Entwurf für Bilderkennungs App

Dokument-Daten			
Schule	CsBe Bern		
Klasse	Bis18/P		
Autor	Michal Durik		
E-Mail	mdurik2@gmail.com		
Ausgabedatum	22.03.2020		
Version	V1.0		
Status	☐ In Arbeit	☐ In Prüfung	☒ Genehmigt, zur Nutzung
Klassifizierung	☐ Vertraulich	☒ Intern	☐ Zur Veröffentlichung

Tabelle 1 – Dokument-Daten

1. Änderungsprotokoll

Version	Datum	Name	Beschreibung
V1.0	22.03.2020	Michal Durik	Vorlage mit Inhalt erstellen

Tabelle 2 - Änderungsprotokoll

2. Inhaltsverzeichnis

3. Funktionelle Anforderungen

- Die Anwendung muss mindestens einen Service von Google Vision Api oder AWS Rekognition beziehen.
- Die Technologie für das Frontend ist frei wählbar
- Die Anwendung verwendet stabile Versionen der eingesetzten Libraries
- Der Entwurf enthält eine Beschreibung der Anwendung
- Der Entwurf enthält eine Beschreibung der Zielgruppe an Usern
- Der Entwurf enthält Mocks für die Benutzeroberfläche
- Der Entwurf beschreibt die Funktionalität der Anwendung
- Der Entwurf enthält eine Beschreibung der wichtigsten Überlegungen und die notwendigen Schritte für die Umsetzung
- Der Entwurf enthält zu den UI Mocks alle Use Cases und Interaktionen als Diagramme
- Der Entwurf enthält Beschreibung der Tests und das Vorgehen beim Testing
- Die Anwendung verwendet einen File Uploader für **die Bilder und/oder Videos**
- Die Anwendung ist responsive und kann auch auf mobilen Geräten verwendet werden

4. Technologien

4.1 Back-End

```
"dependencies": {
  "@google-cloud/vision": "^1.11.0",
  "cors": "^2.8.5",
  "dotenv": "^8.2.0",
  "express": "^4.17.1",
  "helmet": "^3.21.3",
  "morgan": "^1.9.1",
  "multer": "^1.4.2",
  "nodemon": "^2.0.2"
}
```

Abbildung 1 – Technologien, Back-End

4.2 Front-End

```
"dependencies": {
  "axios": "^0.19.2",
  "core-js": "^2.6.5",
  "vue": "^2.6.10",
  "vue-router": "^3.0.3",
  "vuetify": "^2.2.17",
  "vuex": "^3.0.1"
},
"devDependencies": {
  "@vue/cli-plugin-babel": "^3.12.0",
  "@vue/cli-plugin-eslint": "^3.12.0",
  "@vue/cli-service": "^3.12.0",
  "@vue/eslint-config-prettier": "^5.0.0",
  "babel-eslint": "^10.0.1",
  "eslint": "^5.16.0",
  "eslint-plugin-prettier": "^3.1.0",
  "eslint-plugin-vue": "^5.0.0",
  "node-sass": "^4.12.0",
  "prettier": "^1.18.2",
  "sass": "^1.19.0",      You, 5 days ago
  "sass-loader": "^8.0.0",
  "vue-cli-plugin-vuetify": "^2.0.5",
  "vue-template-compiler": "^2.6.10",
  "vuetify-loader": "^1.3.0"
},
```

Abbildung 2 - Technologien, Front-End

5. Mock-Up

Ich habe Mock-Up in **Adobe XD** erstellt.

5.1 Verwendete Farben:

- #5da5ef
- #e0e0e0
- #6140b1
- #3777cc
- #000000
- #ffffff

5.2 Bilden

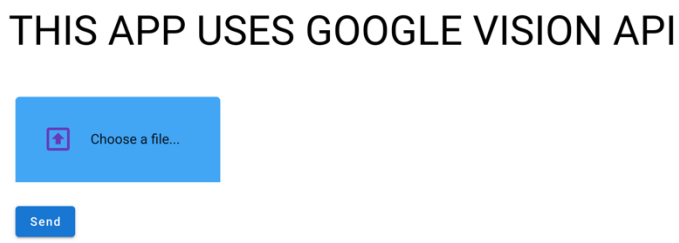

Abbildung 3 - Mock-Up, Home

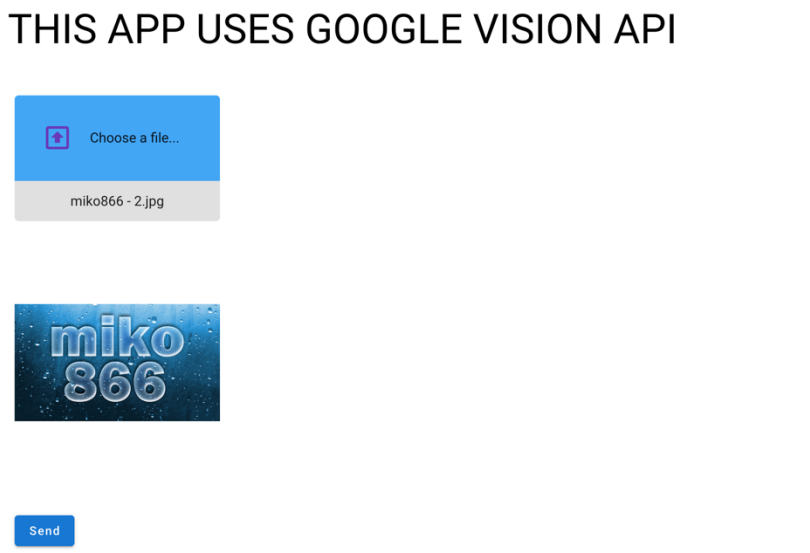

Abbildung 4 - Mock-Up, Upload image

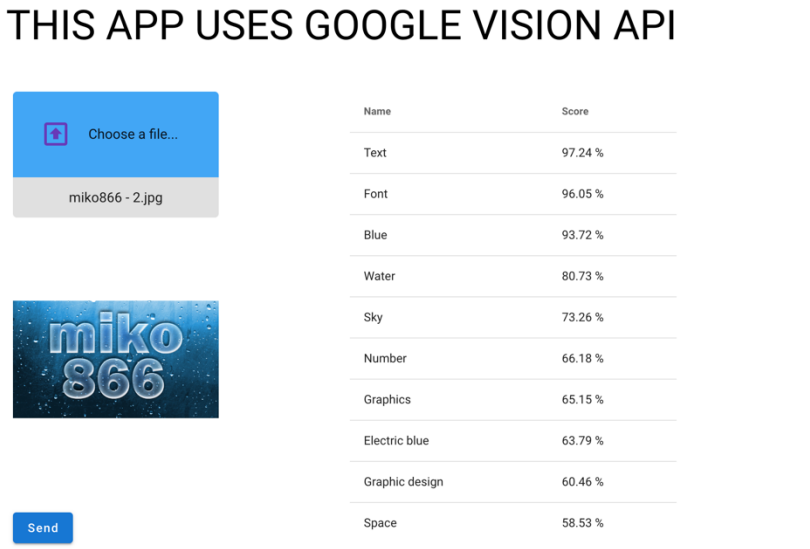

Abbildung 5 - Mock-Up, Show data

6. Use-Case Diagramm

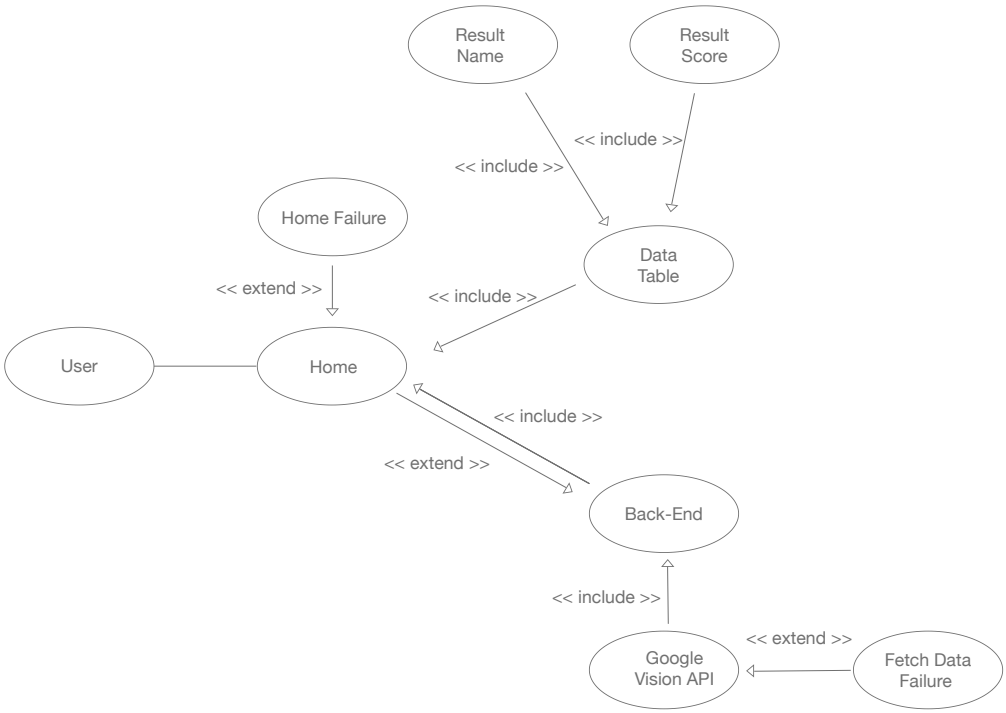

Abbildung 6 - Use-Case Diagramm

7. Sequence Diagramm

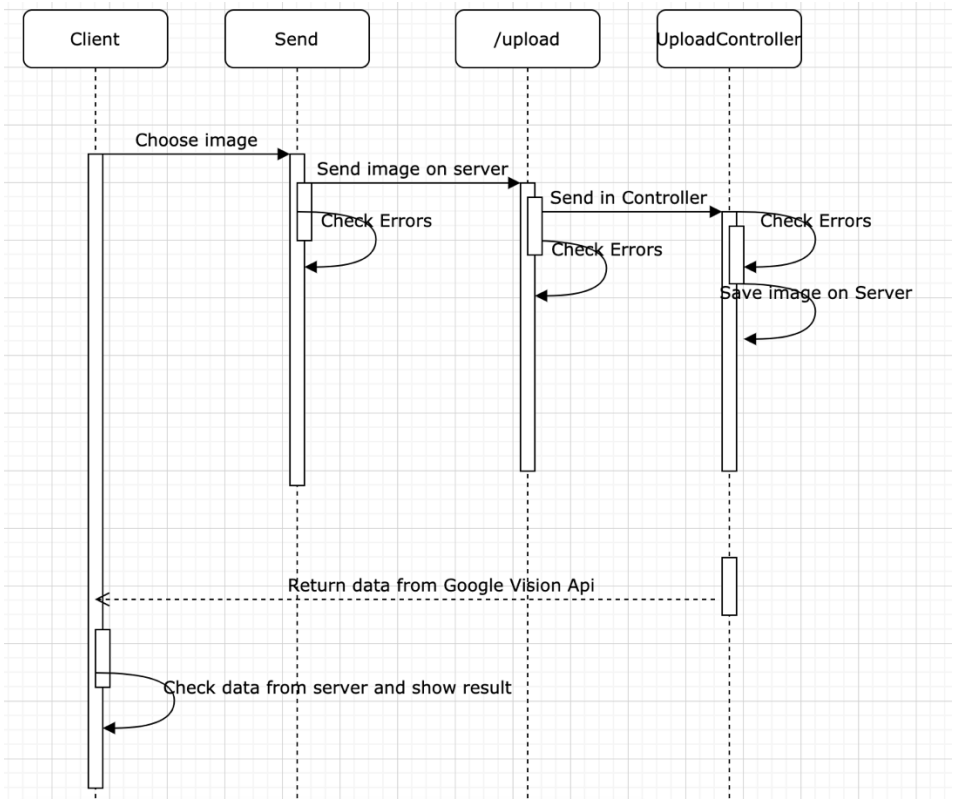

Abbildung 7 - Sequence Diagramm

8. Rest API

In meinem Projekt verwende ich nur einen Rest-End Point, welcher sendet ein Bild von der Client an den Server.

End Point	Type	Params	Beschreibung
/upload	POST	FormData	Senden Sie das Bild an den Server

Tabelle 3 - Rest API

8.1 Fehler

8.1.1 Client

End Point	Type	Beschreibung
/upload	500	Server error. Fehler bei fetch Data von Server.

Tabelle 4 - Rest API Fehler Client

8.1.2 Server

End point	Type	Beschreibung
/upload -> routes	422	Das Bild ist zu gross. Die maximale Grösse beträgt nur 50 MB.
/upload -> controller	500	Server error. Fehler Send Data for Client.

Tabelle 5 - Rest API Fehler Server

9. Test

Testfall	Resultat	Beschreibung
Starten der App	Erfolgreich	Die Webanwendung wurde ohne Probleme geladen.
Bild auswählen	Erfolgreich	Das Bild wurde ohne Probleme ausgewählt.
Bild-Preview	Erfolgreich	Das ausgewählte Bild wird als Vorschau angezeigt.
Ergebnis	Erfolgreich	Nach dem Drücken der Senden-Taste wurden die Daten an den Server gesendet und von Google-Api weiterverarbeitet. Daten von Google Api wurden weiter an den Client zurückgesendet, wo sie in der Tabelle angezeigt wurden.
Mobile-Version	Erfolgreich	Getestet mit Google Chrome, Opera, Firefox und Safari.

Tabelle 6 - Test

10. Beschreibung

10.1 Anwendung mit Ziel Gruppe

Die Benutzeroberfläche ist sehr einfach und intuitiv. Es erfolgt keine Anmeldung, da die App jedem Benutzer zur Verfügung stehen soll.

Nach dem Laden der ersten Seite sieht der Benutzer nur den Titel und eine grosse Schaltfläche zum Hochladen des Bildes. Nach dem Hochladen des Bildes wird eine Vorschau mit einer Schaltfläche zum Senden darunter angezeigt. Das Bild wird an den Server gesendet, wo es von Google Vision Api verarbeitet wird, und gibt die Daten zurück, die dann an den Benutzer weitergeleitet werden. Dort wird es auch in Form von Name und Punktzahl angezeigt, die die prozentuale Darstellung einzelner Elemente im Bild darstellen.

10.2 Umsetzung

Zuerst habe ich Mock-Up in AdobeXD erstellt, um eine bessere Vorstellung zu bekommen, als der Client aussehen soll. Dann habe ich ein Anwendungsfalldiagramm erstellt, wie die Anwendung funktionieren soll.
Als ich das Mock-Up- und Use-Case-Diagramm ausgefüllt hatte, erstellte ich einen neuen Api-Key für Google Vision API und registrierte ich auch den Dienst, um ihn zu verwenden. Dazu musste ich noch Billings aktivieren.

10.2.1 Back-End

Ich habe ein Back-End namens Server erstellt, auf dem ich Express und Google Vision Api für Javascript verwendet habe. Auf Servern habe ich neue Ordner-Controller erstellt, in denen sich die .js Datei befindet, die die Google Vision-API steuert. Dazu brauchte ich die Routes, die ich in index.js registriert hatte. In Controller habe ich nach Erhalt der Response von Google Servern diese verarbeitet und die daraus resultierenden Daten an den Client gesendet. Ich musste einen Ordner erstellen, in dem ich Bilder speichern werde, die vom Client hierher kommen. Ich habe die Multer.js-Bibliothek verwendet, um Bilder zu verarbeiten.

10.2.2 Front-End

Ich habe Client als Vue.js App erstellt und dann Vuetify hinzugefügt. Ich habe den Assets-Ordner erstellt, der einen anderen Ordner für den CSS-Style enthält. Im Ansichtsordner habe ich Home.js, das über App.js geladen wird, wo ich die Routenfunktion definiert habe.

In Home.js wird eine Komponente verwaltet, SimpleUpload.Vue, die im Komponentenordner definiert ist. Es definiert den gesamten Front-End als das Hochladen eines Bildes auf einen Server und einer Tabelle, in der Daten angezeigt werden. Nachdem ich das Bild hochgeladen habe, packe ich es in formData und sende es an den Server. Um es anzuzeigen, muss ich es in das Base 64 Bild wiederholen, damit es dem Benutzer angezeigt wird.

11. Project Struktur

```
├── README.md
├── backup-whateverfolder-2020-03-22.txt
├── client
|   ├── babel.config.js
|   ├── package-lock.json
|   ├── package.json
|   ├── public
|   |   ├── favicon.ico
|   |   └── index.html
|   ├── src
|   |   ├── App.vue
|   |   ├── assets
|   |   ├── components
|   |   ├── main.js
|   |   ├── plugins
|   |   ├── router.js
|   |   ├── store.js
|   |   └── views
|   └── vue.config.js
└── server
    ├── APIKey.json
    ├── controllers
    |   └── UploadController.js
    ├── image
    ├── index.js
    ├── package-lock.json
    ├── package.json
    └── routes
        └── Upload.js
```

12. Abbildungsverzeichnis

13. Tabellenverzeichnis