Dokumentation: Projekt Mediaportal

Inhaltsverzeichnis

Anforderungsanalyse 1
Frontend 3
Backend 4
Hosting 6
Praktisches Beispiel 6
Links zum Source-Code 7

Anforderungsanalyse

1. **Sich selbst bzw. eigene Firma vorstellen**
   MediaPortal: Das MediaPortal ist eine Internet-Plattform, die den Usern ermöglicht, schnell und werbefrei, Videos im gewünschten Dateiformat von der Plattform YouTube (https://www.youtube.com) zu downloaden. Auch ist das Konvertieren von schon vorhandenen Mediadateien in andere Medienformate möglich.
2. **Was ist meine Zielgruppe?**
   Zur Zielgruppe gehören alle Menschen, welche Mediendateien von YouTube downloaden oder einfach nur konvertieren wollen. Das Alter der Benutzer spielt dabei keine Rolle. Das Webportal ist für eine breite Zielgruppe ausgelegt, der Benutzer muss also kein Experte sein, um die Seite zu bedienen. Deshalb ist eine einfache und intuitive bzw. selbstsprechende Navigation nötig. Die Website soll von Desktopgeräten, aber auch von Mobilgeräten aus, einwandfrei verwendbar sein.
3. **Welcher Inhalt soll präsentiert werden? (Art der Website)**
   Auf der Website soll ein Online-Tool präsentiert werden.
4. **Wie ist die Struktur der Website?**
   Das Webportal kann grundsätzlich in zwei Hauptteilen gegliedert werden. Der erste Teil hat ausschließlich die Aufgabe den User über das Produkt zu informieren und ihn davon zu begeistern. Hier werden Preislisten, Feature-Listen und eine Registrierungs- bzw. Login-Funktion realisiert. Der zweite Teil der Website implementiert die Features des Online-Tools selbst, was innerhalb eines geschützten Kundenbereichs realisiert wird. Nach einem erfolgreichen Login kann der Benutzer Einstellungen an seinem Account tätigen und die Tools selbst dann auch verwenden.
5. **Funktionalität der Website (Use Cases)**
   - Ein User kann sich auf der Homepage über das Tool informieren
     - Eine Preisliste zeigt welche Features zu welchem Preis verfügbar sind

    - Ein Impressum zeigt relevante Informationen über die Websitebetreiber an
- Ein User kann ein neues Nutzerkonto anlegen (registrieren)
    - Username, E-Mail und Passwort müssen angegeben werden
- Ein User kann sich mit seinem Nutzerkonto authentifizieren (anmelden)
    - Username und Passwort müssen angegeben werden
- Ein eingeloggter User kann sein Passwort abändern
    - Das alte und das neue Passwort müssen angegeben werden
- Ein eingeloggter User kann ein YouTube-Video downloaden
    - Ein valider YouTube-Link muss angegeben werden
    - Das Dateiformat, in welchem die Datei nach der erfolgreichen Konvertierung vorliegen soll, muss angegeben werden
- Ein eingeloggter User kann eine Mediendatei konvertieren
    - Eine Video- bzw. Audiodatei muss angegeben werden
    - Das Dateiformat, in welchem die Datei nach der erfolgreichen Konvertierung vorliegen soll, muss angegeben werden
- Ein eingeloggter User kann sich ausloggen
- Ein User erhält jederzeit Feedback über erfolgreiche und nicht erfolgreiche Operationen

6. **Soll die Website für versch. Sprachgruppen zugänglich sein?**
   Die Website soll auf einfach gehaltenem Englisch angezeigt werden, um eine große Zielgruppe ansprechen zu können. Da verhältnismäßig wenig Text auf der Website dargestellt wird und dieser durch geeignete graphische Mittel intuitiv beschrieben wird, sind keine weiteren Übersetzungen für spezifische Sprachgruppen geplant. Der Aufwand der Übersetzung würde die Zielgruppe nicht wirklich vergrößern.
7. **Wie wird die Aktualität der Website gewährleistet?**
   Das Webportal beinhaltet keine essenziellen Inhalte, deren Aktualität schnell schwindet. Da hier lediglich ein Tool implementiert wird, stellt die Aktualität keine Problematik dar. Sicherheits- und datenschutztechnische Aspekte müssen natürlich jederzeit auf dem neuesten Stand gebracht werden, während sich der angezeigte Inhalt der vorhandenen Features nicht ändern dürfte.
8. **Wie wird die Website veröffentlicht?**
   Das Frontend wird mithilfe eines Docker-Containers gehostet. Im Container sorgt der NGINX-Webserver dafür, dass die Website auf dem Port 443, für alle über https erreichbar ist und ordnungsgemäß angezeigt wird. Das Rust-Backend läuft auf einem Server, welches Intern wiederum einen lokalen MySQL-Server zugreift.
9. **Gibt es Konkurrenz bzw. vergleichbare Websites?**
   Es gibt eine große Anzahl an vergleichbaren Websites, wie z.B. notube.cc, yt2conv.com, ytmp3.cc, youtubetomp3.digital. Allerdings haben alle diese Webseiten gemeinsam, dass entweder die Tools nicht richtig funktionieren und/oder mit unverhältnismäßig viel Werbung spamen. Zusätzlich besteht die Gefahr, dass beim Download eine Maleware mitinstalliert wird.
10. **Budget und Deadline**

Um die Website zu hosten, wurden 2,19€ für das Mieten des Servers ausgegeben. Deadline des Projekts ist Donnerstag, der 18.11.2022.

Frontend

Um das Frontend der Website zu realisieren, haben wir uns für das Framework Angular entschieden. Das Framework ermöglicht modernes Web-Development mit Typescript und kümmert sich zugleich um die Browserkompatibilität und um die Effizienz der Website. Wir haben uns hierbei für die neueste Angular Version 14.2.4 mit Node.js 16.18 und dem Node Package Manager (npm) 8.19.2 entschieden. Nachfolgend werden die wichtigsten Aspekte des Angular-Frontends aufgezeigt.

Die Projektstruktur

Unter /src in der default Angular-Struktur finden wir die wichtigsten Dateien eines Angular-Projekts:

- favicon.ico enthält das Icon, welches am Browser Tab erscheint
- index.html enthält die leere Grundstruktur der Website. Nachdem Fonts und Icon geladen wurden wird hier <app-root></app-root> aufgerufen, welches die eigentliche Website-Struktur aus dem /app-Ordner lädt, wie wir später noch sehen werden
- main.ts, polyfills.ts, styles.css und test.ts initialisieren die Webseite, sodass diese auch für verschiedene Browser kompatibel ist
- Im Ordner /environments werden Environment-Variablen abgespeichert, wobei hier zwischen Production und Development unterschieden werden kann
- Der Ordner /assets beinhalten alle weiteren Bilder, Icons und Grafiken, welche durch die eigentliche Webapp geladen werden

Der Ordner /src/app beinhaltet schließlich die Dateien, welche für den angezeigten Inhalt direkt verantwortlich sind:

- _helpers beinhaltet Dateien, welche einzelne Hintergrund-Funktionalitäten implementieren und für die Sauberkeit des Codes hier ausgelagert wurden
- _models enthält die Typdefinitionen von Typescript-Klassen, welche von den Components dann instanziiert werden können
- _services beinhaltet Services, welche von den Components direkt verwendet werden können
- alle anderen Ordner implementieren genau ein Component. Jedes Component hat folgende Dateistruktur
    - in der .html-Datei wird die DOM-Struktur des Components definiert
    - in der .scss-Datei werden die Styles festgelegt

    - in der .ts-Datei werden die eigentlichen Funktionalitäten implementiert (z.B. Drücken eines Buttons)
    - in der .spec.ts-Datei sind automaische Tests definiert, welche mit dem Framework karma.js ausgeführt werden können

- app.component.html/.scss/.spec.ts/.ts implementieren dass <app-root> Component, welches durch die index.html-datei geladen wird und selbst den Output des Routing-Modules lädt
- app.module.ts handelt die Imports der Components und Librarys
- app-routing.module.ts regelt das Routing der Webpage, sodass Angular weiss, unter welchen Umständen, welches Component geladen kann bzw. darf

Das Authentifizierungs-System
Es gibt eine Großzahl an Angular-Frameworks, welche ein Authentifizierungssystem realisieren. Allerdings haben wir uns dazu entschieden die Authentifizierungsmanagement selbst zu implementieren.
Drückt der User auf den Registrieren-Button werden die Eingaben validiert und diese an den AuthenticationService weitergegeben. Der Service erstellt mit diesen Daten ein API-Call. War das Registrieren nicht erfolgreich (also Response-Code != 200), so wird der Backend-Fehler vom error.interceptor.ts abgefangen, gehandelt oder an den Benutzer direkt ausgegeben. Bei einer erfolgreichen Registrierung ruft das SignUp-Component mit denselben Daten die login()-Funktion des Authentication-Service auf. Der Service erstellt mit diesen Daten ein API-Call auf dem Login-Endpoint. War diese Operation erfolgreich, so wird aus der Backend-Response ein User-Objekt zusammengebaut und im lokalen Speicher des Benutzers abgelegt. Damit kann jederzeit ohne Cookies verifiziert werden, dass der User korrekt eingeloggt ist. Bei einem potenziellen Logout wird dieses User-Objekt aus dem lokalen Speicher wieder entfernt. Nach dem erfolgreichen Login navigiert das SignUp-Component automatisch zur Hauptseite des geschützten Kundenbereichs.
Damit ein nicht eingeloggter User nicht auf die geschützten Seiten zugreifen kann, wurde der auth.guard.ts implementiert. Der AuthGuard wird vom app-routing-module verwendet, um zu überprüfen, ob der aktuelle User die gewünschte Seite aufrufen darf oder nicht. Dabei wird das CanActive-Interface implementiert, um bestimmte Routen für den User zu blockieren oder zu aktivieren. Der AuthGuard überprüft über den AuthenticationService, ob der User eingeloggt ist oder nicht. Wenn ja, so wird der User auf die gewünschte Route weitergeleitet, falls nein, so findet sich der User auf der Login-Seite wieder. Im letzten Fall wird die angeforderte Route zwischengespeichert, sodass diese nach einem erfolgreichen Login automatisch aufgerufen werden kann.
Die Passwörter werden vom Frontend nicht gehasht, da die Daten über https automatisch mit SSL verschlüsselt werden. Das Backend speichert dann die Passwörter in der Datenbank in gehashter Form, um maximale Sicherheit zu gewährleisten.

Backend

Die zum Projekt zugehörige REST-API wurde in Rust realisiert und bildet das stabile Backend zum Frontend. Hierbei wurde Rust ausgewählt, da die fertige Binary sehr Memory effizient

ist und die zero-cost-abstractions der Sprache keine Latenz zuführen. Durch diese Gegebenheiten begnügt sich die API mit nur 5MB Ram. Die bildet die Basis für einen Webservice, welcher im Millisekunden-Bereich Operationen durchführen kann. Als Datenbank um die Archivierung und Organisation zu ermöglichen kann jedes SQL kompatible Datenbanksystem verwendet werden.

Da für die API ein Webserver benötigt wird wurden zuerst durch das Ausschlussverfahren mehrere Webservercrates getestet, bis schließlich die Wahl auf Actix-web fiel, welche einen sehr anpassbaren Webserver, logische Syntax und gute Performance mit an den Tisch brachte. Anschließend wurden die einzelnen Endpunkte per async Funktion implementiert und in Module aufgeteilt, um die Organisation zu erleichtern. So gibt es nun den [api-address]:/data/… Endpoint, welche sämtliche Datenoperationen wie den Download von der Datenbank, als auch das schon genannte Konvertieren zwischen Formaten entgegennimmt und auf die Requests entsprechend antwortet. Hierbei wird als Kommunikationsmethode zwischen Front- und Backend json verwendet. Eine Ausnahme hierbei ist der Verkehr von Dateien, welche vom Client zum Server via Multipart und vom Server zum Client via HttpBody erfolgt. Diese Daten werden vom Frontend anschließend in eine, für den Nutzer brauchbare, Datei umgewandelt.

Die Hauptpunkte des Projekts sind der Download von Onlinevideos und die Konvertierung von beliebigen Formaten in weiter verbreitete Formate wie mp3 oder mp4. Dies wird alles durch Tools im Backend realisiert.

Beim Download eines YouTube Videos wird eine Anfrage mit dem Link und dem gewünschten Format vom Frontend an die API geschickt. Hierbei wird zuerst das übergebene Format (1...3) überprüft und eine Kette an Argumenten aufgebaut. Anschließend wird im temporären Ordner ein weiterer Ordner mit eindeutigem Namen erstellt, in welchem ein CLI-Programm das Video herunterlädt. Sollte dies ohne Probleme funktioniert haben, wird die Datei vom Backend eingelesen und in der Datenbank entsprechend abgespeichert. Danach wird eine Antwort ans Frontend geschickt, in welcher alles Nötige übermittelt wird, um die Datei von der API anzufragen und dem Nutzer schlussendlich zukommen zu lassen. Sollte der Nutzer die Datei herunterladen wollen, kann eine Anfrage gestellt werden, auf welche hin die entsprechende Datei (insofern der Nutzer berechtigt ist auf diese zuzugreifen) aus der Datenbank geholt und an das Frontend zurückgeschickt wird, wo sie mit dem vom Backend generierten Namen versehen und heruntergeladen werden kann.

Beim Konvertieren ist der Vorgang ähnlich. Hierbei schickt das Frontend mittels Multipart die Datei und das gewünschte Ausgangsformat, worauf hin die API die Datei wieder in einen einmaligen, temporären Ordner schreibt und mit ffmpeg die Konvertierung durchführt. Sollte alles funktioniert haben wird die Datei erneut eingelesen und in die Datenbank eingespeichert. Durch eine Anfrage kann der Nutzer die Datei erhalten.

Da viele Faktoren im Backend sich ändern könnten (SQL Server Adresse / Benutzer) und das neu kompilieren nach Konfigurationsänderungen auf low spec Systemen nicht unbedingt machbar ist, wurde die Möglichkeit eingeführt, einen Einstellungsdatei zur Verfügung zu stellen in welcher viele Variablen, wie z.B. die SQL Anmeldedaten, der API-Port, usw. abgeändert werden können. Auch kann hier festgelegt werden, wo sich der temporäre

Ordner für die Dateiprozesse befindet. Sollte die API https benötigen, so kann das SSL aktiviert und ansonsten deaktiviert werden. Dabei können auch die Pfade der cert und key Dateien angegeben werden, um maximale Flexibilität zu ermöglichen.

Durch das sehr modulare Design der Software ist es durchaus möglich, das Frontend, die API und die Database auf drei separaten Servern zu nutzen. So läuft als kleines Experiment der SQL-Server, welcher für die Vorführung verwendet wird, unter einer ReactOS VM auf einem anderen Server. Dies ermöglicht es, die einzelnen Server den Anforderungen anzupassen, und bei Bedarf unabhängig zu skalieren.

Hosting

Das Frontend

Das Frontend wird mithilfe eines Docker-Containers auf einem Virtual Private Server (VPS) von https://signaltransmitter.de/ mit Ubuntu gehostet. Im Docker-Container läuft ein Nginx-Webserver welcher die vorher im production mode gebuildeten Files auf dem https-Port (443) anzeigt. Das Image des Containers wird mithilfe der DockerFile erstellt (Befehl: „docker build -t image-name .“). Der Container basiert dabei auf dem latest image von nginx und die nginx.conf und die gebuildeten Angular-Files in /dist/media-portal werden automatisch in das Image kopiert. Mit dem Befehl „docker run --name container-name -d -p 8080:443 image-name“ wird aus dem Image der fertige Container erstellt und eingeschalten. Damit die Website über https erreichbar ist, musste in der nginx.conf das SSL-Zertifikat hinzugefügt werden.

Das Backend

Das Rust-Backend läuft direkt auf demselben VPS. Dabei ist es über dem Port 8080 zu erreichen. Um die Sicherheit zu erhöhen, wurde auch hier ein SSL-Zertifikat zur sicheren Datenübertragung hinzugefügt. Die Datenbank (mysql), auf welcher das Backend zugreift, wurde aus Performance-Gründen auf einen weiteren Server ausgelagert, welcher mit React OS läuft.

Praktisches Beispiel

Man öffne im Browser ihrer Wahl https://web.hpschleppi.ga/convert/youtube und gebe dort eine YouTube-URL ein (z.B. https://www.youtube.com/watch?v=UFGO_OSkUVI). Man wähle das gewünschte Dateiformat (z.B. MP4 audio + video) und drücke auf den Submit-Button. Nach der erfolgreichen Konvertierung durch das Backend, wird im Frontend der Dateiname, sowie ein Download-Button angezeigt. Durch das Drücken auf den Download-Button, kann man den Speicherort auswählen und die Datei schlussendlich downloaden.

Man öffne im Browser ihrer Wahl https://web.hpschleppi.ga/convert/media und lade dort die zuvor heruntergeladene Mediendatei hoch. Man wähle das gewünschte Dateiformat (z.B. MP3) und drücke auf den Submit-Button. Nach der erfolgreichen Konvertierung durch das Backend, wird im Frontend der Dateiname, sowie ein Download-Button angezeigt. Durch das Drücken auf den Download-Button, kann man den Speicherort auswählen und die Datei schlussendlich downloaden.

Links zum Source-Code

Frontend

https://github.com/tfobz-informatik/website-hell-morandell

Backend

https://github.com/HellBjoern/mediaportal_api