbank gibt.

Um die Elo-Ratings der Bäume zu aktualisieren, sucht eine Funktion einen Baum mit einer gegebenen ID heraus und ersetzt das alte Rating mit einem neuen mitgegebenen Rating.

Um einen spezifischen Baum aus der Datenbank zu bekommen, kann ein Baum mit seiner ID gesucht und aus der Datenbank geladen werden. Um diese Anfrage schneller zu machen, werden dabei die Bild Daten nicht mitgegeben.

Um trotzdem auf die Bilddaten zugreifen zu können gibt es eine separate Funktion, mit der ein Baum mit seiner ID gesucht und nur seine Bilddaten zurückgegeben werden kann.

Bei beiden Funktionen wird ein Error geworfen, wenn die gesuchte ID nicht in der Datenbank existiert.

Eine weitere Funktion sortiert die Bäume in der Datenbank nach ihrem Elo-Rating und gibt die Bäume mit dem höchsten Ratings zurück. Die Anzahl der zurückgegebenen Bäume kann mit einem parameter bestimmt werden. Wenn der parameter höher als die Anzahl der vorhandenen Bäume ist, werden einfach alle Bäume zurückgegeben. Dabei werden wieder die Bilddaten rausgelassen.

Als letztes gibt es eine Funktion, die einfach alle Bäume aus der Datenbank lädt und zurückgibt. Auch hier wieder ohne Bilddaten.

EloLogic

EloLogic berechnet die neuen Elo-Zahlen, die aus den Bewertungen der Bäume entstehen. Dafür wird das von Arpad Elo Modell genutzt, welches ursprünglich für den Amerikanischen Schachverband USCF entwickelt wurde. Das Grundprinzip dahinter ist, dass jedem Spieler (hier jedem Baum) eine Rating-Zahl zugewiesen wird. Je besser der Baum, desto höher die Zahl. Nach jeder Begegnung werden die Zahlen beider Bäume angepasst, sodass die Elo-Zahl des gewinnenden Baumes erhöht wird und die des verlierenden Baumes nideriger wird.

Dafür wird im ersten Schritt die Wahrscheinlichkeit berechnet, mit der der jeweilige Baum die Bewertung gewinnt. Dies passiert nach folgender Formel.

$$E_A = \frac{1}{1 + 10^{\frac{R_B - R_A}{400}}}$$

E_A : Der erwartete Punktestand für Baum A

R_A : Die bisherige Elo-Zahl von Baum A

R_B : Die bisherige Elo-Zahl von Baum B

Dies wird sowohl für Baum A als auch für Baum B ausgerechnet. Dabei gilt immer:

$$E_A + E_B = 1$$

Mit diesen Erwartungswerten werden dann die angepassten Elo-Zahlen berechnet. Dabei kriegt ein Baum bei einem Sieg je mehr Punkte, desto unwahrscheinlicher sein Sieg, also desto niedriger sein Erwartungswert ist. Umgekehrt verliert er auch bei einer Niederlage mehr Punkte, wenn sein Erwartungswert sehr hoch ist.

Dies geschieht über folgende Formel:

$$R'_A = R_A + 20 \cdot (S_A - E_A)$$

R'_A : Die neue Elo-Zahl von Baum A

R_A : Die alte Elo-Zahl von Baum A

20 : Die Zahl 20 legt den maximal möglichen Rating-Gewinn und -Verlust fest.

S_A : Die tatsächlich erreichte Punktzahl des Baumes (1 für einen Sieg und 0 für eine Niederlage)

Die neue Elo-Zahl wird für beide Bäume einer Bewertung berechnet. Und danach in der Datenbank angepasst.

TreeCognition

Ziel des Services ist die Erkennung von Bäumen auf Bildern mit Googles Image Classification Service. Für dieses Service wird auch die `gcloud.json` benötigt. Jeder hochgeladene Baum wird an Google Cloud gesendet und es wird überprüft, was sich auf dem Bild befindet und wenn das Stichwort "Tree" sich in der Liste befindet, wird der Baum als echter Baum gewertet.

Anleitung zum Starten der Anwendung

Im Folgenden wird erklärt, wie die Anwendung entweder im "Production" Modus oder im "Developing" Modus gestartet werden kann. Der Production Modus ist optimiert und leichter zu starten.

Vorraussetzung: Google Cloud Account

Um das Backend richtig nutzen zu können, wird ein Google-Cloud-Konto benötigt.

Bitte folgen Sie dem Tutorial für die Vision-API-Authentifizierung und Projekteinrichtung auf dieser Webseite. Nachdem Sie erfolgreich ein Konto erstellt haben, laden Sie die Konfigurationsdatei herunter, benennen Sie sie in `gcloud.json` um und fügen Sie sie in den `server` Ordner ein.

Run (Production)

Notiz: Bitte stellen Sie sicher, dass Sie die `docker-compose` ausführen können. Um Docker zu installieren, besuchen Sie die [Docker Website](#).

Um die Applikation im **Production**-Modus zu starten, führen Sie einen der folgenden Befehle aus.

```
cd docker
docker-compose up

# Alternativ
docker-compose -f docker/docker-compose.yaml up
```

Dies kompiliert den TypeScript-Server, erstellt den Ionic-Client und optimiert die Anwendung für den Produktionsmodus. Dies kann etwas dauern, also abwarten und Tee trinken. Dann startet der TypeScript Express-Server und bedient den statischen Ionic-Client. Die Anwendung sollte auf `localhost` (Port 80) starten.



Die Datenbank ist beim Start der Anwendung leer. In server/treeexamples sind fünf Beispiel-Bäume die innerhalb der Applikation in die Datenbank geladen werden können. Dabei kann mit Baum auch direkt die TreeCognition getestet werden, da er nicht als Baum erkannt wird.

Run (Development)

Um das Projekt in VS Code zu öffnen kann das Projekt über die `tannder.code-workspace` Datei über folgenden Befehl geöffnet werden.

```
code ../vscode/tannder.code-workspace
```

Danach sollten alle vorgeschlagenen Erweiterungen installiert werden. Dies wird sowohl ein automatisches linting als auch ein automatisches Formatieren bei jedem Speichern ermöglichen.

Bewährte Praxis: MongoDB in einem Container starten

```
cd server
npm run start:db
```

oder nativ

```
docker run --rm -p 27017:27017 --name tannder-dev-db -e MONGO_INITDB_ROOT_USERNAME=admin -e MONGO_INITDB_ROOT_PASSWORD=admin mongo:latest
```

Wichtig zu beachten ist, dass die Datenbank bei jedem neuen start leer ist!

Notiz: Bevor der `server` gestartet wird, sollte sicher gegangen werden, dass die MongoDB gestartet hat.

Um den `server` im Entwicklungs-Modus zu starten, führen Sie folgenden Befehl aus:

```
cd server
npm run start:dev
```

Sie müssen den `ionic-cli` global installieren:

```
npm install -g @ionic/cli
```

Dann kann der `client` im Entwicklungs-Modes gestartet werden:

```
cd client
npm run start
```

Um einen beliebigen `Client` - oder `Server` -Code zu linten, führen Sie `npm run lint` aus. Zum Formatieren verwenden Sie `npm run format`. Wenn Sie jedoch das Projekt als VS Code Workspace mit der Datei `../vscode/tannder.code-workspace` geöffnet haben, sollte der Code beim Speichern formatiert werden.

Vielleicht müssen Sie `eslint` global mit installieren:

```
npm i -g eslint
```

