

14. Mai 2021

Erin Bachmann

Luzerner Kantonalbank AG

BBZW SURSEE

Hands on OOP-/GUI-Projekt

Crypto-Desk

# Inhalt

[Inhalt 1](#_Toc75016471)

[Einleitung 2](#_Toc75016472)

[Zweck dieses Dokuments 2](#_Toc75016473)

[Einrichtung / Installation 2](#_Toc75016474)

[Projekt öffnen 2](#_Toc75016475)

[Projekt lokal einrichten 2](#_Toc75016476)

[User Story 3](#_Toc75016477)

[Mockup 6](#_Toc75016478)

[Usability Test 9](#_Toc75016479)

[Angular Architektur 11](#_Toc75016480)

[Angular Architektur im Projekt 11](#_Toc75016481)

[Klassendiagramm 14](#_Toc75016482)

[Home Component 14](#_Toc75016483)

[Detail Component 16](#_Toc75016484)

[Coin Service 18](#_Toc75016485)

# Einleitung

Dies ist die Dokumentation zu der «CryptoDesk» Website. Dieses Projekt wurde im Rahmen des Moduls 120 erstellt. Im M120 geht es um das implementieren von benutzerschnittstellen. Dies wurde hier mit einer Benutzerfreundlichen Website gemacht, welche einfach und übersichtlich gestaltet ist.

# Zweck dieses Dokuments

Dieses Dokument dient zur Orientierung und Installation des Projekts.

# Einrichtung / Installation

## Projekt öffnen

Um das Projekt lokal zu **öffnen,** gehe zu zur ZIP-Datei in der Email und öffne das Projekt in der Entwicklungsumgebung deiner Wahl. Ich persönlich empfehle folgende IDE’s:

* Visual Studio Code
* WebStorm

## Projekt lokal einrichten

Um das Projekt lokal zum Laufen zu bringen, schliesse zuerst den ersten Schritt ab. Danach kanns du im Terminal ( welches sich in dem Projektordner befindet ), **npm i** ausführen. Dies installiert alle Dependencies. Danach kannst du mit **ng s** das Projekt starten.

Hier ein Tutorial dazu: <https://www.youtube.com/watch?v=XvemczQn56U>.  
(Bis 5:40min, Danach die oben erwähnten Schritte durchführen.)

# User Story

**Priorität 1**

**Story 1: Crpytocurrency-Suche**

Als Benutzer möchte ich durch einen Suche-Button die Möglichkeit haben, nach etlichen Cryptocurrencies zu suchen. Nach der Eingabe wird ein Fenster oder eine Seite mit Informationen über die Cryptocurreny auftauchen. Dadurch kann ich selber entscheiden nach welcher Cryptocurrency ich suchen will und kann mich dadurch informieren.

Abnahmekriterien:

* Ein Button zur Suche nach einer Cryptocurrency
* Eingabefeld für Cryptocurrency
* Fenster / Seite mit Informationen über die Cryptocurreny

Status: **In Arbeit …**

**Story 2: Cryptocurrerny-Übersicht**

Als Benutzer möchte ich auf der Startseite der Website eine Crpycurrency Übersicht haben. In dieser werden etliche Currencies mit einigen Informationen in einer Tablle angezeigt. Dadurch bekommt der User schnell einen Überblick über die wichtigsten Currencies.

Abnahmekriterien:

* Tablelle mit Informationen über einige Currencies
* Mindestens 10 Currencies werden dargestellt.

Status: **In Arbeit …**

**Priorität 2**

**Story 3: Graph zu Cryptocurreny**

Als Benutzer möchte ich einen Graph zu einer Cryptocurreny angezeigt bekommen. In diesem Graph erkenne ich den Preis über eine bestimme Zeitperiode. Dadurch kannn ich die Preisentwicklung der Cryptocurreny besser bestimmen.

Abnahmekriterien:

* Der Benutzer sieht einen Graph zu der gesuchten Currency
* Der Graph zeigt dem User die Preisentwicklung über eine gewisse Zeit.

Status: **In Arbeit …**

**Story 4: Cryptocurrerny-Umrechner**

Als Benutzer möchste ich die Möglichkeit haben eine gewünschte Cryptocurreny in USD oder CHF umrechnen zu können. Dadurch kann ich mir als Benutzer besser vorstellen wie viel die Cryptocurreny wert ist.

Abnahmekriterien:

* Der Benutzer sieht ein Eingabefeld für die Eingabe der gewünschten Cryptocurrency zum umrechnen.
* Der Benutzer bekommt nach dem klicken eines Buttons den Wert der Cryptocurreny angezeigt.

Status: **In Arbeit …**

**Story 5: Eingabevorschläge bei Cryptocurreny-Suche**

Als Benutzer möchte ich während der Suche nach einer Cryptocurrency lauffend Vorschläge bekommen. Dadurch wird die Eingabe wesentlich erleichtert und die Möglichkeit dass der User etwas falsches eingibt sinkt.

Abnahmekriterien:

* Der Benutzer sieht während der Eingbe im Eingabefeld lauffend Vorschläge.

Status: **In Arbeit …**

**Priorität 3**

**Story 5: Trending Currencies**

Als Benutzer möchte ich eine Übersicht mit den mommentan beliebtesten Crptyocurrencies haben. Dadurch kann ich schauen in was andere Personen investieren und kann daraus evt. schliessen in was ich selber investieren möchte.

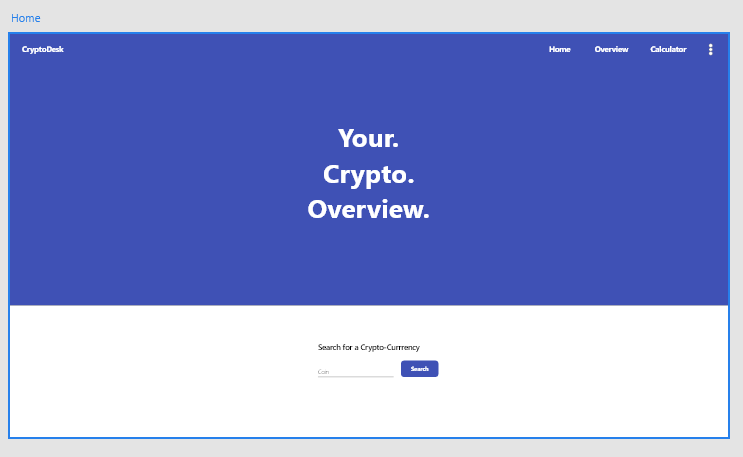
Abnahmekriterien:

* Der Benutzer sieht eine Tabelle mit den 10 beliebstesten Cryptocurrencies.
* In der Tablelle werden informationen über die Cryptocurreny angezeigt.
* Die Beliebtheit wird nach dem Kurs berechnet.

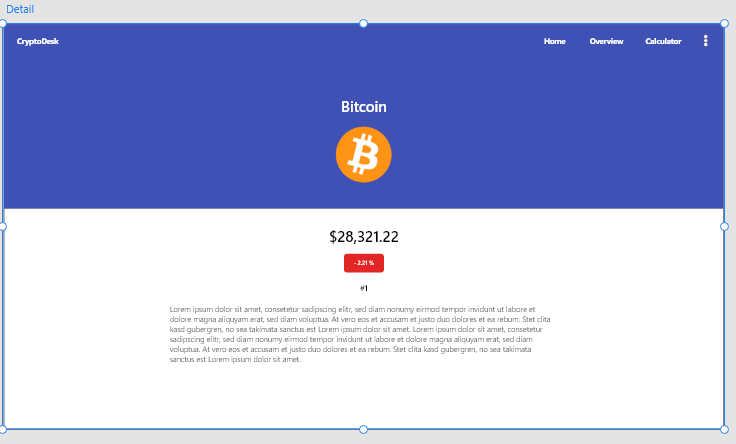
Status: **In Arbeit …**

# Mockup

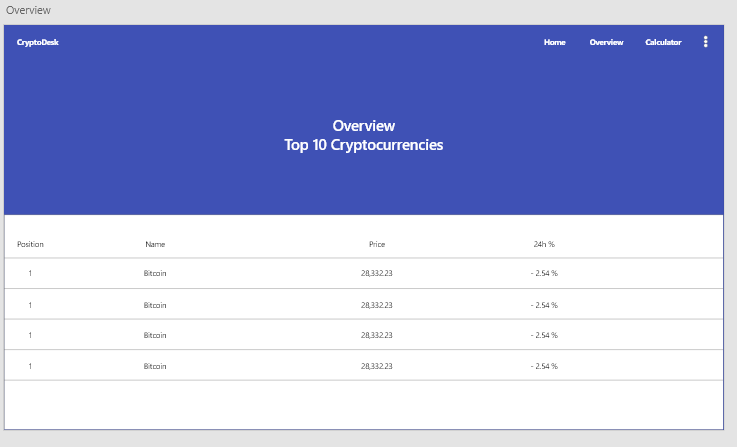
Das Mockup wurde mit dem Programm «Adobe XD» erstellt. Das ganze kann über diesen Link: <https://xd.adobe.com/view/9167a1f4-1551-4e4d-b92c-4dc45ff0e5fe-c434/?fullscreen&hints=off> aufgerufen werden.



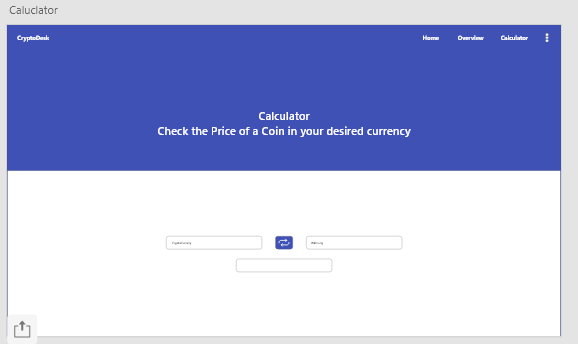
Auf diesem Bild sieht man die Landing page der Website. In der Rechten oberen Ecke sind die Navigationspunkte zu sehen. Die 3 kleinen Punkte sind dazu da, das ganze bei kleineren Bildschirmen Responsive zu machen.  
  
Auf der unteren Seite kann man nach einer Cryptocurrency schauen. Bei der eingabe werden lauffen Vorschläge aufzauchen. Nachdem auf den «Suchen» Button geklick wurde, wird man auf die nächste Seite weitergeleitet.



Auf dieser Seite werden Informationen über eine Währung angezeigt. Oben ist der Name und das Logo und unten der Preis, die Veränderung der Währung in den Letzen 24h, die Platzierung im Ranking und die Beschreibung des Coins.



Beim Overview warden die 10 höchst gerankten Coins angezeigt. In der Tabelle wird die Position, der Preis und die Veränderung in den letzten 24h angezeigt.



Bei dieser Seite wird es möglich sein eine Cryptowährung in eine gewünschte Währung umzurechnen. Im ersten Feld wird die Cryptowährung und im Zweiten die Währung angegeben. Das Resultat wird im unteren Feld ausgegeben.

# Usability Test

Mit diesem Test soll untersucht werden, wie gut die Usability der Webappliaktion «CryptoDesk» für die gegebenen Testszenarien ist.

Es geht darum in der App zu navigieren. Der Benutzer sucht nach einer Cryptowährung und bekommt eine klare Übersicht angezeigt. Danach möchte der Benutzer eine Übersicht sehen. Diese wird er durch eine Navigation angezeigt bekommen.

Ablauf:

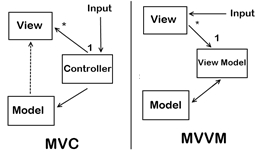
1. Website Aufrufen (localhost)
2. Navigieren zu home (automatisch)
3. Im Textfeld «Bitcoin» eingeben.
4. Auf «Suchen» klicken.
5. Es wird die Detailseite angezeigt.
6. In der navbar auf «home» klicken.
7. Es wird die Seite «home» angezeigt.

Frage an die Testperson

* Wie haben Sie die Bedienung des Steuerkalkulators erfahren?

|  |  |
| --- | --- |
| Moderator | Erin Bachmann |
| Testperson | Johannes Jund |
| Beobachter | Moritz Wicki |
| Datum | 04.06.2021 |
| Uhrzeit | 16:00 |
| Beobachtungsprotokoll | Der Benutzer ( Johannes Jund ) konnte sich gut durch die Website navigieren. Dank der einfachen und Responsiven Navbar konnte er zu jedem Zeitpunkt und auf mehreren Geräten zu den gewünschte Seiten navigieren. Er hat das Eingabe Feld für die Cryptocurreny schnell gefunden und konnte durch das Dropdownmenu ( Eingabevorschläge ) schnell und Einfach zur geschwünschten Währung wechseln. |

# Angular Architektur

Angular ist im Grunde auf der MVC Architektur aufgebaut. Jedoch ähnelt die Architektur eher der sogenannten MVVM-Architektur. MVVM heisst si viel wie «Model-View-ViewModel». Das MVVM-Modell wird genutzt um «two-way data

binding» innerhalb einer Applikation zu ermöglichen (Dies geschieht in Angular durch das sogenannte «ngModel»). Durch das «two-way binding» kann der View automatisch auf eine Änderung innerhalb des View-Models eingehen und diese anzeigen.

Hier ein Beispiel zum NgModul:

<https://drive.google.com/file/d/1V0OPZJ_nulKrky-XF7wLzLFdHogQNIOn/view?usp=sharing> (am besten Video herunterladen)

Des Weiteren ist es durch die Vereinfachte MVC-Grundstruktur nicht vonnöten Getter uns Setter zu deklarieren. In diesem Projekt habe ich mich aber 1- bis 2-mal dazu entschieden Getter uns Setter zu benutzen. Dies vor allem, um die Übersichtlichkeit im Code ein bisschen zu verbessern. Im Normalfall werden diese aber nicht benutzt.

Angular organisiert den Code durch sogenannte Buckets, bzw. Modules. Durch die daraus resultierenden wiedergebrauchbaren features und chuncks, wird die Applikation um einiges Übersichtlicher.

## Angular Architektur im Projekt

Bei diesem Projekt habe ich diese Architektur mit «Interfaces» umgesetzt. Interfaces kann man auch als Objekte sehen, welche beim gebrauch bestimmte eigenschaften vorgeben. Hier das Beispiel am «CoinPrice» interface / CoinService und DetailComponent.

Hier wird ein Objekt vom Typ «CoinPrice» Zurückgegeben.

 async getCoinMarketDataById(id: string | null):Promise<CoinPrice>{

    return await ((this.http.get<CoinPrice>("https://api.coinpaprika.com/v1/tickers/" + id).toPromise()));

Der Typ «CoinPrice» hat dabei folgende Eigenschaften.

export interface CoinPrice {

  id: string;

  name: string;

  circulating\_supply: number,

  total\_supply: number;

  max\_supply: number;

  quotes: quotes;

}

export interface quotes {

  USD:USD;

}

export interface USD {

    price: number,

    volume\_24h: number;

    volume\_24h\_change\_24h: number;

    market\_cap: number,

    market\_cap\_change\_24h: number,

    percent\_change\_15m: number,

    percent\_change\_30m: number,

    percent\_change\_1h: number,

    percent\_change\_6h: number,

    percent\_change\_12h: number,

    percent\_change\_24h: number,

    percent\_change\_7d:number,

    percent\_change\_30d: number,

    percent\_change\_1y: number

}

Wenn wir das ganze nun loggen, sehen wir die Daten, den richtigen Eigenschaften zugeordnet.

Nun wird die oben beschriebene Funktion «getCoinMarketDataById()» vom DetailComponent aufgerufen.

async getCoinMarketDataById(){

    await this.coinService.getCoinMarketDataById(this.coinId).then((data =>{

      this.coinMarketPrice = data;;

    }));

  }

Danach könne die Daten im View, durch das ansteuern der Objektes «coinMarketPrice» angezeigt werden.

<div class="content-container-2">

      <h1>$ {{removeDecimal(coinMarketPrice.quotes.USD.price)}}</h1>

      <button mat-raised-button color="warn">{{coinMarketPrice.quotes.USD.market\_cap\_change\_24h}}%</button>

    </div>

Hier ist ein kurzes Video dazu: <https://www.youtube.com/watch?v=I-lawhoR5os>   
Es beschreibt das DataBinding/ MVC-/MVNM-Model in Angular

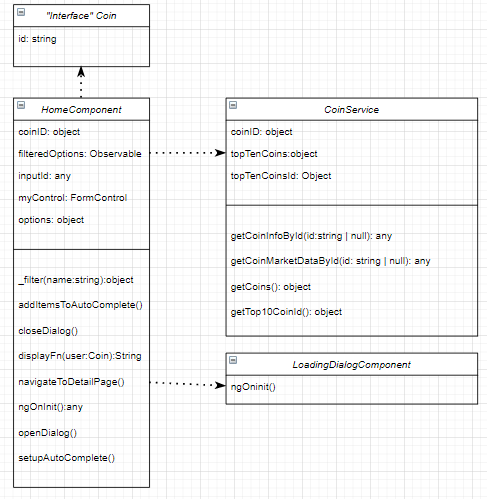
# Klassendiagramm

Das gesammte Klassendiagramm kann über diesen Link: <https://drive.google.com/file/d/1QeB3HvmfPHP5DqZMaPy_So9JjSZQW6xn/view?usp=sharing> abegrufen werden. Das ganze ist sehr gross, wesswegen es nicht in diese Dokumentation gepasst hätte.

Ich werden aber nun auch noch die 2 Wichtigsten Seiten (Home-/Detailpage), und die darausfolgenden Beziehungen, aufzeigen.

Zusätzlich werden die Beziehungen von dem CoinService noch aufgezeigt.

## Home Component



Hier sieht man eindeutig die 3 Beziehungen, welche vom HomeComponent ausgehen. (Die Beziehungen welche vom «CoinService» ausgehen werden separat noch näher beleuchtet)

Die Beziehung HomeComponent zu CoinService ist eine Lose-Beziehung vom Typ2 (use)

async ngOnInit() {

    this.openDialog();

    await this.coinService.getCoins().then((data =>{

      this.coinID = data;

    }

Die Beziehung HomeComponent zu LoadingDialogComponent ist eine Lose-Beziehung vom Typ2 (use).

openDialog(){

    this.dialog.open(LoadingDialogComponent);

  }

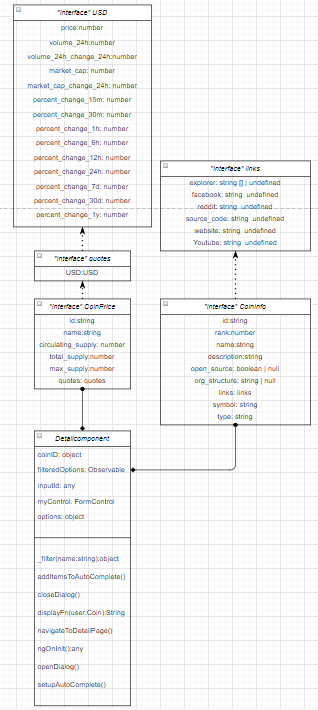
));.

Die Beziehung HomeComponent zum Interface «Coin» ist eine Lose-Beziehung vom Typ1 (instanziiert).

options: Coin[] = [];

## Detail Component

Hier sieht man eindeutig die 5 Beziehungen, von welchen 2 vom DetailComponent ausgehen.



Die Beziehung DetailComponent zum interface «CoinInfo» ist eine Kompositions-Beziehung. CoinInfo ist Teil von DetailComponent und davon abhängig.

public \_coinInfo: CoinInfo;

Die Beziehung DetailComponent zum interface «CoinPrice» ist eine Kompositions-Beziehung. CoinPrice ist Teil von DetailComponent und davon abhängig.

public \_coinMarketPrice: CoinPrice;

Die Beziehung vom interface «CoinInfo» zum interface «links» ist eine Lose-Beziehung vom Typ1 (instanziiert).

export interface CoinInfo {

  id: string;

  rank: number,

  name: string,

  description: string,

  open\_source: boolean | null,

  org\_structure: string | null

  links: links;

  symbol: string;

  type: string;

}

Die Beziehung vom interface «CoinPrice» zum interface «quotes» ist eine Lose-Beziehung vom Typ1 (instanziiert).

export interface CoinPrice {

  id: string;

  name: string;

  circulating\_supply: number,

  total\_supply: number;

  max\_supply: number;

  quotes: quotes;

}

Die Beziehung vom interface «quotes» zum interface «USD» ist eine Lose-Beziehung vom Typ1 (instanziiert).

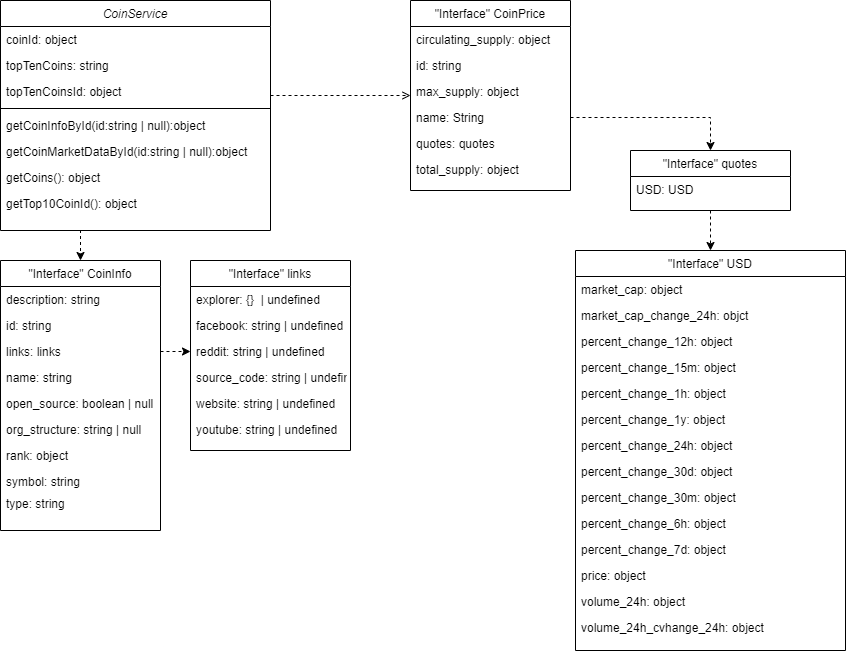
export interface quotes {

  USD:USD;

}

## Coin Service

Hier sieht man eindeutig die 5 Beziehungen, von welchen 2 vom CoinService ausgehen.



Die Beziehung CoinService zum interface «CoinPrice» ist eine ist eine Lose-Beziehung vom Typ1 (instanziiert).

  async getCoinMarketDataById(id: string | null):Promise<CoinPrice>{

    return await ((this.http.get<CoinPrice>("https://api.coinpaprika.com/v1/tickers/" + id).toPromise()));

  }

Beziehung CoinService zum interface «CoinInfo» ist eine ist eine Lose-Beziehung vom Typ1 (instanziiert).

 async getCoinInfoById(id: string | null):Promise<CoinInfo> {

    return await ((this.http.get<CoinInfo>("https://api.coinpaprika.com/v1/coins/" + id).toPromise()));

  }

Beziehung interface «CoinPrice» zum interface «quotes» ist eine ist eine Lose-Beziehung vom Typ1 (instanziiert).

export interface CoinPrice {

  id: string;

  name: string;

  circulating\_supply: number,

  total\_supply: number;

  max\_supply: number;

  quotes: quotes;

}

Beziehung interface «quotes» zum interface «USD» ist eine ist eine Lose-Beziehung vom Typ1 (instanziiert).

export interface quotes {

  USD:USD;

}

Beziehung interface «CoinInfo» zum interface «links» ist eine ist eine Lose-Beziehung vom Typ1 (instanziiert).

export interface CoinInfo {

  id: string;

  rank: number,

  name: string,

  description: string,

  open\_source: boolean | null,

  org\_structure: string | null

  links: links;

  symbol: string;

  type: string;

}