



GPT Chatlog

▼ Clicking twice on favorites

```
function renderFavorites() {  
  const favorites = JSON.parse(localStorage.getItem("favorites") || "[]");  
  const list = document.getElementById("favoritesList");  
  list.innerHTML = "";  
  
  favorites.forEach(ticker => {  
    const li = document.createElement("li");  
    li.textContent = ticker;  
    list.appendChild(li);  
  });  
}
```

Chat GPT said:

✅ Fix: Force capture of current input

Instead of reading `.value` on button click, wrap it in a slight delay using `requestAnimationFrame()`:

Fix:

```
addFavoriteButton.addEventListener("click", () => {  
  
  requestAnimationFrame(() => {  
  
    const ticker = tickerInput.value.trim().toUpperCase();  
  
    if (!ticker) return;
```

```

const favorites = JSON.parse(localStorage.getItem("favorites") || "[]");

if (!favorites.includes(ticker)) {

  favorites.push(ticker);

  localStorage.setItem("favorites", JSON.stringify(favorites));

  console.log(`Added ${ticker}`);

  renderFavorites();

} else {

  console.log(`${ticker} is already a favorite`);

}

});

});

```

Why this works:

`requestAnimationFrame()` waits until the DOM updates (including input blur/value changes)

Ensures `.value` reflects the latest input state

▼ fetchIntraDayData()

```

// intradayAPI.js
'use strict';

const apiKey = "OUHQGMU8N0KTHTEB";

```

```

export async function fetchIntradayData(symbol = "IBM", interval = "5min") {
  const TimeSeriesIntradayAPI = `https://www.alphavantage.co/query?function=TIME_SERIES_INTRADAY&symbol=${symbol}&interval=${interval}&apikey=YOUR_API_KEY`;

  try {
    const response = await fetch(TimeSeriesIntradayAPI);
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const data = await response.json();

    const key = `Time Series (${interval})`;
    if (!data[key]) {
      console.warn(`Geen geldige data gevonden voor ${symbol} met interval ${interval}`);
      return { [key]: {} };
    }

    console.log(`volledige data voor ${symbol}:`, data);
    return data;
  } catch (error) {
    console.error("Fout bij ophalen van data:", error);
    const fallbackKey = `Time Series (${interval})`;
    return { [fallbackKey]: {} };
  }
}

```

Waarom gebruikt mijn export async function fetchIntradayData()

geen promises?

ChatGPT said:

Goede vraag — laat me dat heel scherp uitleggen:

Je **gebruikt wél een Promise**, maar je gebruikt hem op een **moderne** manier met `async/await` in plaats van `then().catch()`.

In detail:

- Elke functie die je schrijft als `async function` **geeft automatisch een Promise terug**.

(Of jij dat nu letterlijk schrijft of niet.)

- Intern maakt `async function` de Promise voor jou aan.
- Dus deze:

```
js
Copy
export async function fetchIntradayData() { ... }
```

is eigenlijk hetzelfde als:

```
js
Copy
export function fetchIntradayData() {
  return new Promise((resolve, reject) => {
    // doe hier fetch-logica
  });
}
```

Maar `async/await` maakt het **veel leesbaarder** en **makkelijker** te schrijven.

Wat gebeurt er onder water?

Als jij schrijft:

```
js
Copy
```

```
const intradayData = await fetchIntradayData(ticker, interval);
```

dan:

- Wacht JavaScript **asynchroon** op de Promise die `fetchIntradayData()` oplevert.
- Als die Promise resolve't (success), krijg je de `data`.
- Als die Promise reject't (error), spring je automatisch naar de `catch` (als je een `try/catch` rond je `await` hebt gezet).

Samenvatting

Aspect	Uitleg
<code>async function</code>	Maakt automatisch een Promise aan.
<code>await</code>	Pauzeert de uitvoering tot de Promise is afgehandeld.
Geen <code>then().catch()</code> nodig	Beter leesbare code, vooral bij meerdere async calls.

Dus:

✅ Je gebruikt Promises.

✅ Alleen veel mooier en moderner via `async/await`.

Laat me weten of je ook eens een voorbeeld wil zien **met** `.then()` erbij, zodat je het verschil nog concreter ziet! 🚀

(Goed dat je deze vraag stelde trouwens — echt een teken dat je het dieper begint te begrijpen.)

You said:

oke, waarom gebruiken we if statements en geen `.then()`?

ChatGPT said:

Top vraag — en ik leg het je meteen scherp uit.

1. `.then()` en `.catch()`

Die worden gebruikt **om Promises af te handelen**:

- `.then()` → als de Promise **goed gaat** (resolved)
- `.catch()` → als de Promise **fout gaat** (rejected)

Voorbeeld met `.then()` / `.catch()` :

```
js
Copy
fetchIntradayData("AAPL", "5min")
  .then(data => {
    console.log(data);
  })
  .catch(error => {
    console.error(error);
  });
```

2. `if` statements

Die gebruik je **binnen** je eigen code **om gewone logica te controleren**:

- Bijvoorbeeld: "Bestaat een bepaalde key in de API-response?"
- Of: "Is een waarde geldig?"

Voorbeeld van onze `if` :

```
js
Copy
if (!data[key]) {
  console.warn("Geen geldige data");
}
```

Dat is **niet** Promise-afhandeling. Dat is gewoon een **normale controle**:

"Staat de verwachte property wel in het object?"

 **Kort samengevat:**

<code>.then()</code> / <code>.catch()</code>	<code>if</code> statement
Voor Promise handling	Voor logische checks binnen data
Controleert: lukt de async call?	Controleert: klopt de data-inhoud?

En waarom kiezen wij voor `async/await` + `if` ?

- `async/await` maakt de code veel vlotter en leesbaarder dan `.then().catch()`.
- `if` zorgt dat we binnen die data die we krijgen **extra validatie** kunnen doen.

Wil je dat ik heel kort een vergelijking maak: dezelfde `fetchIntradayData` één keer met `.then()` en één keer met `async/await`, zodat je het verschil écht voelt? 🚀
(duurt 1 minuut)

You said:

oke

ChatGPT said:

Nice, hier komt een heel duidelijk vergelijking:

1. Met `async/await` (zoals wij nu werken):

```
js
Copy
async function fetchIntradayData(symbol, interval) {
  try {
    const response = await fetch(`https://...`);
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }

    const data = await response.json();

    if (!data[`Time Series (${interval})`]) {
      throw new Error("Geen geldige data gevonden.");
    }
  }
}
```

```

    }

    return data;
  } catch (error) {
    console.error("Fout bij ophalen:", error);
    return {}; // fallback
  }
}

```

Kort:

- `await` stopt de functie even tot de fetch klaar is.
- `try/catch` vangt alle fouten samen af.
- Binnen de `try` gebruik je `if` voor normale logica (bv. checken of de juiste data er is).

2. Dezelfde logica maar dan met `.then()` en `.catch()` :

```

js
Copy
function fetchIntradayData(symbol, interval) {
  return fetch(`https://...`)
    .then(response => {
      if (!response.ok) {
        throw new Error(`HTTP error! status: ${response.status}`);
      }
      return response.json();
    })
    .then(data => {
      if (!data[`Time Series (${interval})`]) {
        throw new Error("Geen geldige data gevonden.");
      }
      return data;
    })
}

```



```

    .catch(error => {
      console.error("Fout bij ophalen:", error);
      return {}; // fallback
    });
  }

```


Hier zie je:

- `fetch()` → `.then(response => ...)`
- `.then(data => ...)`
- `.catch(error => ...)`

Je moet de ketting (`then.then.catch`) blijven schrijven, wat snel **rommelig** wordt als je meer logica nodig hebt.

Wat je moet onthouden:

- `async/await` + `try/catch` = veel overzichtelijker en moderner.
- `.then().catch()` is ouder en moeilijker te lezen bij meer stappen.

Als je echt de best practices wil volgen vandaag, dan blijf je gewoon bij wat je nu doet: `async/await` . 

▼ fetching data and transforming this into a table

fetching data and transforming this into a table:

how do i put the rawdata that is fetched from this function into a table format?

```

"use strict";

import { StockRepository } from "../api/stockRepository";
import { validateParams } from "../utils/validateParams.js";
import { transformIntradayData } from "../utils/transformIntradayData.js";

```

```

// import { renderChart } from "../utils/renderChart.js";
import { renderChartByType } from "../main/main.js";

export async function intradayData(ticker = "AAPL", interval = "15m") {
  try {
    validateParams(ticker, interval);

    //fetch rawdata from API
    const stockRepo = new StockRepository();
    const rawData = await stockRepo.fetchIntradayData(ticker, interval);

    //transform rawdata to readable data for renderchart
    let chartData = transformIntradayData(rawData);

    renderChartByType(chartData);
  } catch (error) {
    console.error("failed to load intraday data:", error);
  }
}

```

I found this online:

```

const todoAPI = 'https://jsonplaceholder.typicode.com/todos';
const tableBody = document.querySelector('tbody');
fetch(todoAPI)
  .then((response) => response.json())
  .then((todos) => {
    tableBody.innerHTML = todos.reduce((itemsToRenderInTable, tod
      return (
        itemsToRenderInTable +
        <tr><td>${index + 1}</td><td>${todo.title}</td></tr>
      );
    }, "");
  });

```

```
})  
.catch(console.error);
```

ChatGPT said:

Perfect! You're almost there. To render your **intraday stock data into a table**, you just need to do what that `todoAPI` example does — loop over your parsed data and inject it into a `<tbody>`.

✅ Step-by-Step Guide

1. Create your table in HTML

Place this below `#mainDiv` :

```
html  
CopyEdit  
<div id="tableSection" class="container py-4">  
  <h2>📄 Intraday Data Table</h2>  
  <div class="table-responsive">  
    <table class="table table-striped table-bordered">  
      <thead>  
        <tr>  
          <th>Time</th>  
          <th>Open</th>  
          <th>High</th>  
          <th>Low</th>  
          <th>Close</th>  
          <th>Volume</th>  
        </tr>  
      </thead>  
      <tbody id="dataTableBody"></tbody>  
    </table>  
  </div>  
</div>
```

▼ Debuggen Invalid date

```
import { StockRepository } from "../api/stockRepository";
// import { renderChart } from "../utils/renderChart";
import { transformDateRangeData } from "../utils/transformDateRangeData";
import { renderChartByType } from "../main/main.js";
import { populateIntradayTable } from "../utils/populateIntradayTable.js";

export async function dateRangeData(ticker, offsetDays) {
  try {
    const end = new Date();
    const start = new Date();
    start.setDate(end.getDate() - offsetDays);

    const formatDate = (date) => date.toISOString().split("T")[0];

    const startDate = formatDate(start);
    const endDate = formatDate(end);

    const stockRepo = new StockRepository();
    const rawData = await stockRepo.fetchDateRangeData(
      ticker,
      startDate,
      endDate
    );
    let chartData = transformDateRangeData(rawData);
    populateIntradayTable(rawData);
    renderChartByType(chartData);
  } catch (error) {
```

```
console.error("failed to load date range data:", error);  
}  
}
```

ChatGPT said:

Yes, goed opgemerkt — je bent in je `dateRangeData()` functie inderdaad al bezig met **formatDate**, maar let op: dat is iets heel anders dan wat je later in je tabel wil weergeven.

🤔 Wat gebeurt er hier precies?

```
js  
CopyEdit  
const formatDate = (date) => date.toISOString().split("T")[0];
```

➡ Deze `formatDate()` functie is bedoeld om een **datum om te zetten naar een string in YYYY-MM-DD formaat**, zoals vereist door jouw **API call**.

Bijv.:

```
js  
CopyEdit  
2025-05-15T09:30:00.000Z → "2025-05-15"
```

✅ Dit is dus enkel voor **de request naar je backend**, niet voor het mooi weergeven in de tabel!

📊 Wat je in de tabel wil tonen is iets menselijks, zoals:

- 15 mei 2025, 09:30
- of 15 May 09:30

Daarom gebruik je bij het renderen van je table de **andere** `formatDate()` functie:

```
js
CopyEdit
function formatDate(dateString) {
  return new Date(dateString.replace(" ", "T")).toLocaleString("en-GB", {
    day: "2-digit",
    month: "short",
    hour: "2-digit",
    minute: "2-digit",
  });
}
```

✓ Conclusie

Je hebt dus:

Context	Doel	Formatter
API-request (bovenin)	Maak <code>startDate</code> en <code>endDate</code> zoals <code>"2025-05-15"</code>	<code>toISOString().split("T")[0]</code>
Tabel tonen	Maak <code>"2025-05-15 09:30:00"</code> mooi zichtbaar	<code>toLocaleString(...)</code>

Ze hebben **allebei het label** `formatDate`, maar doen totaal iets anders.

(Je zou één van de twee gerust `formatDateForDisplay()` kunnen noemen voor duidelijkheid.)

Laat maar weten als je wil dat ik ze opschoon of hernoem! ✓

You said:

DateRange gives back invalid date

ChatGPT said:

Als je `"Invalid Date"` krijgt bij je **DateRange-data**, dan betekent dat meestal:

🔥 De Datetime-string die je binnenkrijgt zit niet in een geldig formaat voor `new Date(...)`

✅ Snel debuggen: check je data

Voeg dit toe net boven `populateIntradayTable(rawData);`:

```
js
CopyEdit
console.log("Eerste datum:", rawData[0]?.Datetime);
```

Je zal iets zien zoals:

```
arduino
CopyEdit
"2025-05-15 09:30:00"
```

⚠️ Let op:

Dit formaat (`"2025-05-15 09:30:00"`) bevat een **spatie** tussen datum en tijd, maar `new Date("2025-05-15 09:30:00")` is **niet altijd geldig in alle browsers**.

✅ Fix: maak het ISO-compatible met `.replace(" ", "T")`

In je `formatDate` -functie (die je gebruikt bij het renderen van de tabel), gebruik je dit:

```
js
CopyEdit
function formatDate(dateString) {
  const isoString = dateString.replace(" ", "T");
  return new Date(isoString).toLocaleString("en-GB", {
    day: "2-digit",
    month: "short",
```

```
    hour: "2-digit",  
    minute: "2-digit"  
  });  
}
```

 Hiermee wordt `"2025-05-15 09:30:00"` → `"2025-05-15T09:30:00"` → **geldig ISO-formaat**

✅ Bonus check: spelling

Zorg dat je gebruikt:

```
js  
CopyEdit  
item.Datetime
```

en niet per ongeluk `item.dateTime`, `item.datetime`, `item.Datetim`, enz.

Samengevat

1. Log je `rawData[0].Datetime` om te checken
2. Gebruik `.replace(" ", "T")` voor compatibiliteit
3. Zorg dat je `formatDate(...)` **correct gespeld** is én gebruikt wordt

Laat me je `rawData[0]` even zien als je wil dat ik het exact voor jouw geval oplos



You said:

intradayData.js:23 failed to load intraday data: ReferenceError: formatDate is not defined

at populateIntradayTable.js:12:7

at Array.reduce (<anonymous>)

at populateIntradayTable (populateIntradayTable.js:9:30)

at intradayData (intradayData.js:20:5)

ChatGPT said:

Ah yes — duidelijke foutmelding:

```
vbnet
CopyEdit
ReferenceError: formatDate is not defined
```

✓ Wat betekent dit?

Je gebruikt `formatDate()` in je bestand `populateIntradayTable.js`, maar je **hebt die functie daar niet gedefinieerd of geïmporteerd**.

🔧 **Oplossing: voeg `formatDate` toe of importeer het**

Optie 1: definieer `formatDate` direct bovenin `populateIntradayTable.js`

```
js
CopyEdit
function formatDate(dateString) {
  const safe = dateString.replace(" ", "T");
  return new Date(safe).toLocaleString("en-GB", {
    day: "2-digit",
    month: "short",
    hour: "2-digit",
    minute: "2-digit"
  });
}
```

➡ Dit is het makkelijkst als je alleen daar met datum werkt


Optie 2: zet `formatDate` in apart bestand, en importeer het

Maak bv. een bestand: `utils/formatDate.js` :

```
js
CopyEdit
export function formatDate(dateString) {
  const safe = dateString.replace(" ", "T");
  return new Date(safe).toLocaleString("en-GB", {
    day: "2-digit",
    month: "short",
    hour: "2-digit",
    minute: "2-digit"
  });
}
```

En importeer dan bovenin je `populateIntradayTable.js` :

```
js
CopyEdit
import { formatDate } from "../formatDate.js";
```

 **Tip:** Dit is een mooie kans om herbruikbare functies zoals `formatDate` apart te zetten.

Zo blijft je code proper, vooral als je ze ook elders wil gebruiken.

Laat maar weten welke manier je kiest — dan pas ik het mee aan. 

You said:

```
import { StockRepository } from "../api/stockRepository";
// import { renderChart } from "../utils/renderChart";
import { transformDateRangeData } from "../utils/transformDateRangeData";
import { renderChartByType } from "../main/main.js";
import { populateIntradayTable } from "../utils/populateIntradayTable.js";
```

```
export async function dateRangeData(ticker, offsetDays) {
  try {
    const end = new Date();
    const start = new Date();
    start.setDate(end.getDate() - offsetDays);

    const formatDate = (date) => date.toISOString().split("T")[0];

    const startDate = formatDate(start);
    const endDate = formatDate(end);

    const stockRepo = new StockRepository();
    const rawData = await stockRepo.fetchDateRangeData(
      ticker,
      startDate,
      endDate
    );
    let chartData = transformDateRangeData(rawData);
    console.log(rawData[0]?.Datetime);
    populateIntradayTable("datum:", rawData);
    renderChartByType(chartData);
  } catch (error) {
    console.error("failed to load date range data:", error);
  }
}
```

ChatGPT said:

Je zit op het juiste spoor — maar hier is wat fout gaat:

✗ Fout in deze regel:

js
CopyEdit

```
populateIntradayTable("datum:", rawData);
```

Je geeft **twee argumenten**, terwijl `populateIntradayTable()` **slechts één verwacht** (namelijk de data-array).

✅ Oplossing:

Verwijder `"datum:"` uit je call:

```
js  
CopyEdit  
populateIntradayTable(rawData);
```

💡 Maar als je gewoon iets wil loggen over de datum:

Laat je `console.log` zo staan:

```
js  
CopyEdit  
console.log("datum:", rawData[0]?.Datetime);
```

Dit logt:

```
yaml  
CopyEdit  
datum: 2025-05-15 09:30:00
```

✅ Samenvatting van wat je moet doen:

```
js  
CopyEdit
```

```
console.log("datum:", rawData[0]?.Datetime); // laten staan = prima  
populateIntradayTable(rawData);           // correct aantal parameters
```

Daarna kun je je `formatDate()` netjes in `populateIntradayTable.js` gebruiken zoals eerder uitgelegd.

Laat weten als je wil dat ik alles netjes structureer of opschoon 🧹✅

You said:

```
console.log(rawData[0]?.Datetime);
```

-0 → undefined

ChatGPT said:

Als `console.log(rawData[0]?.Datetime);` `undefined` logt, betekent dat één van deze drie dingen misloopt:

🔍 Mogelijke oorzaken:

1. `rawData` is een lege array

Controleer dit:

```
js  
CopyEdit  
console.log("Lengte van rawData:", rawData.length);  
console.log("Eerste item van rawData:", rawData[0]);
```

Als `rawData.length === 0`, krijg je uiteraard `undefined` bij `[0].Datetime`.

2. De property heet anders (denk aan `datetime`, `date`, `timestamp` ...)

Check:

```
js
CopyEdit
console.log("Eerste keys:", Object.keys(rawData[0]));
```

Misschien zie je iets als:

```
css
CopyEdit
["datetime", "open", "high", "low", "close", "volume"]
```

Dan moet je dus schrijven:

```
js
CopyEdit
rawData[0].datetime
```

(let op kleine letters!)

3. `rawData` is geen array (maar een object of `null`)

Check met:

```
js
CopyEdit
console.log("Type van rawData:", typeof rawData);
console.log("Is Array?", Array.isArray(rawData));
```

Wat nu doen?

Plak dit even in je `try` -blok voor de `populateIntradayTable(...)` :

```
js
CopyEdit
console.log("Type rawData:", typeof rawData);
console.log("Is array?", Array.isArray(rawData));
console.log("Lengte:", rawData.length);
console.log("Eerste item:", rawData[0]);
console.log("Keys van eerste item:", rawData[0] && Object.keys(rawData[0]));
```

➔ Dan weet ik 100% zeker waar het misloopt en fixen we het direct.

Wil je dat ik je `transformDateRangeData()` functie ook even bekijk? Misschien gaat het daar mis.

You said:

```
response json {data: Array(21), ticker: 'AAPL'}
transformDateRangeData.js:3 raw date range: {Close: 201.87527465820312,
Date: 'Tue, 15 Apr 2025 04:00:00 GMT', Dividends: 0, High:
203.24347561495443, Low: 199.53834280484799, ...}
dateRangeData.js:25 Type rawData: object
dateRangeData.js:26 Is array? true
dateRangeData.js:27 Lengte: 21
dateRangeData.js:28 Eerste item: {Close: 201.87527465820312, Date: 'Tue, 15
Apr 2025 04:00:00 GMT', Dividends: 0, High: 203.24347561495443, Low:
199.53834280484799, ...}
dateRangeData.js:29 Keys van eerste item: (8) ['Close', 'Date', 'Dividends',
'High', 'Low', 'Open', 'Stock Splits', 'Volume']
```

ChatGPT said:

Perfekte debugging 💪

Je hebt exact ontdekt waarom `rawData[0]?.Datetime` `undefined` is:

✅ De reden:

De property heet `Date`, niet `Datetime`.

➡ Het correcte veld is dus:

```
js
CopyEdit
rowData[0].Date
```

↻ En in je tabel-rendering moet je dus:

```
js
CopyEdit
<td>${formatDate(item.Date)}</td>
```

Bonus:

Omdat je ook met intraday data werkt (waar wél `Datetime` voorkomt), is het slim om **een check in te bouwen**:

```
js
CopyEdit
const dateValue = item.Datetime || item.Date;
<td>${formatDate(dateValue)}</td>
```

Of specifieker voor je `populateIntradayTable()`:

```
js
CopyEdit
function formatDateSafe(item) {
  const dateString = item.Datetime || item.Date || "";
  return new Date(dateString.replace(" ", "T")).toLocaleString("en-GB", {
    day: "2-digit",
    month: "short",
```



```
    hour: "2-digit",
    minute: "2-digit",
  });
}
```

Wil je dat ik je `populateIntradayTable()` hierop aanpas? Of dat ik ze opsplits voor `intraday` vs `date range`? 🧠✅

▼ sorteer functies

geef me hardcoded voorbeelden van table sorteer functies

ChatGPT said:

hier zijn **twee concrete voorbeelden** van hardcoded sorteerfuncties:

✅ 1. Sorteer op Volume (hardcoded)

```
js
CopyEdit
import { populateDataTable } from "./populateDataTable.js";

populateDataTable(data);

let currentData = [];

export function setDataForSorting(data) {
  currentData = data;
}

let volumeAscending = true;

export function sortByVolume() {
  if (!currentData.length) return;

  const sorted = [...currentData].sort((a, b) => {
```

```

return volumeAscending
  ? (a.Volume || 0) - (b.Volume || 0)
  : (b.Volume || 0) - (a.Volume || 0);
});

volumeAscending = !volumeAscending;
populateTable(sorted);
}

```

👉 Hier wissel je telkens tussen oplopend en aflopend als je opnieuw klikt.

✅ 2. Sorteer op Date / Datetime

```

js
CopyEdit
let dateAscending = true;

export function sortByDate() {
  if (!currentData.length) return;

  const sorted = [...currentData].sort((a, b) => {
    const dateA = new Date(a.Datetime || a.Date);
    const dateB = new Date(b.Datetime || b.Date);
    return dateAscending ? dateA - dateB : dateB - dateA;
  });

  dateAscending = !dateAscending;
  populateDataTable(data);
}

```

🧠 Voordeel

- Geen abstractie, volledig begrijpelijk

- Werkt perfect voor jouw project
- Makkelijk uit te breiden naar `sortByOpen()`, `sortByClose()`, etc.

▼ Scripts

◆ `start_api.sh` voor macOS/Linux

```
bash
CopyEdit
#!/bin/bash
cd "$(dirname "$0")"
source venv/bin/activate
export FLASK_APP=stockdata.py
export FLASK_ENV=development
flask run --port=8080
```

| Zorg dat het script uitvoerbaar is:

```
bash
CopyEdit
chmod +x start_api.sh
```

◆ `windowsStart_api.bat` voor Windows

```
bat
CopyEdit
@echo off
cd /d %~dp0
call venv\Scripts\activate
set FLASK_APP=stockdata.py
```

```
set FLASK_ENV=development  
flask run --port=8080
```