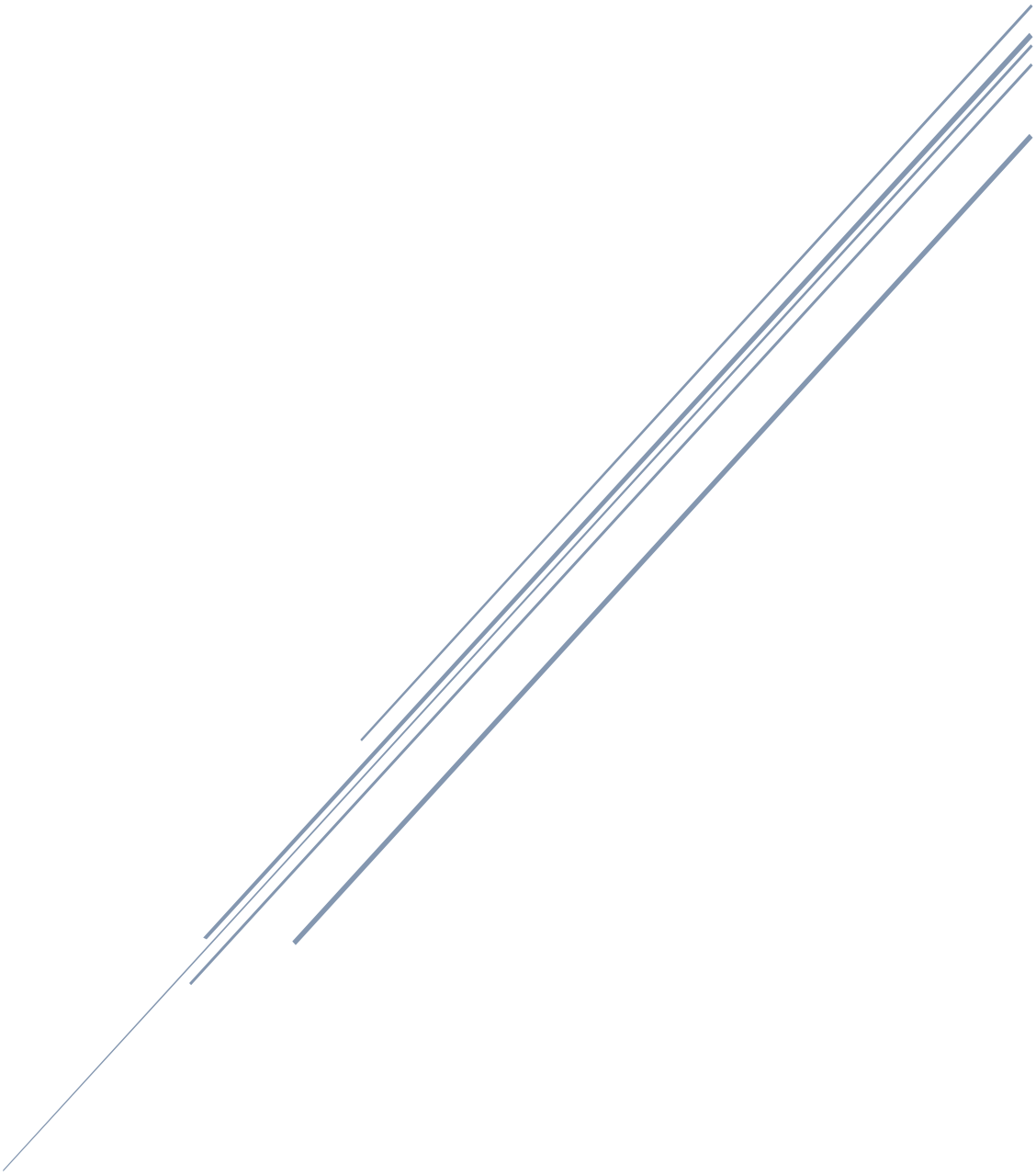
TRADENATION

Alle wichtigen Informationen zu Kryptowährungen und Aktien.



DHBW

INF20B

4873531, 8528264, 8233288

Inhalt

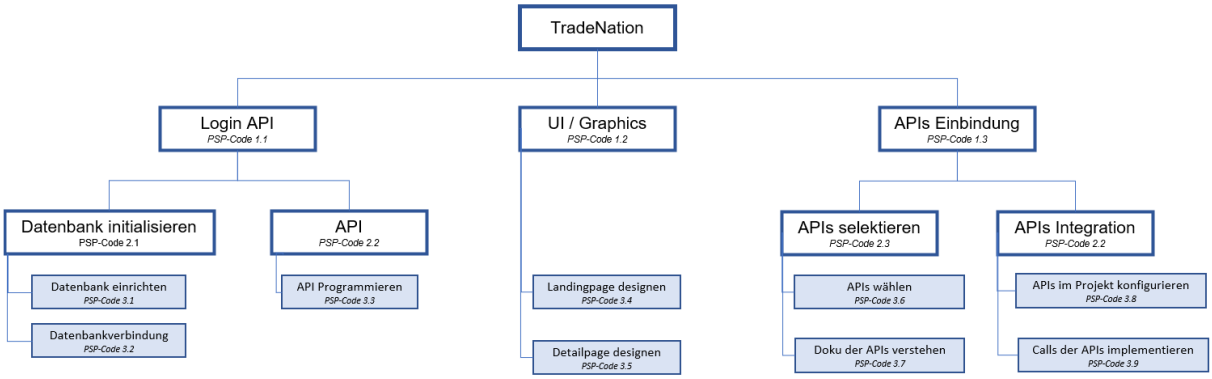
Organisatorische Daten

Gruppenmitglieder:

- Mohamed Haji 8528264
- Lukas Hautzinger 4873531
- Dominic Merkle 8233288

Projektplan

Zur nachvollziehbaren Strukturierung unseres Vorgehens beim Bearbeiten dieses Projekts haben wir einen Projektstrukturplan (Seite 2) erstellt. Das Projekt wurde in einzelne Arbeitspakete aufgegliedert und zwischen den Gruppenmitgliedern delegiert.



Grafik 1 Projektstrukturplan TradeNation

Dabei waren folgenden Aufgabenbereiche unter den Gruppenmitgliedern aufgeteilt:

Mohamed Haji:

- Erstellung/Testen der eigenen Login API
- Docker & Gitlab
- Planung Vorgehensweise & Organisation

Lukas Hautzinger:

- Implementierung der APIs (Abfrage)
- Erstellung React-App
- Dokumentationsarbeiten

Dominic Merkle:

- Erstellung Design von Website & Logo
- Recherche APIs
- Präsentation

Projektdokumentation

Zielsetzung

Die Erstellung eines auf APIs basierenden Web-Service welcher Nutzende über aktuelle Werte und Preisentwicklungen von Kryptowährungen und Aktien informiert. Das Ganze soll als Website umgesetzt sein, um einen einfachen Zugriff über einen Browser zu gewährleisten.
Rahmenbedingungen bei der Realisierung unseren Projekten waren folgende Konzepte, auf welche wir besonders Wert legten:

Modernes und Übersichtliches UI:

Das Userinterface soll ansprechend gestalten sein. Dabei soll vor allem auf eine moderne Designsprache gelegt werden. Durch eine minimalistische Farbauswahl soll das Design unaufgeregt wirken und dem wichtigen Content (Werte, Prozentuale Trends) hervorheben. Somit soll eine angenehme User Experience gewährleistet sein.

Wichtige Trends und Entwicklungen im Fokus:

Der User soll beim Betreten der Seite über die aktuell wichtigsten Trends und Informationen der Märkte informiert werden. Hierzu kann eine Auswahl an besonders interessanten Aktien oder Kryptowährungen durch visuelles Hervorheben und Filtern einen besonderen Stellenwert erlangen.

Prozessmodellierung

Folgenden Geschäftsprozesse wurden als BPMN Diagramme modelliert. Dabei wurden Aktionen, die ein User auf unserer Seite durchführen können zur übersichtlicheren Darstellung in Unterdiagrammen aufgeteilt. Die Modellierung wurde mit Signavio realisiert.

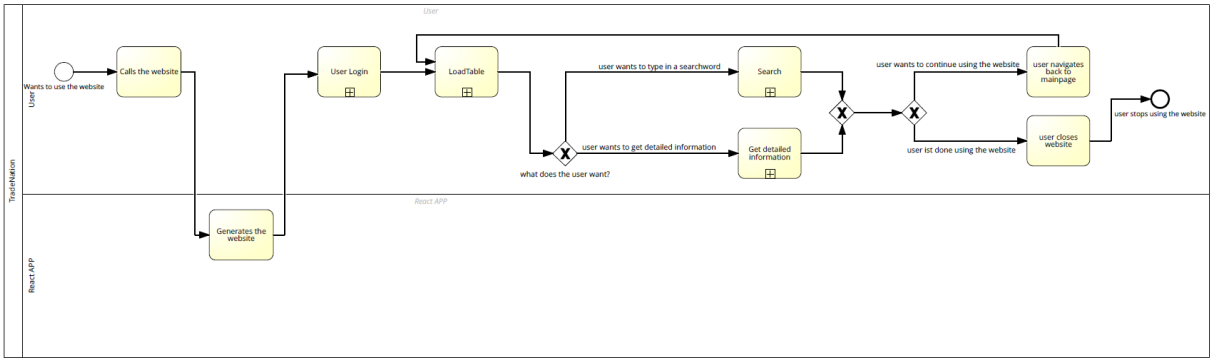


Abbildung 1 Hauptprozess BPMN

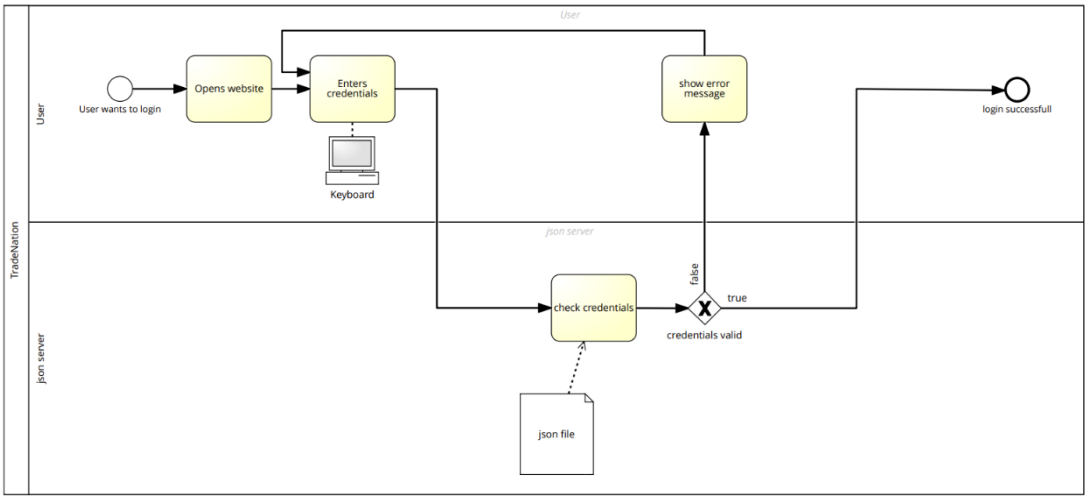


Abbildung 3 BPMN Diagramm User Login

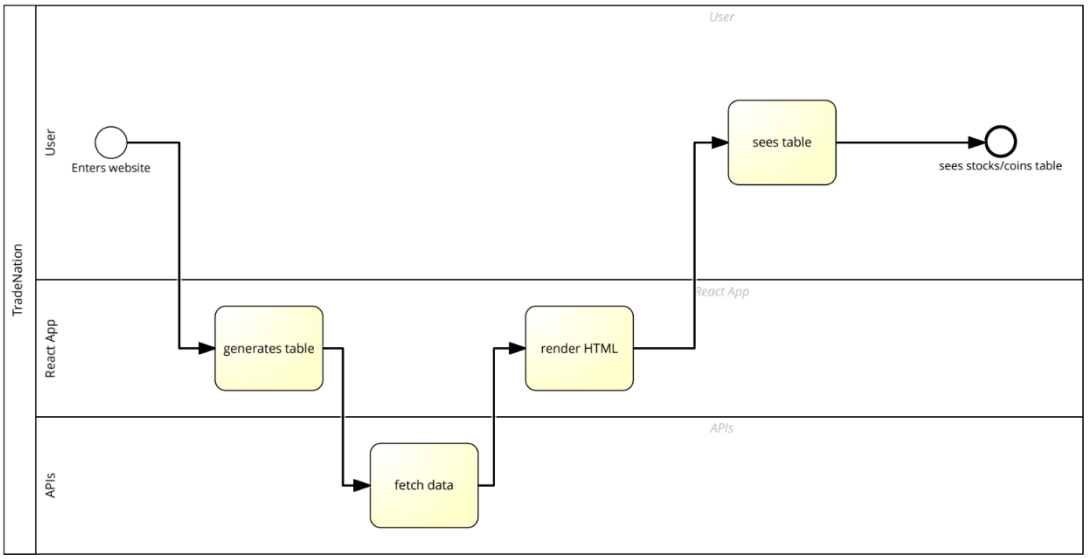


Abbildung 2 BPMN Diagramm Load Table

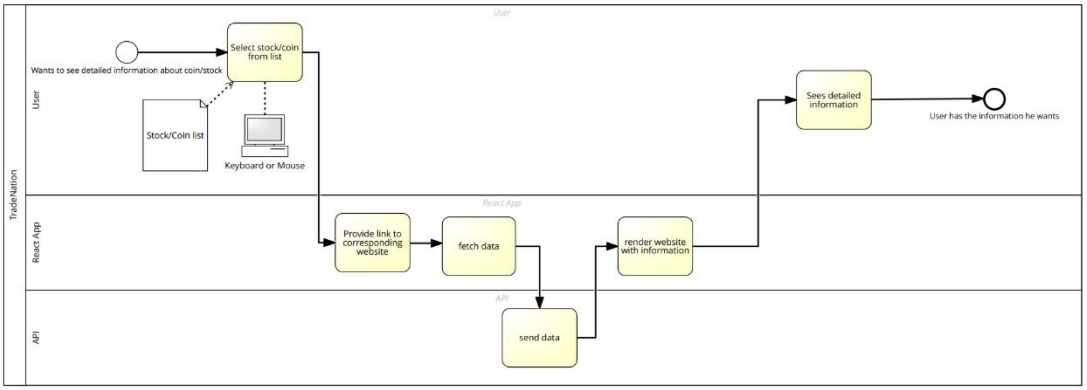


Abbildung 5 BPMN Diagramm Detailed Information

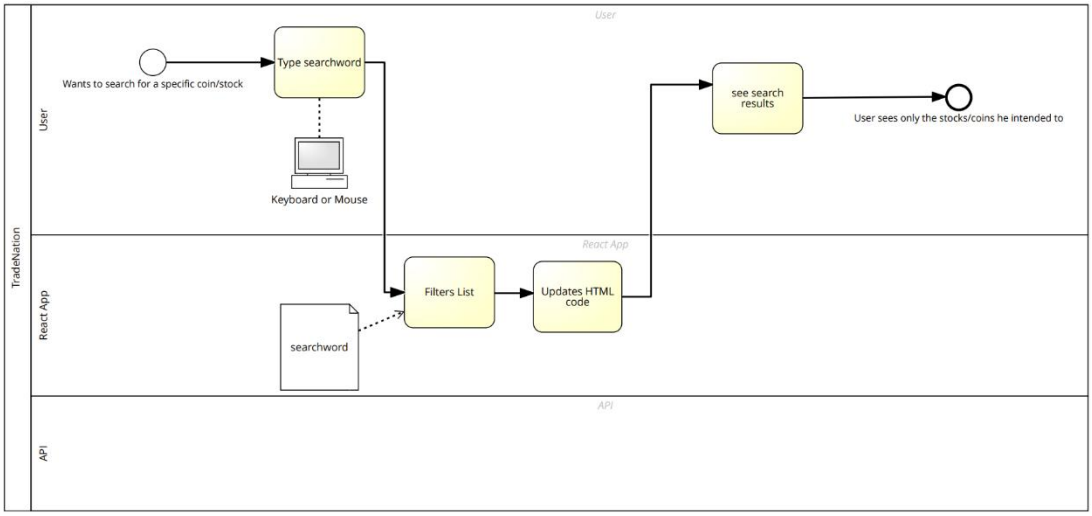


Abbildung 4 BPMN Diagramm Search

Graphical User Interface

MaterialUI

Die Gestaltung der Website wird mit MaterialUI V4 realisiert. Das UI Framework setzt auf Design-Guidelines im Stil des von Google entwickelten minimalistischen Material Design. Durch das Framework werden neue Komponenten hinzugefügt, die über Tags in den HTML Code eingebunden werden. Im Folgenden werden Components des Frameworks erläutert welche in unserem Projekt eine tragende Rolle haben.

Abbildung 6 Material UI Logo

<Typography></Typography>

Der Typography Tag findet an vielen Stellen Verwendung. Der Tag kann zum Darstellen von Textinhalten wie z.B. Überschriften verwendet werden. Mit dem `variant` Attribut kann die Darstellung gewählt werden. Das Konzept hat Ähnlichkeiten zu den Klassen in CSS. Sowohl vordefinierte Design wie z.B. „h1“, „subtitle2“ aber auch eigene erstelle variant können hier ausgewählt werden. Über das `component` Attribut kann ein HTML Tag angegeben werden, von dem der Text bzw. der zugehörige Typography Component umschlossen werden soll.

<ThemeProvider></ThemeProvider>

Der ThemeProvider Component kommt als parent Tag zum Einsatz. Alle in ihm definierten Objekte nehmen sich dem durch ihn definierten Design an. Mit dem Attribut `theme` lässt sich das entsprechende Theme wählen, hier kann auf eine mit zuvor createTheme() erstelltes Design verwiesen werden. Im createTheme() selbst können z.B. Primär-, und Sekundärfarben gewählt werden oder mit type: „dark“ eine Art Dark Mode auf das Design angewandt werden.

<Pagination></Pagination>

Der Pagination Component stellt den Nutzenden eine Schaltfläche zur Verfügung, mit der die auf mehrere Seiten aufgeteilten Einträge der Bitcoin und Stocks Liste hin und her gewechselt werden können. Mit dem `count` Attribut lassen sich die Anzahl der Seiten definieren. Wir legen den count hierbei dynamisch fest. Jede unserer Seiten zeigt zehn Einträge an. Die Gesamtanzahl alle, durch die entsprechende API erhaltenen Einträge, wird durch zehn geteilt und somit die notwendigen Seiten ermittelt.

Chart.js

Zur graphischen Aufbereitung der Kursverläufe verwenden wir Chart.js. Dabei wird eine sogenanntes Line Chart, das mehrere Datenpunkte mit Linien verbindet, mit den aus der API gelieferten Werten als Dataset c angezeigt. Die Daten werden hierbei aus der dem historicData Array gemappt und mit einem Label versehen (siehe Abbildung 8). Damit die Daten auch vom Nutzer eingeordnet werden können wird über labels die zeitlichen Eckpunkte auf der horizontalen Achse angezeigt.

Abbildung 7 Chart.js Logo

```
datasets: [
  {
    data: historicData.map((coin) => coin[1]),
    label: `Price ( Past ${days} Days ) in $`,
    borderColor: "#2CC3DB",
  },
],
```

Abbildung 8 Dataset der Kursverläufe für Coins

REACT Applikation

React ist ein JavaScript Webframework, welches die Umsetzung einer Web-Applikation ermöglicht. Dabei setzt React auf ein Komponentenbasiertes Konzept, das aus verschiedenen Komponenten eine Website zusammensetzt.

Speichern und Mappen der Daten

Die Daten werden nach dem Erhalten durch die APIs in Arrays gespeichert (siehe Abbildung 9). Dabei wird die useState() Funktion verwendet welche nur in funktionalen Components und nicht in Klassen

```
const [coins, setCoins] = useState([]);
const [stocks, setStocks] = useState([]);
```

Abbildung 9 Arrays für das Speichern der Coins & Stocks

verwendet werden kann. Als initialen Parameter der Methode wird in unserem Projekt ein leeres Array angegeben, welches dann in der korrespondierenden Methode z.B. fetchCoins() mit den Daten gefüllt werden. Zum Anzeigen der Daten auf der Website wird in der React render() Funktion die .map() Funktion auf eines der Arrays mit den gespeicherten Werten angewandt. Diese Funktion iteriert über das Array und generiert für jedes Item des Arrays HTML Code. Im Fall unserer Auflistung aller Stocks bzw. Coins wird für jedes Item eine eigene <TableRow> generiert. Um auf die gespeicherten Werte wie Name der Kryptowährung, URL des Icons oder Preis zuzugreifen und diese an die entsprechend richtige Stelle im HTML Code zu setzen kann wird folgende Syntax verwendet {item.price}.

Suchfunktionalität

Um den Nutzenden die Suche nach den richtigen Stocks und Coins zu vereinfachen, verfügt unser Projekt über eine Suchfunktion. Diese filtert im Fall der Kryptowährungen das Array mit allen

```
const handleCoinSearch = () => {
  return coins.filter(
    (coin) =>
      coin.name.toLowerCase().includes(search) ||
      coin.symbol.toLowerCase().includes(search)
  );
};
```

Abbildung 10 Implementierung der Suchfunktion für Coins

gespeicherten Coins und gibt eine auf die Suche angepasstes Array zurück. Die Funktion für das Filtern der Stocks funktioniert analog zur Funktion der Coins. Die Suche ignoriert beim Filtern Groß- und Kleinschreibung und vergleicht das durch den Nutzenden eingegebene Suchwort sowohl mit den Symbolen der Coins als auch Namen. Um also nach der Kryptowährung Ethereum zu suchen kann sowohl nach ETH als auch Ethereum gesucht werden. Das zurückgegebene gefilterte Array wird dann zur Anzeige gemappt (siehe Speichern und Mappen der Daten).

Application Programming Interface

CoinGecko

Abbildung 11 CoinGecko Logo

CoinGecko ist eine Freeware API deren Nutzen darin liegt aktuelle und historische Daten zu Kryptowährungen je nach Spezifikation den Nutzern zu Verfügung stellen kann. Hierbei wird die Abfrage über Aufgabespezifische Links ermöglicht. Diese Links starten immer mit https://api.coingecko.com/api/v3/coins/, was die verwendete API spezifiziert. Hiernach werden die Links mit den spezifischen query Parametern erweitert.

Einer der einfachsten jedoch wichtigsten Parameter ist ${id}. Hiermit können Nutzer eine einzelne Kryptowährung spezifizieren. Dieser Parameter kann alleine verwendet werden um Standardinformationen von einer einzelnen Kryptowährung zu erhalten oder in Verbindung mit dem Parameter market_chart liefert die API den Preis und market cap über den spezifizierten Zeitraum. Hierfür müssen ID, Währung und die Dauer in Tage in dem folgenden Format geliefert werden: ${id}/market_chart?vs_currency=${currency}&days=${days}.

Natürlich liefert CoinGecko nicht nur die Möglichkeit einzelne Kryptowährungen zu verarbeiten, sondern auch mehrere auf einmal. Hierfür gibt es unter anderem die Möglichkeit den Parameter markets, mit der Hilfe dieses erhält der Nutzer eine Liste aller spezifizierten Kryptowährungen mit deren Preis, market cap, Menge und weitere relevante Daten. Hierfür wird nur die Währung als Eingabewert erfordert, aber weitere Eingaben wie beispielsweise die Anzahl der Ergebnisse pro Seite, die Seitennummer oder den Zeitraum für die prozentuale Preisänderung können ergänzt werden. Die vorgegebene Struktur hierfür sieht wie Folgt aus: markets?vs_currency=USD&order=market_cap_desc&per_page=100&page=1&sparkline=false&price_change_percentage=24h

Hiermit wird diese JSON Datei geliefert:


```
[
  {
    "id": "bitcoin",
    "symbol": "btc",
    "name": "Bitcoin",
    "image": "https://assets.coingecko.com/coins/images/1/large/bitcoin.png?1547033579",
    "current_price": 46588,
    "market_cap": 886404144538,
    "market_cap_rank": 1,
    "fully_diluted_valuation": 979712783102,
    "total_volume": 35291412281,
    "high_24h": 46658,
    "low_24h": 44347,
    "price_change_24h": 85.91,
    "price_change_percentage_24h": 0.18475,
    "market_cap_change_24h": 1727846850,
    "market_cap_change_percentage_24h": 0.19531,
    "circulating_supply": 18999943,
    "total_supply": 21000000,
    "max_supply": 21000000,
    "ath": 69045,
    "ath_change_percentage": -32.42374,
    "ath_date": "2021-11-10T14:24:11.849Z",
    "atl": 67.81,
    "atl_change_percentage": 68707.78971,
    "atl_date": "2013-07-06T00:00:00.000Z",
    "roi": null,
    "last_updated": "2022-04-01T15:29:59.693Z",
    "price_change_percentage_24h_in_currency": 0.1847518653896632
```


Abbildung 12 JSON Response von CoinGecko

IEX Cloud

Abbildung 13 iexCloud Logo

IEX Cloud ist eine Freeware API welche ähnlich wie CoinGecko Finanzielle Daten liefert. Jedoch liefert IEX Cloud nicht wie CoinGecko Daten zu Kryptowährungen, sondern die erwünschten Daten des Aktienmarkts. Ähnlich wie bei CoinGecko werden die query anfragen über eine URL angefragt. Hierfür gibt es wieder ein standardisiertes Format der Anfragen. Bei IEX Cloud lautet dieses: https://cloud.iexapis.com/stable/stock/, wenn Informationen zu Aktien erwünscht sind. Des Weiteren wird für alle Abfragen das Attribut Token benötigt. Die hierfür verwendete Variable muss mit einem von IEX Cloud generierten Token befüllt werden. Dieser Token ist mit einem Account verknüpft und dessen Coin wert. Diese Coins werden für Anfragen bei IEX Cloud verwendet. Jeder Account hat eine definierte Menge an Coins welche bei Anfragen verwendet werden.

Jedoch ist es nicht möglich mit nur diesen Parametern eine erfolgreiche Anfrage an IEX Cloud zu senden, da diese nur angeben welchen API angefragt wird und von wem diese Anfrage stammt. Was genau abgefragt werden soll muss mit weiteren Parametern spezifiziert werden. Ein möglicher Parameter ist Quote welcher Aktienkurse in Echtzeit zurückgibt. Hierfür muss zunächst die Abkürzung der Aktie übergeben werden gefolgt von quote? also ${symbol}/quote?/. Wenn Preisdiagramme erstellt werden sollen, kann man hierfür den Parameter Charts verwenden. Hierfür braucht man sowohl das symbol also auch die Dauer in Tage ${symbol}/chart/${days}d?/. Zu jeder Firma gibt es aber nicht nur den Aktienpreis, sondern auch deren Firmenlogo, den Link dazu kann man über den Parameter Logo erhalten (${symbol}/logo?).

Dies waren bislang sehr spezifische Informationen zu einzelnen Firmen. Um einen generelleren Überblick über eine Firma zu erhalten kann man den Parameter Company verwenden. Mit diesem Parameter werden unter anderem der Firmenname, die Webseite der Firma, eine Beschreibung der Firma und Informationen des CEOs übermittelt. Diese Abfrage kann man wie folgt ausführen {symbol}/company?/. Ein weiterer Parameter, welcher viele Informationen über eine einzelne Aktie zurückgibt, wäre Stats. Wobei sich Company auf die Firmenspezifischen Daten bezogen hat, bezieht sich Stats nur auf deren finanziellen Stand. Hierunter befinden sich unteranderem deren aktuelle Schulden, gesamten Geldstand, wie viel Gewinn sie gemacht haben und noch vieles mehr. Dies geht über ${symbol}/stats?.

Wenn jedoch nicht nur eine Firma verarbeitet werden soll der Parameter List mit angegeben. Jedoch werden auch noch Spezifikationen der Verarbeitung benötigt, da bislang nur gesagt wird das mehrere verarbeitet werden sollen. Hierfür ist ein möglicher Parameter mostactive, welcher die zehn aktivsten Aktien zurückgibt. Um dies zu erhalten muss man market/list/mostactive?/ der Anfrage mitgeben.

Eigene API – User Login

Die Aufgabe unser eigenen API ist es den Zugang für unsere Seite zu beschränken. Hierfür haben wir eine Login Seite erstellt, welche die Nutzerdaten mit Hilfe einer JSON Datenbank validiert. Die Nutzerdaten werden mit einem post request an den Web Service übermittelt. Hierfür werden die Login URL, Nutzername und Passwort benötigt. Jedoch werden Nutzername und Passwort zunächst mit JSON.stringify zu einem String verarbeitet. Somit sieht die, sich in einem try Statement befindenden Login Anfrage, wie folgt aus.

```
try {
    const response = await axios.post(LOGIN_URL, JSON.stringify({user, pwd}), {
        headers: {
            'Content-Type': 'application/json'
        },
        withCredentials: true
    }
    );
    console.log(JSON.stringify(response?.data));
    //console.log(JSON.stringify(response));

    setAuth({user, pwd});
    setUser('');
    setPwd ('');
    setSuccess(true);
} catch (err) {
    if(!err?.response) {
        setErrMsg('No Server Response');
    }else if(err.response?.status === 400) {
        setErrMsg('Missing Username or Password');
    }
    else if(err.response?.status === 401) {
        setErrMsg('Unauthorized');
    }else {
        setErrMsg('Login Failed');
    }
    errRef.current.focus();
}
```

Abbildung 14 Code Snippet Anfrage Login API

Performancetest

Das Analysieren der Performance unsere API wird durch Postman realisiert. Postman ist eine Anwendung zum Testen von APIs. Mit folgendem Code wird unsere API aufgerufen und die benötigte

```
var time = pm.response.responseTime;
pm.test("Responsive Time: ${time}ms", function (){
    pm.response.to.have.status(200);
    var data = pm.response.json();
    pm.expect(JSON.stringify(data[0]['username'])).to.eql('"admin"');
    setTimeout(function(){}, [100]);
});
```

Abbildung 15 Code-Snippet für den Performancetest der Responsetime

Responsetime geloggt. Das ganze Testing wurde lokal durchgeführt.

Die erste Iteration benötigt 20ms was wir auf das Laden in den Cache zurückführen. Alle weiteren 99 Iterationen haben eine Responsetime von ca. 7ms und liegen somit in einem performanten und akzeptablen Bereich.

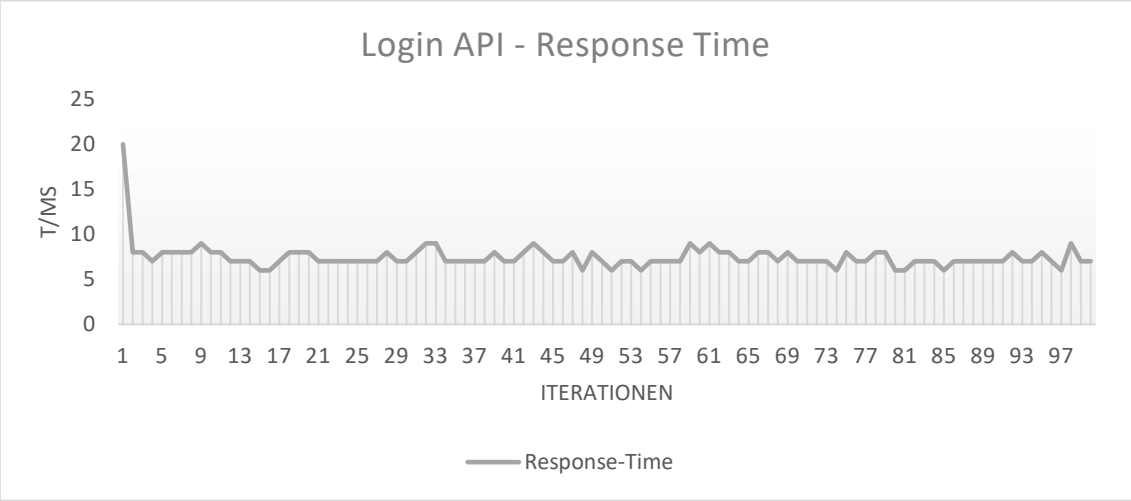


Abbildung 16 Responsetime der Login API

Abbildungsverzeichnis