

# Datalzyer

Nico Marra & Gabriel Greco

---

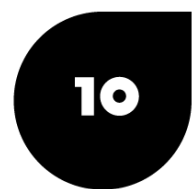
M431

30.10.2023



**TBZ**

Technische Berufsschule Zürich



## Inhalt

1. Der Projektauftrag in einer Kurzbeschreibung nach SMART .....	2
1.1. Eckdaten zum Projekt .....	2
1.2. SMART-Ziel .....	2
1.2.1 <b>S</b> pezifisch .....	2
1.2.2 <b>M</b> essbar .....	2
1.2.3 <b>A</b> kzeptabel .....	2
1.2.4 <b>R</b> ealistisch .....	2
1.2.5 <b>T</b> erminiert .....	2
2. Projektidee .....	3
3. IPERKA Vorgehensweise .....	3
<b>3.1. Informieren</b> .....	3
<b>3.2. Planen</b> .....	3
3.2.1 Nutzwertanalyse .....	4
3.2.2 Gant Diagramm .....	4
<b>3.3. Erstellen</b> .....	5
<b>3.4. Realisieren</b> .....	6
3.4.1. Scraper .....	6
3.4.2. Daten Formatieren und Sortieren .....	6
3.4.3. Pages .....	7
3.4.4. API und Open Graph .....	8
3.4.5. Google .....	9
3.4.6 Frontend .....	9
3.4.7 Anleitung für Lokalen gebraucht .....	10
<b>3.5. Kontrollieren</b> .....	10
3.5.1 Backend .....	10
3.5.2 Frontend .....	11
<b>3.6. Auswerten</b> .....	11
4. Schlusswort .....	12
5. Anhang .....	13

# 1. Der Projektauftrag in einer Kurzbeschreibung nach SMART

## 1.1. Eckdaten zum Projekt

Auftraggeber	Schule / Wir
Projekt Beginn	28.08.2023
Projekt Ende	30.10.2023
Beschreibung	Datenanalyse von Social Media Website über einen bestimmten Zeitraum
Projektergebnisse	Website
Budget	5.- einmalig
Terminvorgaben	Präsentation 06.11.2023
Projektmitglieder	Nico Marra & Gabriel Greco
Aufgabenteilung	Greco Frontend Marra Backend

## 1.2. SMART-Ziel

### 1.2.1 Spezifisch

Entwickeln und launchen Sie bis zum 30. Oktober 2023 eine Website namens "Datalyzer", die in der Lage ist, Daten von mindestens einer Sozial-Media-Plattform (insbesondere YouTube) zu scrapen und diese Daten in ansprechender und übersichtlicher Form darzustellen.

### 1.2.2 Messbar

Die Website sollte in der Lage sein, Daten von mindestens 100 YouTube-Videos pro Tag zu scrapen und diese Statistiken in Echtzeit auf der Website anzuzeigen.

### 1.2.3 Akzeptabel

Die Entwicklung der Website wird von den Teammitgliedern Gabriel und Nico durchgeführt, die über die erforderlichen Fähigkeiten und Ressourcen verfügen.

### 1.2.4 Realistisch

Das Ziel ist realistisch, da die Teammitglieder über die notwendigen Fähigkeiten und Erfahrungen verfügen und ausreichend Zeit bis zum 30. Oktober 2023 zur Verfügung steht.

### 1.2.5 Terminiert

Das Projektziel muss bis spätestens 30. Oktober 2023 erreicht sein, um die rechtzeitige Fertigstellung sicherzustellen.

## 2. Projektidee

Zu Beginn haben wir uns erstmals zusammengesetzt und erst mal Ideen gesammelt. Dazu haben wir ein Mindmap erstellt und uns ausgetauscht. Es war sehr komisch da, wir neu an der Schule sind und direkt eine Gruppenarbeit machen müssen. Wir haben uns ausgetauscht und erzählt, was wir aktuell im Betrieb machen und welche Programmiererfahrung wir schon haben. Nico hat erzählt, dass er schon viel Backend Erfahrung hat. Gabriel erläutert, er habe noch nicht ganz so viel programmiert. Gabriel hat gerade im Betrieb ein Frontend Projekt mit TailwindCSS gemacht. Da das gut passt, sind wir auf die Idee gekommen eine Webseite zu machen, die online mit einer Domain aufzufinden ist. Nico hat dann die Webseite Nindo.de gezeigt und hat gesagt, dass man so was eher einfach nachstellen kann. So sind wir auf den Entschluss gekommen, eine Abklatschversion von Nindo.de zu machen. Diese hat jedoch neue

## 3. IPERKA Vorgehensweise

IPERKA ist eine 6-Schritt-Methode, die für alle Arbeiten und Aufgaben zur Hilfe herangezogen werden kann. Die Abkürzung steht für informieren, planen, entscheiden, realisieren, kontrollieren und auswerten. Diese wendeten wir das komplette Projekt über an und gingen Schritt für Schritt durch.

### 3.1. Informieren

Wir haben uns den Auftrag angeschaut und überlegt, was wir alles abgeben müssen. Schauten das gegebene Kompetenzraster und das Gitlab an, schreiben uns auf, was wir alles brauchen und wie wir vorgehen sollten. Wir informierten uns über das neu erlernte, z. B. was eine Nutzwertanalyse ist, wo eine gewichtete Nutzwertanalyse sinnvoll ist und wo eine ungewichtete.

### 3.2. Planen

Unseren Projektauftrag haben wir gemeinsam definiert und unsere Tasks eingeteilt. Grob gesagt, Gabriel macht das Frontend und Nico das Backend. Wir haben uns Gedanken gemacht, welche Programmiersprachen wir benutzen, welche Daten wir Scrapen, wie wir das Hochschalten und von wo der Server sind. All diese Fragen haben uns beschäftigt. Auch, wie wir unsere Webseite nennen. Wir wollten sie erst Nindo-Fake nennen. Da wir aber die Webseite online laufen lassen möchten, könnte das sehr unwahrscheinlich, trotzdem nicht unmöglich, rechtliche Probleme geben. Deswegen sind wir auf den Name Datalyzer gekommen. Dieser Name besteht aus Data und Analysis. Wir haben uns Gedanken gemacht, wo wir die Website hosten. Zum Glück konnte Nico kostenlos in seiner Firma einen Windowsserver beziehen. Auf dieser wird der Scraper laufen und die Website gehostet. Die Domain Datalyzer.ch haben wir bei Hostpoint gekauft. Wir hätten auch die Möglichkeit gehabt, von Nicos Firma kostenlos eine Subdomain zu bekommen. Jedoch wollten wir dies nicht.

### 3.2.1 Nutzwertanalyse

Programiersprache Backend							
Kriterien	Wichtung	Python		Javascript		C#	
Vorkentnis Einfach zu lernen möglichkeiten vorhanede Lybaries	40	10	400	8	320	1	40
	30	8	240	7	210	4	120
	20	8	160	8	160	10	200
	10	7	70	8	80	8	80
		Gewichtung	Punkte	Gewichtung	Punkte	Gewichtung	Punkte
		Punkte 870 Note: <u>5.35</u>		Punkte 770 Note 4.85		Punkte 440 Note: 3.2	
Frontend Framework							
Kriterien	Wichtung	Tailwind		Bootstap		CSS	
Vorkentnis Einfach zu lernen möglichkeiten Effizienz	40	8	320	6	240	10	400
	10	8	80	7	70	10	100
	20	8	160	8	160	3	60
	30	9	270	9	270	1	30
		Gewichtung	Punkte	Gewichtung	Punkte	Gewichtung	Punkte
		Punkte 830 Note: <u>5.15</u>		Punkte 740 Note 4.7		Punkte 590 Note: 3.95	

### 3.2.2 Gant Diagramm

Dieses Diagramm wurde am Anfang erstellt und laufend angepasst. Vor allem Nico verschob eine Menge in die erste Herbstferienwoche, da er neben dem Arbeiten nicht für das Schulprojekt arbeiten mochte. Jedoch jetzt am Schluss spiegelt es einen guten Ablauf, was wir wann getan haben.

Aufgaben	KW 34	KW 35	KW 36	KW 37	KW 38	KW 39	KW 40	KW 41	KW 42	KW 43	KW 44
HerbstFerien											
Nico											
Projekt Start											
Server Einrichtung											
Scraper bau											
Daten verarbeiten											
Daten Sortieren											
Charts Implementieren											
Seiten erstellen											
zusammenfügen											
Fertigstellen											
Dokumentation											
Abgabe	Montag, 30. Oktober 2023										
Gabriel:											
Tailwind initialisierung											
Git Erstellen											
Design überlegung											
Frontpage erster entwurf											
Design überarbeitung											
Design für alle Pages erstellen											
Auf Live Server implementiern											
Dokumentation											

### 3.3. Erstellen

Als wir diese Fragen beantworten, gingen wir zum Erstellen. Wir haben für die Programmiersprachen auf unsere Stärken vertraut, da es für uns optimal ist als erstes Projekt erst mal auf die Stärken und Erfahrungen zu vertrauen. Fürs Backend benutzen wir Flask von Python, fürs Frontend benutzen Tailwindcss. Wir haben uns dann die Domain bei Hostpoint gekauft für 5.- im Jahr. Wir mussten uns für den Server uns überlegen, welchen wir mieten, da die schon recht teuer sind und vor allem haben wir mit viel Daten zu tun. Dies führt zu viel Speicherplatz und dieser ist relativ teuer (im Verhältnis zum Lohn eines erst Lehrjahr Lehrling) Nico kann zum Glück einen Server vom Betrieb organisieren, welches uns eine Menge an Kosten spart. Auch haben wir unser Git eröffnet und alle Frameworks heruntergeladen und Files. Alle Anfragen laufen über main.py. Nachdem wir den Server zum Laufen bringen konnten, haben wir als Test ein altes Template File von Nico benutzen, um zu schauen, ob die Webseite läuft.

Nico hat die Domain über Cloudflare eingebunden. Dies hat zwei Vorteile. Erstens, weil Cloudflare kostenlos SSL-Zertifikat zu Verfügung stellt und zweitens, die Sicherheit. Da man über Cloudflare Tunnels, die Website freigeben kann, ohne Portfreigaben, macht Cloudflare die Infrastruktur enorm sicher. Zudem versteckt Cloudflare die IP, somit findet man nie oder nur sehr schwer die IP-Adresse des Servers. Der Cloudflare Tunnel ist praktisch eine VPN zwischen Cloudflare und dem Server. Diese Verbindung ist verschlüsselt und nur sehr schwer verfolgbar. Zudem ist er sehr einfach installierbar. Nur die Software von Cloudflare installieren und anmelden. Schon ist er einsatzbereit.



The screenshot shows the Cloudflare DNS management interface for the domain 'datalyzer.ch'. It includes a search bar, a table of DNS records, and various action buttons.

Type	Name	Content	Proxy status	TTL	Actions
CNAME	datalyzer.ch	4eabd95d-6dc1-45fa-94d9-ec5ef62edac.fargatunnel.com	Proxied	Auto	Edit

Die Webseite läuft schon (stand 18.9) auf datalyzer.ch und man kann sie im Browser finden.

Nico hat als Erstes ein Batch File geschrieben (start.bat) welches uns hilft, immer das aktuellste Git auf dem Server zu haben. Immer wenn datalyzer.ch/Restart aufgerufen wird, wird das neuste Git (Develop Branch) heruntergeladen und dieser ausgeführt. Jedoch haben wir zurzeit keine Security für diesen Command. Dies wird irgendwann hinzugefügt, falls wir Zeit haben.

```
@echo off

:a
git reset --hard
git fetch
git pull

start /B py scrapper.py > log.txt
start /B py main.py > log.txt

:loop
timeout /t 1 /nobreak > NULL
if exist "restart" (
    DEL restart
    echo Found 'restart' file. Exiting...
    taskkill /f /im python.exe
    goto a
)
goto loop
```

## 3.4. Realisieren

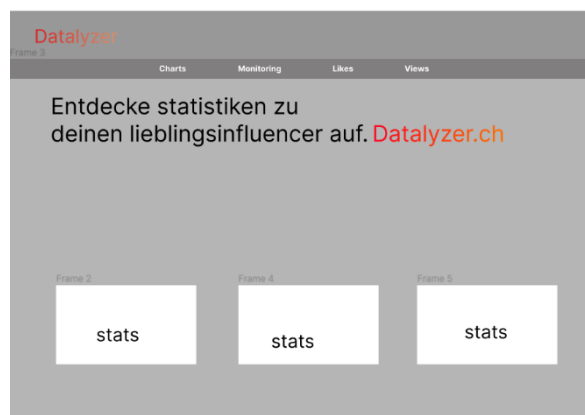
### 3.4.1. Scraper

Nico hat sich anfangs mit dem Scraper befasst (Def.: Beim Web Scraping werden Daten von Webseiten extrahiert und gespeichert, um diese zu analysieren oder anderweitig zu verwerten). Wir wollten schon früh Daten sammeln, damit wir bis zum Abgabetermin möglichst viele Daten haben. Zuerst wollte Nico über die YouTube API Daten erlangen. Jedoch wurde uns nach kurzer Zeit bewusst, dass diese sehr teuer ist, so viele Anfragen pro Tag an die YouTube API zu schicken. Deshalb verwerfen wir schnell diesen Gedanken. Wir überlegten, wie wir am besten die Daten erhalten. Da kamen wir auf eine gute Idee. Wir scrapen die Daten von Nindo. Nico programmierte einen Webscraper, welche auf die Nindo Website geht und stündlich die neuen Daten scrapt. Dies funktioniert gut. Nun wird stündlich ein JSON im /Output Folder erstellt.

```
{
  "08": {
    "Julien Bam": {
      "url": "/julien-bam",
      "views": 19000000,
      "subs": 61000000,
      "likes": 269796000
    },
    "Jindaouis": {
      "url": "/jindaouis",
      "views": 12000000,
      "likes": 50933000
    },
    "Aldi S\u00fcd": {
      "url": "/aldi-sud",
      "views": 732841000
    },
    "Bulien Jam": {
      "url": "/julien-bam",
      "views": 722920000,
      "subs": 21000000,
      "likes": 60811000
    },
    "Dave Henrichs": {
      "url": "/dave-henrichs",
      "views": 691545000,
      "likes": 48243000
    }
  }
}
```

Gabriel hat dann über figma.com ein erster Entwurf von der Webseite gemacht und angefangen zu planen, wie sie aussehen soll. Auch wichtig war der Entscheid von den main Farben. Da wir eine eigene Version von nindo.de machen, ist Gabriel den Farben weiss, violett (die von Nindo.de benutzt wird) umgangen. Gabriel ist nach dem Entscheid auf ein rot, oranges design gekommen und hat angefangen zu Realisieren.

So sah die allererste Design-idee aus von unserer Webseite. Gabriel hat für dieses Mockup design nicht viel zeit investiert und wollte nur eine grobe Idee haben wie die Webseite in ungefähr aussehen sollte. Auch hat Gabriel zu beginn nicht an die anderen Pages gedacht. Gabriel hat dann die Konfiguration von Tailwind aufs git gepusht und konnte schon mit dem design Starten.



### 3.4.2. Daten Formatieren und Sortieren

In dieser Zeit arbeitete Nico daran, mit Python die Daten zu bearbeiten, damit man sie richtig sortieren und ausgeben kann. Er programmierte ein Skript, welches in alle JSON-Dateien geht, eines Tages durchschnitt und einen durchschnitt über alles erstellt. Dies ging relativ einfach, kostete aber enorm viel Zeit. Er musste Fallbacks einbauen, falls bei Scrapen einen Error gab, oder Daten falsch eingetragen wurden. Dies wurde auch mehrfach mit verschiedenen Eingaben getestet (mehr dazu später)

Dieser Code befindet sich in der main.py als scraper\_formatter() Funktion.

Darin werden die Daten auch gleich nach Likes, Views und Subs sortiert. Es gibt ein Scoreboard, welches immer am jeweiligen Tag «top stats» ausgibt. Alle Daten kann über <https://datalyzer.ch/scraper/show> angezeigt werden.

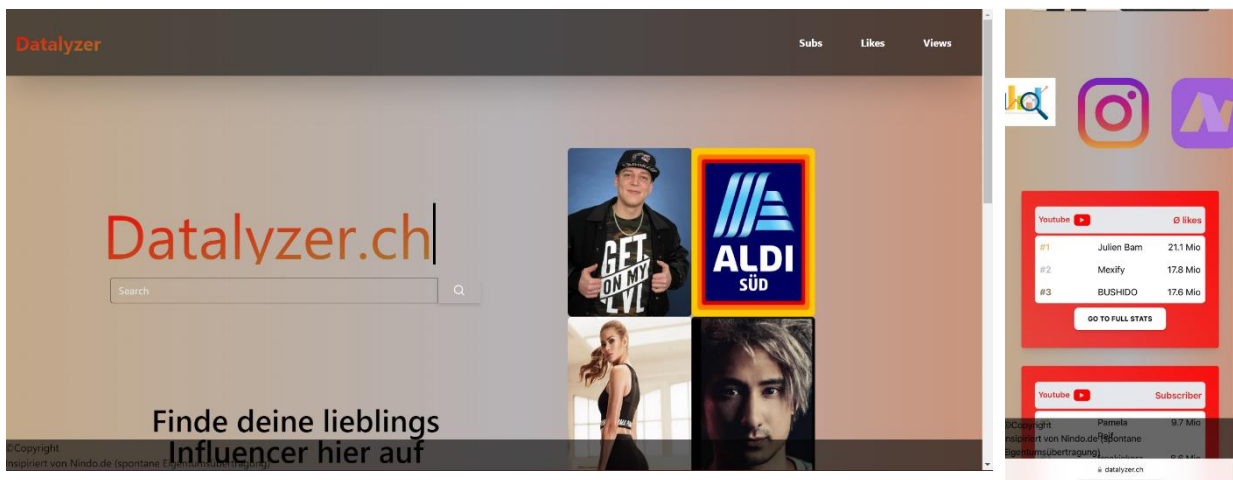
### 3.4.3. Pages

Wir erstellten 3 Seiten.

#### 3.4.3.1 FrontPage

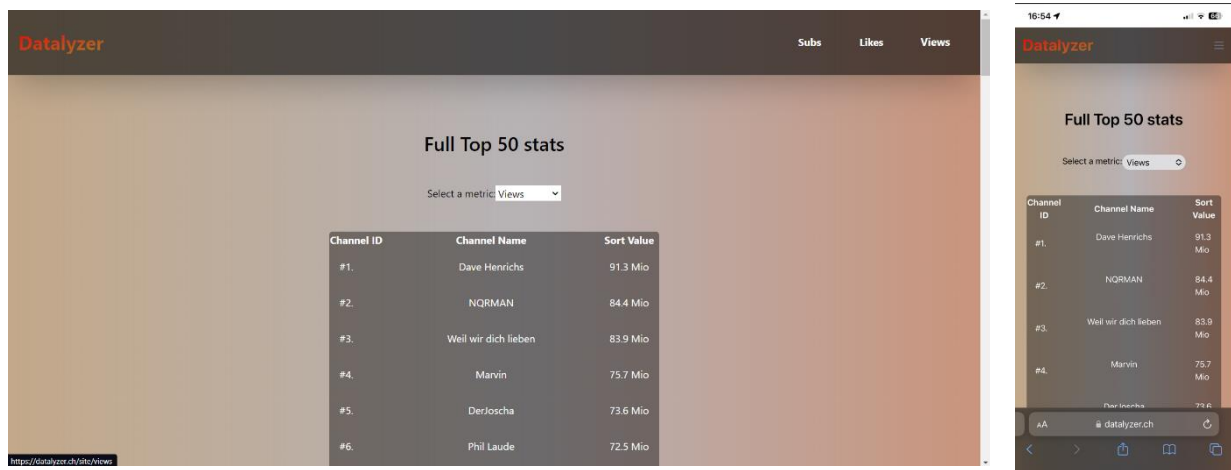
Die Seite, welche angezeigt wird, sobald man Datalyzer.ch aufruft. Dort sieht man die Top 3 im Scoreboard in Likes, Views und Subs. Zudem ein Button, welche auf die komplette Statistik/Scoreboard Page zeigt. Diese Page soll ein grober Überblick repräsentieren und eine schnelle Navigation ermöglichen.

Hier die Frontpage(oberer teil) Desktop Version



#### 3.4.3.2 Scoreboard

Auf dieser Page wird 1 zu 1 ein Scoreboard erstellt, welches man nach Likes, Subs und Views sortieren kann. Zudem kann man auf die Namen der einzelnen YouTuber klicken, um auf die YouTuber Page zu kommen (nächste Page)

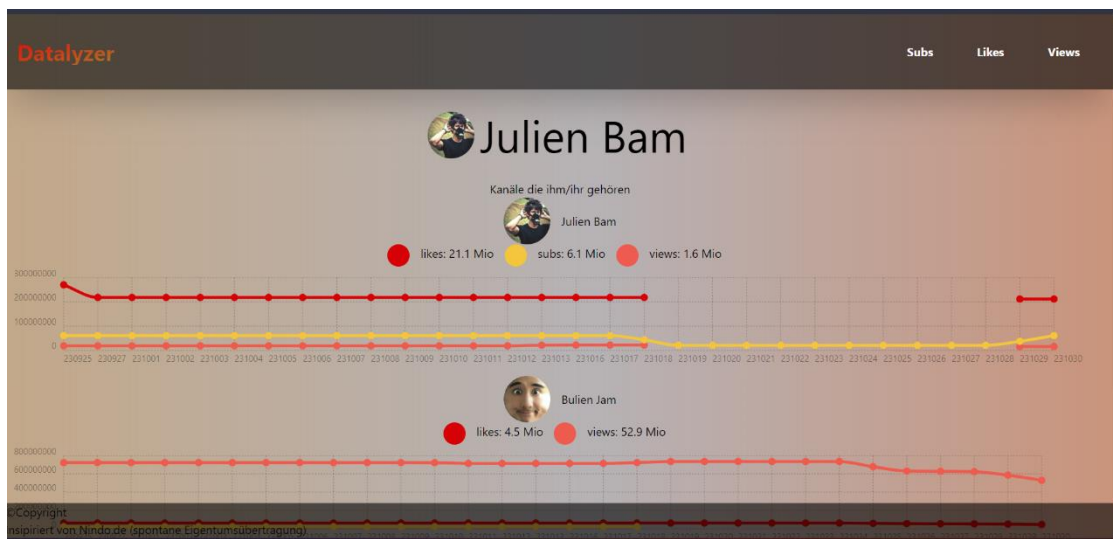


#### 3.4.3.3 Youtuber Page

Diese Page wird Komplet generiert und gibt es für jeden einzelnen YouTuber, der jemals bei uns in der Statistik war. Auf dieser Page werden die Daten angezeigt, welche wir gespeichert haben. Soweit vorhanden, gibt es Likes, Views und Subs. Zudem alle Channels, die ein YouTuber besitzt. Es gibt pro Channel auch ein Liniendiagramm mit (falls vorhanden) Likes, Views und Subs angezeigt werden.



Dies war mit Abstand die aufwendigste Page und auch mit Abstand die Page, die am meisten Fehler hat. Nico hat eine Menge Fehler behoben, jedoch noch lange nicht alle. Nico vermutet, dass er 90 % der Bugs gefunden und behoben hat.



Jede Page wird zum Teil automatisch generiert. Der Header und Footer wird automatisch angefügt, (ersichtlich auf <https://datalyzer.ch/>) dies hat der Vorteil, dass wir nur einmal den Header haben, in einem HTML. Falls wir Änderungen beispielsweise an der Navbar oder im Footer vornehmen möchten, ist dies in einer Datei zu tun(base.ht<\$ml). Jedoch wird das auf jeder Page angewandt.

### 3.4.4. API und Open Graph


Nico wollte OpenGraphs einbinden. Open Graph ist ein Protokoll, mit dem sich beeinflussen lässt, wie eine Webseite dargestellt wird, wenn sie z. B. auf Social Media geteilt wird. Diese werden durch Metatags eingefügt. Zuerst waren sie statisch, doch Nico programmierte eine API, welche die Meta tags anpassbar machte. Auf <https://datalyzer.ch/settings> kann man nun, OpenGraph Bild und Titel bearbeiten. Nico hat hinzugefügt, dass man das Favicon ändern kann. Dies sollte man in Zukunft noch Passwort schützen, wird jedoch nach hinten verschoben. Serverseitig nimmt der Code alles entgegen und schreibt es in ein JSON. Wir haben uns der Einfachheit halber für ein JSON als Config Datei entschieden. JSON hat der

Vorteil in Python sowie in JavaScript enorm schnell Daten auszulesen sowie auch zu schreiben. Wir haben für die API drei Befehle. Einer, der «get» welcher Daten abfragt und entweder als Text

```
@application.route("/api/<what>/<format>")
def api(what,format):
    with open("config.json") as f:
        data = json.load(f)
    if what in data.keys():
        if data[what]["type"] == "IMG":
            return redirect(data[what]["value"])
        else:
            daten = data[what]["value"]
    if format in ["json","text"]:
        if format == "text":
            return daten
        else:
            return jsonify({what:daten})
```

```
@application.route("/static/image/<img>")
def img(img):
    return send_file(f"images/{img}")
```

oder als JSON zurückgibt. Ein anderer, mit dem man über die API die Daten in der Config.json bearbeiten kann. Der dritte und letzte im Bunde hat wenig mit einer API zu tun. Er dient dazu, Bilder zurückzugeben. Da es Nico störte, dass die Bilder nicht landeten, wenn man Lokal arbeitet, machten Nico ein kleines Stück Code, welches über eine URL das gewünschte Bild zurückgibt.



```
<link rel="icon" href="/api/icon/img" type="image/x-icon">
<meta property="og:title" content="[Placeholder_OGTitle]" />
<meta property="og:image" content="https://datalyzer.ch/api/og_img/img" />
```

### 3.4.5. Google

Nico implementierte Google Analytics auf jeder Seite. Jedoch nur das Minimum an Daten sammeln, da wir nur wissen möchten, wie viele Views wir haben. Wir möchten nicht ein Datensammler werden, sondern nur aus Neugier wissen, wann und wer auf die Seite geht.

Das ging erstaunlich schnell. Nico hat das Script, was Google uns gab in das base.html eingefügt und schon war es auf jeder Page funktionsbereit. Falls wir unsere Seite auch live schalten möchten, hat Nico uns bei [search.google.com](https://search.google.com) angemeldet und eine Sitemap Index Page gemacht. (<https://datalyzer.ch/sitemap>) Jetzt crawlt Google unsere Page und auch wenn wir sie nicht öffentlich in die Suche freigeben, gibt uns Google Tipps, was noch fehlt oder wo etwas zu langsam oder nicht korrekt lädt.

### 3.4.6 Frontend

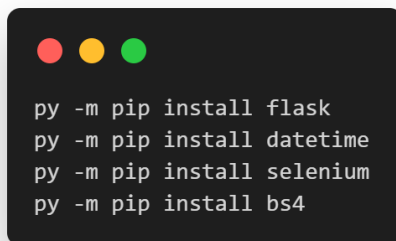
Beim Start hat Gabriel dann direkt von dem Grauen Hintergrund zu einem gradient Hintergrund gewechselt und hat auch die Navbar anders gestalten. Den Titel habe ich gemacht und einen Einleitungssatz geschrieben und rechts daneben 4 Bilder von Influencer die in der Top-5 vorhanden sind. Dann kam es zum Designen der Cards, auf welche die Top- 3 vertreten ist in den Kategorien Likes, Abos und Views. Für die Cards habe ich bisschen gebraucht, da sie ein passendes Design haben sollen. Nico hat dann noch die separaten pages gemacht, mit der ganzen Statistik und mit einem YouTuber im Fokus. Diese habe ich dann ebenfalls designend. Da Nico eine /Restart Route gemacht hat, die den neusten Stand vom Git pulled gab es viele /Restart anfragen und vor allem habe ich diese Funktion benutzt, um die Webseite zu benutzen. Normalerweise machen es viele mit dem live Server auf dem Localhost, aber da es einige Bugs gab und es mit dem /Restart genauso gut (wenn sogar besser) funktionierte, präferierte ich diese mehr. Der einzige Nachteil, ist das es viele Pushes von meiner Seite gab, da ich auch schon kleine Changes hochgeladen habe auf die Webseite.

### 3.4.7 Anleitung für Lokalen gebraucht

Wie gewünscht gibt es hier eine kleine Anleitung, wie man die Website Lokal starten kann. Dies ist zum Glück relativ einfach. Jedoch gibt es im Git keine daten. Deshalb vorab, es kann sein das es nicht richtig funktioniert. Sie können den Scraper einmal starten, jedoch haben sie das nur einen Datenpunkt und nicht wie auf der Website hunderte.

Zuerst brauchen sie Python. Wir benutzen die Version 3.9, jedoch sollte es auf jeder neueren oder etwas älteren Version funktionieren

Nun müssen sie alle Pakete installieren. Hier die Commands



```
py -m pip install flask
py -m pip install datetime
py -m pip install selenium
py -m pip install bs4
```

Nach dem erfolgreichen Installieren aller Pakete können sie bereits versuchen die main.py Datei zu starten. Dazu im CMD «py main.py» eingeben. Alle weiteren Informationen stehen dann in der Console, wie z. B. Port oder IP

## 3.5. Kontrollieren

Die Website haben wir mehrfach getestet. In der Dokumentation unterteilen wir in Front und Backend Tests.

### 3.5.1 Backend

Jede Funktion im Backend wurde auf Funktion und mögliche Eingaben getestet. Dabei wurden fragen geklärt wie z.b was ist, wenn als Subs Count ein String anstatt ein INT zurückgeben wird? Oder was ist, wenn ein Key Value ein leeren Sting ist? Oder was ist, wen ein Key im JSON nicht vorhanden ist?

Diese noch eine Menge mehr hat Nico Möglichkeiten Nico das Backend getestet. Dabei sind auch eine Menge Bugs aufgetaucht. Die Bugs, die Nico gefunden hat, haben wir behoben.

Besonders auf die Sortier-Funktion hat Nico ein Auge gehabt, da dort viel schief laufen kann. Deshalb nahm er sich für diese Funktion ganze 6h Zeit und testete diese intensiv. Aber auch der Scraper wurde getestet. Da wir abhängig von [Nindo.de](https://nindo.de) sind und sie jederzeit ihr Frontend ändern können, ist immer ein Restrisiko, dass der Scraper nicht mehr richtig funktioniert. Jedoch hat Nico eine Menge Fallbacks implementiert, welche verhindern sollte, dass falsche Daten und falsche Datentypen in die JSON geraten. Falls dies dennoch passiert, ist der Sortierer gut genug, um diese Daten zu ignorieren oder sogar zu reparieren.

### 3.5.2 Frontend

Beim Frontend ist wichtig das die Symmetrie stimmt. Zu beginn gab es noch Schwierigkeiten mit dem tailwind.config file. Lange war die Webseite auf einer Datei index.html. Doch als Nico die Files aufgeteilt hat für eine bessere Übersicht gab es einige bugs die, die Webseite nicht schön aussehen lassen. Zwar tauchten diese Bugs paarmal auf, Aber sie konnten behoben werden. Wichtig ist auch das die Webseite Responsive ist d.h. auf Desktop, Laptop und Mobile soll sie für ihre Proportionalität gut aussehen. Mit Tailwind war das sehr einfach und konnte gut implementiert werden. Auch musste ich achten das die Animationen die ich hinzugefügt haben Einwand frei laufen und es keine bugs entstehen. Aufgrund dessen habe ich die auch erst später hinzugefügt. Auch habe ich geschaut das die buttons richtig redirecten.

Uns ist bewusst, dass die Page noch einige Bugs hat. Alle zu finden und zu beheben ist in dieser kurzen Zeit unmöglich. Jedoch schätzen wir, läuft die Page zu 90 % einwandfrei.

### 3.6. Auswerten

Das Projekt ist aus unserer Sicht mehrheitlich gelungen. Wir haben viel gelernt und uns hat es Spass gemacht, was in unsren Augen der wichtigste Aspekt für ein Projekt ist. Wir haben beide viel Zeit und Herzblut investiert. Ein paar Features waren noch geplant, wie z. B. die Settings und Restart Passwort zu schützen. Oder bei der Site Page ein Diagramm einzubauen. Jedoch reichte die Zeit nicht mehr. Das Ziel wurde erreicht und sogar übertroffen. z. B. der Aspekt, «Daten von mindestens 100 YouTube-Videos pro Tag zu scrapen». Wir scrapen pro Stunde 150 Videos/Channels. Pro Sektion (Subs, Views, Likes) sind es ca. 50 Channels.

Die Zusammenarbeit lief mehrheitlich gut. Wir hatten eins zwei Auseinandersetzungen bezüglich Deadline Einhaltung, jedoch Kleinigkeiten, wenn man das Produkt anschaut. Jedoch war es sehr spannend, sich so wenig zu kennen und bereits ein Projekt zusammen zu machen. Wir sagten uns jedoch, dass es später mit den Kunden nicht anders sein wird. Deshalb arbeiten wir so gut wie möglich zusammen und unterstützen uns gegenseitig, wo wir konnten.

Besonders stolz ist Nico auf den Sortier-Prozess. Dieser ist nahezu fehlerfrei (wir haben keine Fehler mehr gefunden, jedoch gibt es immer Bugs, wir finden sie nur nicht) und funktioniert seitdem in Betrieb nehmen einwandfrei.

Wir hätten uns mehr Zeit nehmen können mit dem Planen. Wir sind beide eher Menschen der Taten, «Learning by doing» war unser Motto. Jedoch hätte man sich Arbeit ersparen können und wäre etwas produktiver gewesen. Zudem wäre die Dokumentation einfach zu schreiben gewesen, wenn man früher begonnen hätte.

Jetzt im Nachhinein hätte man Google Tagmanager von Anfang an mit einplanen sollen. Nico musste im Nachhinein eine Menge Änderungen vornehmen. Wenn man einmal die Google Analytics Page besucht hätte, wüste man auf was man achten müsste.

Wir sind stolz auf unsere Arbeit, damit stehen wir mit unsrem Namen.

## 4.Schlusswort

Es ist ein gelungenes Projekt zu beginn unserer Lehre und es hat uns viel spass gemacht und wir haben eine menge neues dazu gelernt zum Thema Projekt-Management und es ist sehr interessant. Ein Besonderes Danke geht an [OPTEN AG](#), der Lehrbetrieb von Nico. Danke, dass wir den Server benutzen durften, dass sparte und eine Menge Zeit bei der Einrichtung, da alles schon vorhanden war, aber auch eine Menge Geld.

## 5. Anhang

Der Code ist (hoffentlich) für immer unter einem von diesen Links verfügbar.

<https://datalyzer.ch/git>

<https://github.com/gabrielgreco11/datalyzer>

<https://github.com/gabrielgreco11/datalyzer/archive/refs/heads/main.zip>

Da wir Stolz auf dieses Projekt sind, möchten wir das wir den Code für immer behalten. Deshalb so viele Möglichkeiten den Code zu haben, damit wir hoffentlich egal wann, denn code öffnen können.