Foodexperience

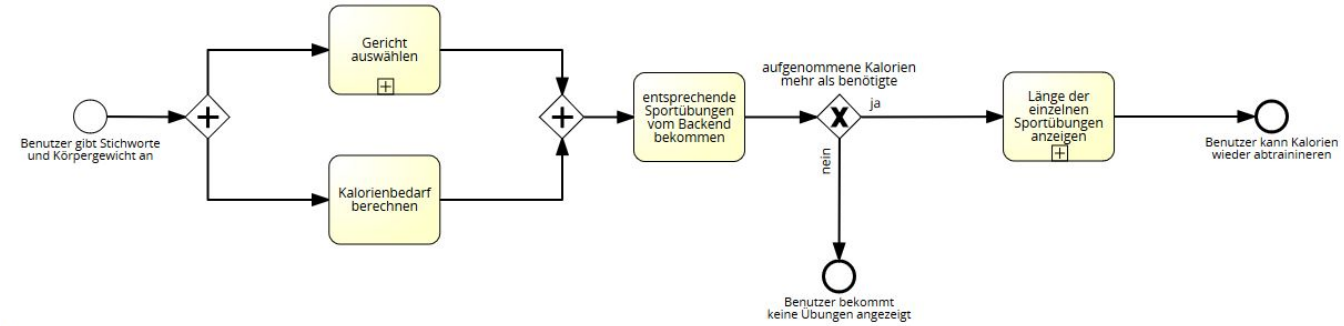
Hannes Philipp
Julia Riedinger
Simon Schwab
Dominik Welk

Unsere Idee

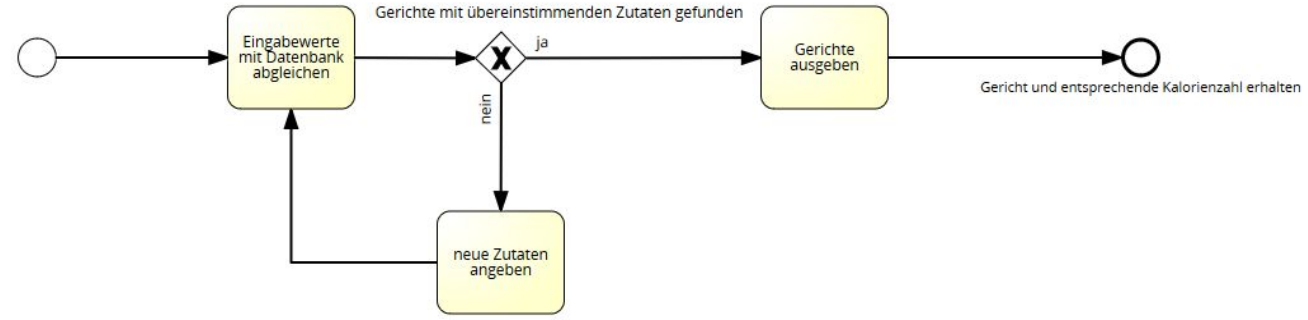
Jeden Tag neue Rezepte ausprobieren, dabei bereits vorhandene Nahrungsmittel nutzen und dabei die Gesundheit nicht aus dem Auge verlieren sehen wir als ein gutes tagtägliches Ziel, weshalb wir uns dazu entschieden haben ein Tool zu entwickeln, das genau dieses Ziel maßgeblich vereinfacht.
Durch die Eingabe des Gewicht des Users und dessen vorhandenen Nahrungsmitteln sollte so eine Auswahl an Rezepten, dessen Nährwerte und natürlich auch Aktivitäten ausgegeben werden, mit denen die aufgenommenen Kalorien wieder abtrainiert werden können.

Planung/BPMN (3928315)

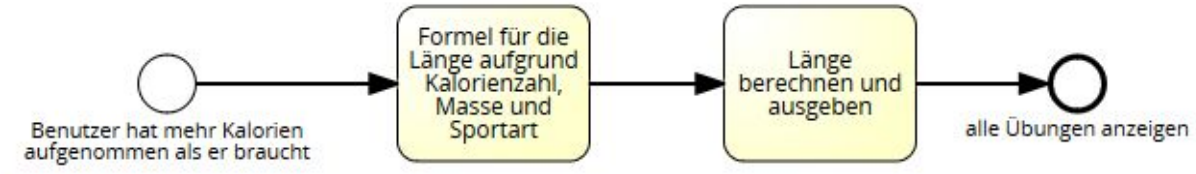
Hauptprozess:



Gerichte auswählen:



Länge der Sportübungen anzeigen:



Frontend (9032365)

Vorgängerversionen

Um ein Frontend zu erzeugen, welches zunächst den groben Design-Vorstellungen entsprechen und reine Funktionalität realisieren sollte, haben wir uns dazu entschieden ein reines Angular-Projekt zu erstellen um dieses später durch Angular Material zu erweitern und zu designen. Diese Version beruhte noch auf Routing über mehrere Seiten, allerdings wurde diese Funktionalität zugunsten der User-Experience in der finalen Version des Frontends entfernt.
Die zweite Version des Frontends sollte wie beschrieben lediglich die erste Webapplikation durch Angular Material erweitern. Zunächst war dies zunächst auch ohne Probleme realisierbar, allerdings stellte sich das Einbinden von komplexeren Angular Material Komponenten als problematisch heraus. Hierbei kam es vermehrt zu visuellen Bugs, wie dem Abweichen des Komponenten-Stylings von dem, wie es von Angular Material vorgesehen ist, oder auch zum nicht Anzeigen der kompletten Seite. Da dies sich trotz intensiver Beschäftigung mit dem Thema nicht so einfach lösen ließ, wurde eine dritte (finale) Version mit dem uns zuverlässiger erscheinenden Tool Bootstrap erstellt.

Aufbau der Website

Wie in dem Abschnitt “Vorgängerversionen” erwähnt haben wir uns aufgrund eines ansprechenderen Designs und besserer Übersichtlichkeit für die User gegen das Routing über mehrere Seiten entschieden und dieses Konzept durch eine Single-Site mit parallax scrolling ersetzt.
Der User startet hierbei auf der Landing-Area der Seite, auf dieser er entweder durch den “Get started”-Button direkt zu unserem Formular gelangt, oder einfach runter scrollen kann und sich somit noch eine nähere Beschreibung über die Idee der Webseite durchlesen kann.

Beim Formular angelangt kann der User nun seine Daten eingeben und erhält durch den Confirm-Button sein Ergebnis. Sollten die Daten nicht ausreichen oder sollte die Api mit den eingegebenen Parametern kein Ergebnis finden, wird der User darauf hingewiesen.
Bei den Ergebnissen angekommen kann der User durch einfaches Klicken auf eines der Ergebnisse nun die von uns vorgesehenen Details erhalten.
Intuitives Verhalten auf der Website wird durch schlichtes Design und durch das daraus resultierende leichte Erkennen wichtiger Abschnitte realisiert.

Datenverarbeitung

Die in dem Formular (Abbildung 1) eingegebenen Daten können direkt als String in die URL eingebunden werden, welche das Frontend aufruft, da das Verarbeiten und Zurückgeben der Daten im Backend asynchron zu dem Frontend verläuft, wird hierbei ein subscribe verwendet. Durch einfaches Databinding in der HTML kann nun die Result-area (Abbildung 2) durch das von dem Backend zurückgegebene JSON-Objekt dynamisch in einer Kachelansicht aufgebaut werden.
Um die Details zu den jeweiligen Ergebnissen anzuzeigen wird beim Anklicken eines Ergebnisses das dazugehörige JSON-Unterobjekt lokal abgespeichert und dadurch in der HTML geladen.

Probleme (Unterstützung durch 9493775)

Außer dem beschriebenen Problem bei der Verwendung von Angular Material traten noch kleine Schwierigkeiten aufgrund der Unerfahrenheit mit der Verwendung von JSON-Objekten auf, diese konnten jedoch relativ schnell durch Recherche in Dokumentationen gelöst werden.
Ein größeres Problem trat jedoch auf, als das direkte Testen des Backends durch das Frontend erfolgen sollte. Hierbei war es zunächst nicht möglich durch den Api-Aufruf ein JSON Objekt zu erhalten, der Grund dafür war ein Cross-Origin Error, welcher eine Verletzung der CORS-security beschreibt (Details zu dem Error: https://developer.mozilla.org/de/docs/Web/HTTP/CORS/Errors) und nur durch das

Öffnen des Browsers mit bestimmten Parametern im Developer-Modus umgehen ließ.
Da keiner innerhalb des Teams jemals mit solch einem Error sich beschäftigt hatte und auch die externe Informationsbeschaffung nicht sehr ausführlich war, beanspruchte die Behebung dieses Errors einige Zeit, konnte jedoch schlussendlich durch Angabe der erlaubten Ports im Backend gelöst werden.

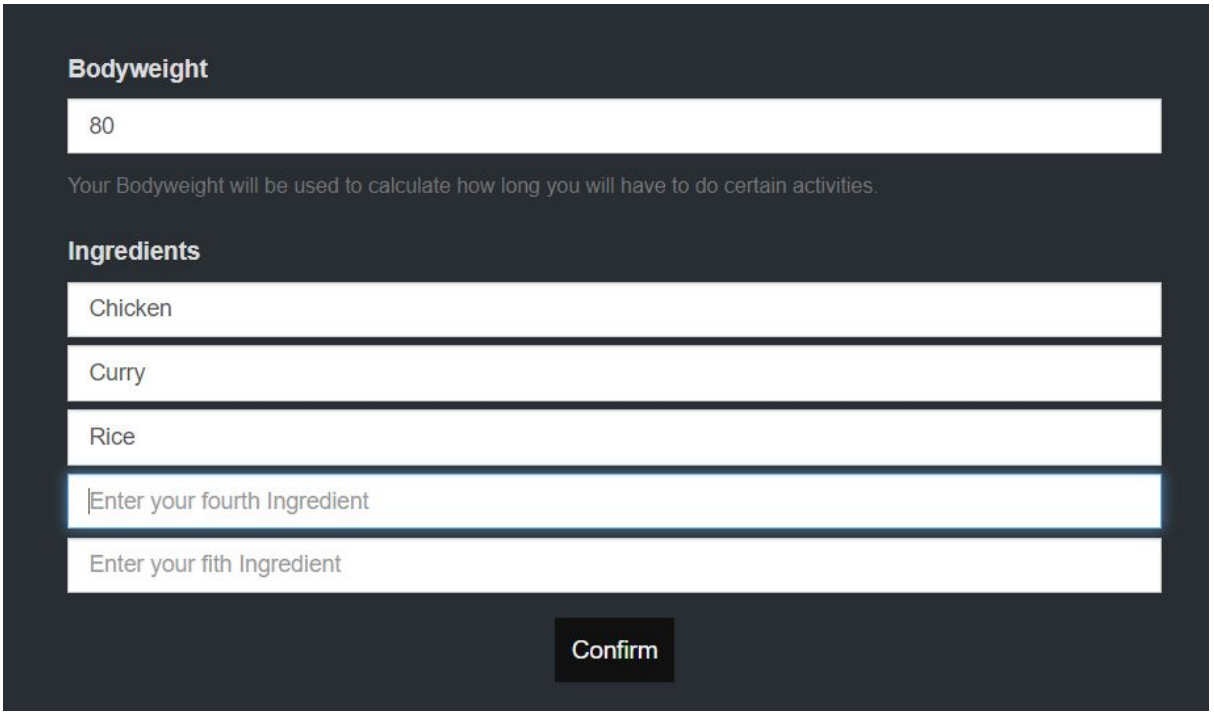


Abbildung 1

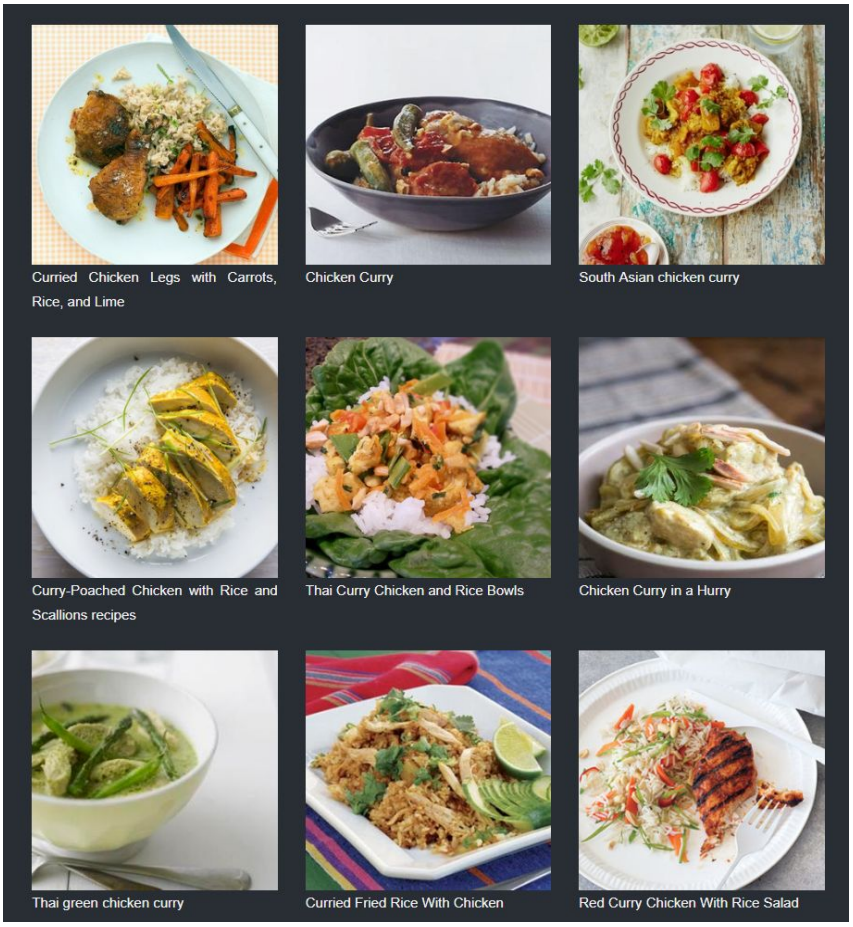


Abbildung 2

Backend (8136672)

APIs

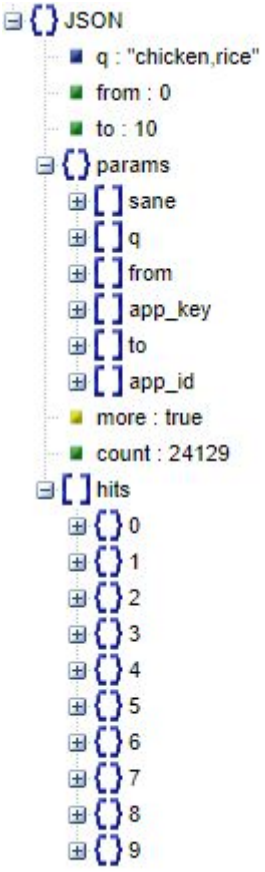
Um Foodexperience zu realisieren, werden im Backend zwei APIs zu einer neuen API vereint (Recipes API), diese liefert alle benötigten Daten.

Edamam API

Die Edamam API (https://www.edamam.com) wird als Datenquelle für die Rezepte verwendet.
Diese findet die Rezepte auf Basis von 5 gegebenen Stichworten. Außerdem werden eine ID, ein Key zur Identifikation und die Anzahl der Gewünschten Rezepte benötigt.

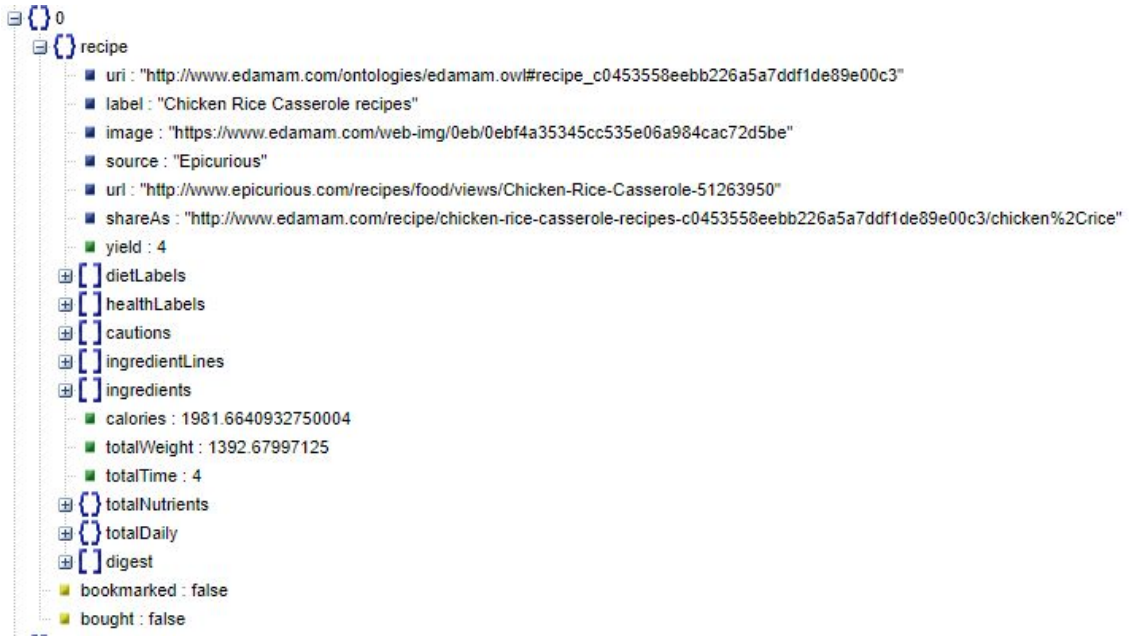
```
"https://api.edamam.com/search?q=" +
“chicken,rice” + "&app_id=" + id + "&app_key=" +
key + "&from=0&to=10"
```

Diese Beispielabfrage (noch im String-Format von Java) liefert 10 Rezepte (`"&from=0&to=10"`), welche die Zutaten Reis und Hühnchen enthalten.



Des Weiteren ist es bei der Edamam API möglich weitere Parameter zu übergeben, welche Rezepte herausfiltern können. Beispielsweise kann nach der benötigten Zeit oder “verbotenen” Zutaten, auf Grund von Allergien, gefiltert werden.

Als Rückgabe liefert die Edamam API eine Objektstruktur im JSON Format (siehe Bild). Diese enthält die Parameter des Requests (q, from, to, params), eine Angabe der gefundenen Resultate (count) und eine Array von Rezepten.



In diesem Bild ist der Aufbau eines “Rezept Objekte” aus dem Array zu sehen. In diesem sind unter anderem ein Bild (image) vom Gericht, die URL zum Rezept, die Anzahl an Personen für die das Rezepte gedacht ist (yield) angegeben. Weitere Angaben sind Nährstoffen, Kalorien etc., inkl. des Prozentsatz am Tagesbedarfs, und einige weitere Informationen.

Activity API

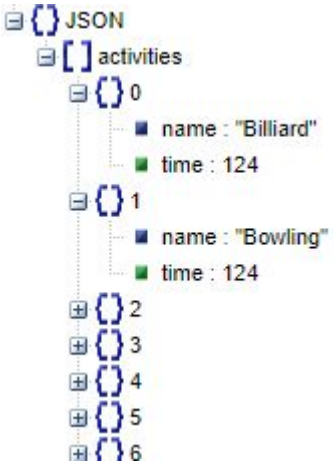
Die Activity API ist eine zweite API die neben der Recipe API im Backend des Projekts angesiedelt ist.
An die Activity API werden Requests bestehend aus der zu verbrennenden Kalorienmenge in kcal und dem eigenen Körpergewicht gesendet.
Über eine Formel wird daraus berechnet, wie lange bestimmte Tätigkeiten ausgeführt werden müssen, um die zu sich genommenen Kalorien zu verbrennen. Dies sind aber nicht nur sportliche Betätigungen wie Radfahren oder Joggen, sondern auch andere Aktivitäten wie zum Beispiel Billard Spielen.

Mittels der folgenden Formel wird die benötigte Zeit, abhängig von der Aktivität, berechnet: `60 * kcal / mass / activity`
“activity” ist eine Konstante die abhängig von der Sportart eingesetzt wird. “mass” gibt die Masse/Gewicht der Person an. “kcal” sind die zu sich genommenen Kalorien, in kcal. Mit 60 wird multipliziert um das Ergebnis in Minuten umzurechnen.

Eine denkbare Abfrage sieht wie folgt aus,

```
http://localhost:25525/activity?mass=80&kcal=496.71
```

Das “mass” gibt das Gewicht in kg an, dies sind im Beispiel 80 kg und “kcal” die Kalorienmenge in kcal, dies sind 496,71 im Beispiel.
Als Result wird eine JSON Datei geliefert. Diese enthält ein Array aus Objekten welches 24 verschiedene Sportarten enthält. Die Sportarten enthalten eine Bezeichnung und eine Zeit in Minuten die angibt wie lange sie ausgeführt werden müssten.



Sämtliche Daten zu den Sportarten und die oben genannte Formel stammen von folgender Website: www.rezeptrechner-online.de/blog/tag/formel-kalorienverbrauch-sport/ (2019.03.19).

Recipes API

Der Kern des Projekts steht die Recipes API, sie stellt die Requests an die Edamam als auch an die Activity API.

Eine Abfrage an die API sieht beispielsweise wie folgt aus,

```
localhost:25525/recipes?keyword=chicken,curry&mass=70
```

“chicken,curry” werden als Schlüsselworte für Rezeptsuche der Edamam API verwendet. “mass” wird als Berechnungsgrundlage der Aktivitäten an die Activity API weitergegeben. Es muss das Gewicht und mindestens ein Keyword übergeben werden.

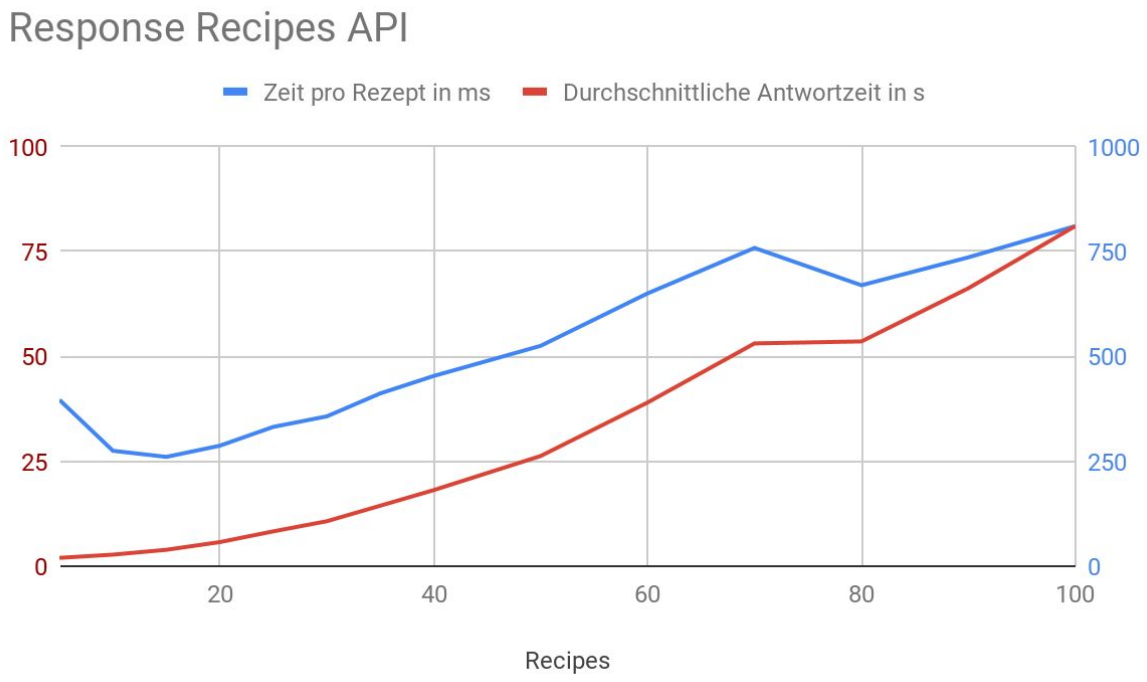
Mit der Information über die Kalorienzahl des Rezeptes und dem Gewicht des Users wird die Anfrage an die Activity API gestellt. Die von der Activity API erhaltenen Daten werden den rezepten hinzugefügt. Die Datenstruktur der Rezepte inkl. der Aktivitätsdaten wird als JSON zurückgegeben.

Probleme

Die meisten Probleme im Backend gab es dabei die Jackson Library. Diese wandelt JSON Strings in Objekte um und umgekehrt. Das Hauptproblem war eine gute Dokumentation zu finden, besonders im Fehlerfall. Letztendlich hat dies aber auch geklappt und das Backend funktioniert wie es soll.

Performancetest

Einen Performancetest im üblichen Sinne der Recipes API können wir nicht durchführen da es uns nur möglich ist 5 Requests pro Minute an an die Edamam API zu stellen.
Da wir aber eine Anzahl an Rezepten festlegen können die wir erhalten, haben wir einen Test durchgeführt der feststellt wie lange es dauert bis wir die fertige Rezeptliste mit unterschiedlich vielen Rezepten erhalten.
Die Zeit haben wir gemessen indem wir zwischen Aufruf und direkt vor der Rückgabe der Daten die Zeit erfasst und die Differenz berechnet haben.
Dadurch haben wir zwar nicht die Zeit, die der User auf seine Antwort warten muss, jedoch weicht diese nur sehr gering davon ab.
Bestimmt haben wir die Zeiten 3 mal für jeden 5. Wert von 5 bis 40 und danach für jeden 10. wert bis 100. Die daraus resultierenden Ergebnisse sind in der Abbildung zu sehen.



Docker (9493775)

Frontend

Um das Frontend in einem Docker-Container laufen zu lassen, muss zunächst ein Docker-Image erstellt werden. Hierzu wird eine Dockerfile erstellt, auf welche im folgenden genauer eingegangen wird.

```
1   # stage 1
2   FROM node:12.13 as node
3   # Create a directory where our app will be placed
4   RUN mkdir -p /app
5
6   # Change directory so that our commands run inside this new directory
7   WORKDIR /app
8
9   # Copy dependency definitions
10  COPY package*.json /app/
11
12  # Install dependecies
13  RUN npm install
14  RUN npm install -g @angular/cli@7.3.7
15
16  # Get all the code needed to run the app
17  COPY . /app/
18
19  # Expose the port the app runs in
20  EXPOSE 4200
21
22  # Serve the app
23  CMD ["npm", "start"]
```

Zunächst wird das node-Image mit der Version 12.13, in welcher auch das Frontend entwickelt wurde, gezogen. Dies wird benötigt, um Angular und seine Abhängigkeiten zu installieren und laufen zu lassen. Daraufhin wird ein Ordner erstellt, in welchem alle Daten gezogen werden, die zum Laufen benötigt werden. Darunter zählt unter anderem die Datei package*.json. In dieser Json sind alle Abhängigkeiten, welche die App benötigt, um zu funktionieren notiert.

Im darauffolgenden Schritt werden alle Abhängigkeiten, sowie das Angular CLI, mit Hilfe des Node Package Managers (npm) installiert. Daraufhin werden die Abhängigkeiten in den richtigen Ordner verschoben und der Port auf 4200 festgelegt. Zuletzt wird mit npm start der Webserver gestartet.

Backend

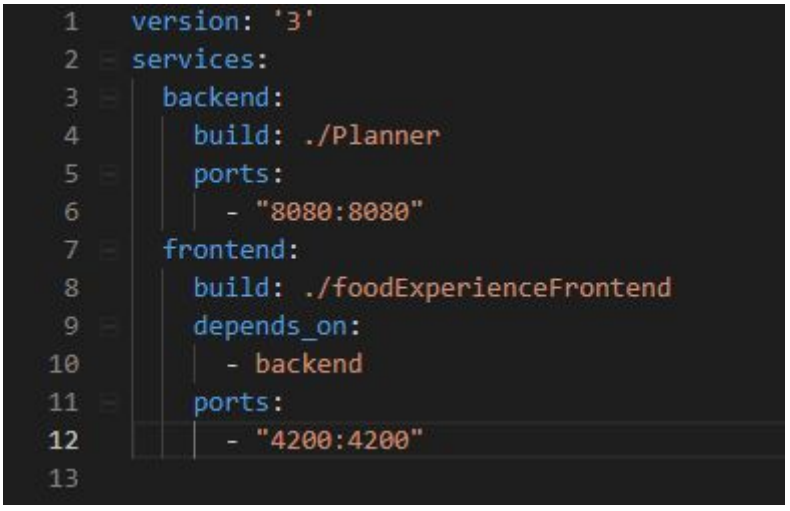
Die Dockerfile für das Backend ist sogar noch simpler.

```
1  FROM openjdk:8-jdk-alpine
2  VOLUME /tmp
3  ARG JAR_FILE=./foodExperience_Backend.jar
4  COPY ${JAR_FILE} app.jar
5  ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Hierbei wird zunächst das Image für die Java jdk gezogen. Daraufhin wird ein tmp Ordner erstellt und die zu verwendende Jar File ausgewählt. Diese wird daraufhin an den richtigen Ort kopiert und als Einstiegspunkt ausgewählt, wodurch der Webserver für das Backend gestartet wird.

Docker-Compose

Um Frontend und Backend auf einmal starten zu können, wird Docker-Compose verwendet, was mehrere Container starten kann. Es führt letztendlich nur die Build Prozesse für Backend und Frontend aus und weist ihnen erneut den richtigen Port zu. Außerdem werden Abhängigkeiten definiert.



```yaml
version: '3'
services:
  backend:
    build: ./Planner
    ports:
      - "8080:8080"
  frontend:
    build: ./foodExperienceFrontend
    depends_on:
      - backend
    ports:
      - "4200:4200"
```

Um die Container nun zu starten, muss man in Docker nur den Befehl ‘docker-compose up’ eingeben, insofern man sich im Ordner der docker-compose.yml befindet. Die Applikation kann daraufhin unter http://localhost:4200 aufgerufen werden.