# IpShare

Studienrichtung
Technische Informatik

Hochschule
Augsburg University of
Applied Sciences

Fakultät für
Informatik

Jan Luis Holtmann

Betreuer: Erich Seifert

Abgabedatum: 02.02.2024

Hochschule für angewandte
Wissenschaften Augsburg

An der Hochschule 1
D-86161 Augsburg

Telefon +49 821 55 86-0
Fax +49 821 55 86-3222
www.hs-augsburg.de
info(at)hs-augsburg-de

Fakultät für Informatik
Telefon +49 821 55 86-3450
Fax +49 821 55 86-3499

An der Hochschule 1
86150 Augsburg
Telefon +49 821 55 86-3450
max@hs-augsburg.de

# Inhaltsverzeichnis

# Kapitel 1

# Motivation und Anforderungen

Musstest du schon einmal die IP-Adresse von einem anderen Gerät per Hand abtippen? Wahrscheinlich nicht und wenn hat es bestimmt auch nicht weh getan. Aber wäre es nicht viel besser stattdessen an beiden Geräten den Browser zu öffen und auf eine Webseite zu gehen, welche die IP-Adresse von einem Gerät zum anderen überträgt. "Nein, es dauert genauso lang" höre ich dich sagen. Wahrscheinlich hast du recht. Na ja, ich habe trotzdem einige Monate an einer Webseite entwickelt, die im Prinzip das macht. Frei nach dem Motto automatisiere 3 Monate, eine Aufgabe die 3 Sekunden dauert. Bei IPShare handelt es sich also im Prinzip um ein overengineertes Clipboard.

## 1.1 Projektidee

Spaß beiseite, tatsächlich habe ich einen Anwendungsfall, indem ich so etwas brauche. An unserer Hochschule bekommt man im Eduroam-WLAN eine öffentliche IP-Adresse. Für Experimente ist das als Student natürlich ein Fest. Jedenfalls wollte ich mich von einem Raspberry Pi aus einem 5G Netz zu meinem Laptop verbinden. Dabei handelt es sich nebenbei um das Maveric-Projekt. Das funktioniert auch, da ich ja eine öffentliche IP bekomme. Nur ändert sich diese IP gelegentlich. Leider kann ich keine Angaben dazu machen, wann oder wie oft das Vorkommt. Es wäre nicht schlecht, automatisiert die neue IP herauszubekommen, da ich nur umständlich Zugriff auf den Raspberry Pi habe. Während ich das hier schreibe, ist das noch nicht erprobt worden, sollte aber dank API funktionieren. Diese Problematik gab mir jedenfalls die Idee für das Projekt.

## 1.2 Ziel

Ziel ist eine Anwendung über die sich IP-Adressen Teilen lassen. Ob mein Projekt große Verwendung für andere hat, bleibt abzuwarten. Im Wesentlichen ging es darum mich mit Webentwicklung und im speziellen Flask zu beschäftigen, etwas zu experimentieren und dabei zu lernen. Dementsprechend sind manche Designentscheidungen auch getroffen, einfach um zu experimentieren. Manche Dinge sind daher auch nicht ganz einheitlich implementiert.

## 1.3 Bestehende Lösungen

Es gibt bereits Webseiten, welche einem die eigene öffentliche IP Adresse verraten. Diese währen z.B. whatismyipaddress.com[7] oder whatismyip[3]. Sie erlauben es aber nicht, die Adresse von einem anderen Gerät zu bekommen.

## 1.4 Anforderungen

Auf die Details möchte ich im Kapitel zu Planung und Entwurf 2 eingehen. Im groben sollen IP-Adressen von einem Gerät geteilt werden können, um sie dann von einem andern Gerät abrufen zu können. Als Rahmenbedingung für das Projekt war noch gegeben, dass es sich um eine multi-user-fähige Webanwendung mit Datenbank handeln soll. Die Entwicklung sollte nach einem agilen Vorgehensmodell erfolgen.

# Kapitel 2

# Planung und Entwurf

## 2.1 Organisation

Da ich nicht im Team arbeite, ist die Organisation trivial. Um einen groben und flexiblen Plan zu haben, orientierte ich mich an Scrum und verwendete einen Backlog mit User-Stories. Eine Unterteilung in Product Backlog und Sprint Backlog war hier nicht angebracht. Die Einträge (User Stories) im Backlog sind nicht technisch, sondern fachlich. Sie folgen einem einheitlichen Schema:

Als “Rolle” möchte ich “Ziel/Wunsch” (, um “Nutzen”).

### 2.1.1 User Rollen

Ich verwende drei einfache Rollen, um Personen zu klassifizieren, die mit der Anwendung interagieren:

- Visitor - jemand der die Seite nur besucht.
- User - jemand der einen Account hat.
- Admin - jemand der die Seite verwaltet.

### 2.1.2 User Stories

☑ Als Visitor möchte ich meine IP-Adresse teilen, um sie auf einem andern Gerät abzurufen.

☑ Als Visitor möchte ich über die Risiken des Teilens, informiert werden.

☑ Als Visitor möchte ich über das Anlegen eines Users informiert werden.

☑ Als Visitor möchte ich einen Accout registirern, um User zu werden.

☑ Als Visitor möchte ich die App ohne Cookies nutzen können.

☑ Als Admin möchte ich eine requirements.txt.

☑ Als User möchte ich eine Liste aller meiner IPs sehen.

☑ Als User möchte ich meine IP Passwort geschützt teilen.

⍈ Als User möchte ich sehen wann die IP geteilt wurde.

☐ Als User möchte ich leicht herausbekommen wie ich die API verwenden kann.

☑ Als User möchte ich mich anmelden.

☑ Als User möchte ich meinen account löschen können.

☑ Als User möchte ich meinen Passwort ändern können.

☑ Als User möchte ich meinen Remember Cookie einstellen können.

☑ Als User möchte ich meine Token invalidieren können.

☒ Als User möchte ich meine Adressen veröffentlichen können.

☑ Als User möchte ich öffentliche IPs importieren/benennen.

☑ Als User möchte ich Adressen manuell eingeben können.

☐ Als User möchte ich IPv4, IPv6 und Url validiert bekommen.

☐ Als User möchte ich meine Adressen in Gruppen teilen können.

☑ Als User möchte ich meine Adressen als QR-Code angezeigt bekommen.

☐ Als Admin möchte ich eine Firewall.

☐ Als Admin möchte ich AGB und Datenschutzerklärung nutzen.

☐ Als Admin möchte ich IPs GEO blocken bzw. differenzieren.

☑ Als User möchte ich meine Aderssen automatisch aktualisiert haben.

☑ Als Admin möchte ich Rate Limits auf den Routen, um vor DOS-Attacken geschützt zu werden.

☒ Als User möchte ich einen Cookie banner.

⏩ Als Visitor möchte ich keine Cookies.

☐ Als User möchte ich über E-Mail informiert werden können, wenn der Service nicht verfügbar ist oder sich ändert.

? Als User möchte ich einen Reverse VPN öffnen können.

☑ Als User möchte ich mein Adressen einfach mit API abrufen können.

☐ Als User und Visitor möchte ich ein übersichtliches Interface.

- ☑ Als User und Visitor möchte ich die Public von der Private Tabele unterscheiden kÃ¶nnen.
- ☐ Als User und Visitor möchte ich Mehrsprachigkeint. (Englisch und Deutsch evtl. weitere)
- ☑ Als User und Visitor möchte ich durch klicken Adressen und Namen Kopieren.
- ☐ Als User und Visitor möchte ich das Teilen durch klicken auf IP klar vermittelt bekommen.
- ☐ Als User und Visitor möchte ich Tooltips.

? Als Visitor möchte ich das es schwer oder unmöglich ist, einen Link zu erstellen der automatisch/unfreiwillig meine Adresse teilt.

? Als Admin möchte ich Datenbank-Migrationen.

☑ Als Admin möchte ich Blueprints.

☐ Als Admin möchte ich Logging, um Angriffe und Fehler zu erkennen.

☐ Als User möchte ich eine Weiterleitung zur Adresse über den Device Namen.

## 2.2 Frameworks und Entwicklungsumgebung

Ich verwende Flask als Framework und programmiere in PyCharm Professional 2023. Ich verwalte mein Python Environment und meine Pakete mit Anaconda[1]. Die verwendete Python Version ist 3.11.5. Zur Versionierung verwende ich git und GitHub. Die verwendeten Bibliotheken und deren Versionen finden sich im environment.yml oder im requirements.txt. Hier eine kurze Übersicht:

- Flask
  - APScheduler
  - Bcrypt
  - Limiter
  - Login
  - SocketIO
  - SQLAlchemy
  - WTF
- NumPy
- OpenCV
- qrcode
- Pillow
- Requests
- PyJWT
- urllib3
- pytest

## 2.3 Datenbankstruktur

Die Datenbank ist relativ simple. Es gibt eine User- und eine Address-Tabelle. Es gelten unter anderem folgende Abhängigkeiten:

$$id \dot{\rightarrow} passwort \tag{2.1}$$
$$id \dot{\rightarrow} remember \tag{2.2}$$
$$id \dot{\leftrightarrow} alternative\_id \tag{2.3}$$
$$id \dot{\leftrightarrow} name \tag{2.4}$$
$$id, device\_name \dot{\rightarrow} address \tag{2.5}$$
$$id, device\_name \dot{\rightarrow} last\_updated \tag{2.6}$$

Abbildung 2.1 zeigt das Entity-Relationship-Modell der Datenbank. Zwischen shared_addresses und user besteht eine nur-eins-zu-null-oder-mehr-Beziehung. Der Fremdschlüssel user ist gleich der id in der user-Tabelle. Der name ist eindeutig und daher auch Schlüsselkandidat der Tabelle. Auch die alternative_id ist ein eindeutiger Schlüsselkandidat. Alle drei id, alternative_id und name determinieren sich gegenseitig.

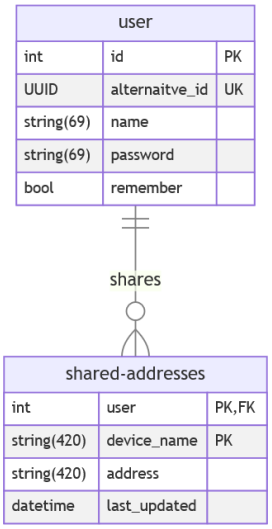


Abbildung 2.1: Entity-Relationship-Modell der DB

Wie Tabelle 2.1 und Tabelle 2.2 zeigen sind beide Relationen in 4ter Normalform. Eine Überführung in die 5te Normalform ist hier nicht nötig da die Relationen keine mehrwertigen Abhängigkeiten haben. Dementsprechend lassen sich durch eine Teilung auch keine neuen Informationen gewinnen.

| **Normalform** | **ok/nok** | **Begründung** |
|---|---|---|
| 1NF | ok | Alle Domänen enthalten nur einfache atomare Werte. |
| 2NF | ok | address und last_updated sind voll funktional abhängig von (id, device_name). |
| 3NF | ok | Es gibt keine transitive Abhängigkeit. |
| BCNF | ok | Die Determinante (id, device_name) ist auch Schlüsselkandidat. |
| 4NF | ok | Es gib keine mehrwertige Abhängigkeit. |
| 5NF | nok | Eine Teilung in address(id, device_namem, address) und updated(id, device_namem, last_updated) wäre möglich. |

Tabelle 2.1: Normalformen shared_addresses

| **Normalform** | **ok/nok** | **Begründung** |
|---|---|---|
| 1NF | ok | Passwort könnte vom Salt getrennt werden. |
| 2NF | ok | password und remember sind voll funktional abhängig von z.B. id. |
| 3NF | ok | Es gibt transitive Abhängigkeiten nur über die Schlüsselkandidaten. |
| BCNF | ok | Jede der drei Determinanten ist auch Schlüsselkandidat. |
| 4NF | ok | Es gib keine mehrwertige Abhängigkeit. |
| 5NF | nok | Teilung in Alternative(id, alternative_id), Name(id, name), Password(id, password) und Remember(id, remember) möglich. |

Tabelle 2.2: Normalformen user

# Kapitel 3

# Entwicklung

## 3.1 IpShare views

Die Tabelle 3.1 zeigt alle implementierten Routen. Zunächst die Einwicklung der IpShare views also der Routen, welche die Funktionalität der Webseite ausmachen.

### 3.1.1 Hauptseite

Serverseitig ist die Hauptseite simple. Der Server ermittelt lediglich die IP-Adresse des Clients und rendert die *index.html*. Dafür ist die Komplexität clientseitig höher. Das *index.js* übernimmt die Kommunikation mit dem Socket, näher beschrieben im Abschnitt 3.2, um die Address-Tabellen zu füllen. Beim Klicken auf den Namen, den Token oder die Adresse werden diese entsprechend in die Zwischenablage kopiert. Die Icons ändern sich, wenn mit der Maus darüber ist. Durch Klicken auf die Edit-Icons (Stift-Symbol) werden die Felder zur Eingabe des Device-Namen und der Adresse sowie Token und QR-Code angezeigt. QR-Code und Token werden hierbei in Abhängigkeit der Adresse und Device-Namen vom Server bezogen. Mehr dazu in den Abschnitten zu Token 3.4.1 und QR-Code 3.5.

### 3.1.2 Jetzt Teilen Route

Über die */now* Route können User ihre IP-Adresse sofort Teilen. Ist der Client authentifiziert, wird die *share_ip_now*() Funktion aus Abschnitt 3.1.3 aufgerufen und auf die Hauptseite weitergeleitet. Ist der Client nicht authentifiziert, wird *now.html* mit dem *ShareNowForm* gerendert. Hier wird der Visitor nochmal gefragt, ob er sich der Risiken des veröffentlichen bewusst ist. Das verhindert auch das jemand eine URL untergeschoben bekommt, die

dann unfreiwillig seine IP veröffentlicht. Ich will nicht ausschließen, dass dies nicht trotzdem möglich ist, aber zumindest ist es nicht so einfach.

### 3.1.3 Jetzt-Teilen-Funktion

Die *share_ip_now*() Funktion ist für das Teilen der vom Server automatisch ermittelten Funktion zuständig. Der Server ermittelt die öffentliche IP-Adresse des Clients wie in Listings 3.1 gezeitigt anhand der environment Variablen. Die Grundlage für den Programmcode ist hierfür von Tirtha Rahaman [8].

Listing 3.1: views.py

```
19      if request.environ.get('HTTP_X_FORWARDED_FOR') is None:
20          ip_addr = request.environ['REMOTE_ADDR']
21      else:
22          ip_addr = request.environ['HTTP_X_FORWARDED_FOR']  # if behind a proxy
```

Wenn die Address bereits in der DB ist, wird sie nur aktualisiert ansonsten wird sie neu angelegt. Je nachdem ob ein authentifizierter User oder ein Visitor teilt, wird der Device-Name anders ermittelt. Bei einem User wird aus dem User-Agent-Header der Browser und das Betriebssystem ermittelt. Bei mir wäre das z.B. “Firefox Windows”. Für den Visitor wird ein globaler Zähler verwendet, der einfach für jede Address um eins hochzählt. Zuletzt wird noch ein Socket Event emittiert, um alle Client Tabellen zu aktualisieren.

### 3.1.4 Impressum

Die Impressum Route rendert lediglich das Impressum aus *impressum.html*.

## 3.2 Socket

Um die Tabellen der User automatisch zu aktualisieren verwende ich Flask-SocketIO [9]. Zu empfehlen ist hier auch das Video vom Entwickler selbst: `https://youtu.be/OZ1yQTbtf5E?si=2_tiaAlKpT-p9HCP` Ich habe mich hier entgegen seinem Anraten für eine Read-Only-Session entschieden. Denn ich möchte die Session nicht über das Socket ändern. Es sind fünf Events implementiert auf die der Server reagiert:

- connect - beginnt die Kommunikation nach dem Verbindungsaufbau.
- disconnect - beendet die Kommunikation nach dem Verbindungsabbau.
- now - teil die IP Adresse mittels *share_ip_now*().
- save - speichert bzw. aktualisiert eine Adresse.
- delete - löscht eine Adresse.

Für den Client sind drei Events implementiert:

- user authenticated - setzt die *user_authenticated* Variable des Client.
- user table - übermittelt die User Adressdaten für die User-Table.
- public table - übermittelt die öffentlichen Adressdaten für die Public-Table.

Die *connect* Funktion sendet die Tabellendaten und emittiert das user-authenticated-Event. Außerdem betritt der Client den *room* des Users. So kann ein User mehrere Tabs öffnen, die sich gegenseitig aktualisieren. Beim Verlassen wird die *disconnect* Funktion aufgerufen. Ein authentifizierter Client verlässt hier den User room. Ein Visitor löscht seine Adresse aus der Datenbank. Wenn ein Visitor seinen Tab schließt, wird seine Adresse also aus der Datenbank entfernt und somit nicht mehr geteilt. Die *save* und *delete* Events sind, denke ich selbst erklärend. Sie speichern, aktualisieren oder löschen die übermittelte Adresse. Die *usertable* und *publictable* Events übermitteln die Daten im JSON Format. Das sieht dann z.B. so aus:

```
1 [{"device_name": "Firefox Windows", "address": "127.0.0.1", "last_updated": "5 days ago"},
2 {"device_name": "Broadcast", "address": "255.255.255.255", "last_updated": "5 days ago"}]
```

## 3.3 Login und User Management

Das User Management System verwendet Flask-Login und Flask-Bcrypt als Grundlage. Über die */register* Route kann sich ein User einen Account mit Username und Passwort anlegen. Er kann außerdem wählen, ob er eine Remember-Me-Cookie möchte. Sollte er sich dazu entscheiden, wird er

automatisch eingeloggt, wenn er die Webseite öffnet. Ansonsten kann er sich entsprechend über die */signin* Route einloggen. Über */signout* kann sich der User wieder abmelden. Beim registrieren und einloggen können Nachrichten angezeigt werden. Zum Beispiel, dass ein Benutzername schon in Verwendung ist oder dass das Passwort falsch eingegeben wurde. Auch noch wichtig ist die *load_user* Funktion, welche den User aus der Datenbank abfrägt. Als kleines extra kann mit der *redirect_next* Funktion ein User an eine bestimmte Adresse weitergeleitet werden. Das ist praktisch, wenn ein User z.B. auf die */me* Route möchte, aber nicht authentifiziert ist. Dann wird er zu */signin* weitergeleitet und nach dem login wieder zurück zu */me*. Darüber, hinaus erlaubt die */me* Route dem User sein Username/Pseudonym anzupassen, sein Passwort zu ändern, seine Token zu invalidieren und seine Remember-Me-Cookie Einstellungen zu ändern. Er kann auch seinen Account löschen. Diese */me* Route ist durch klicken auf den Nutzernamen rechts oben erreichbar.

## 3.4 API

Die API ist recht simple gehalten. Adressen können abgerufen, hochgeladen und gelöscht werden. Der Token übernimmt hierbei nicht nur die Authentifizierung, sondern gibt auch gleich an, um welche Adresse von welchem User es sich handelt. Das Abrufen kann dann z.B. in Python mit dem requests Paket wie folgt durchgeführt werden:

```
1    import requests
2    headers = {"Authorization": f"Bearer APITOKEN"}
3    print(requests.get("http://127.0.0.1:5000/v1/", headers=headers).text)
```

Hochladen oder Aktualisieren funktioniert wie folgt:

```
1    import requests
2    headers = {"Authorization": f"Bearer APITOKEN"}
3    data = "2.3.4.5"
4    requests.put("http://127.0.0.1:5000/v1/", data=data, headers=headers)
```

Auch das Löschen ist leicht:

```
1    import requests
2    headers = {"Authorization": f"Bearer APITOKEN"}
3    requests.delete("http://127.0.0.1:5000/v1", headers=headers)
```

### 3.4.1 Token

Die API ermöglicht es auch den Token für einen Device-Name auszustellen. Diese Funktion ist aber nicht direkt für den User gedacht. Sie wird vom

*index.json* verwendet, um den Token in der EDIT-Ansicht zum Kopieren zur Verfügung zu stellen. Der User sollte also erst manuell eine Adresse anlegen, um den Token zu erhalten.

Um die Token zu invalidieren, benötigt jeder user einen alternative_id, die sich ändern kann. Die Token werden auf die alternative_id ausgestellt. Um die Token zu invalidieren wird die alternative_id geändert. Alle Token sind dann für einen nicht existenten User und somit ungültig.

Bei den Token handelt es sich um JSON Web Tokens (JWT) welche mit der PyJWT Library [6] erstellt werden. Wie Abbildung 3.1 zeigt, beinhaltet der Token die alternative_id des User und den Device-Name. Diese Decodierung kann auf `https://jwt.io/#debugger-io` durchgeführt werden. Hier ist auch noch etwas Optimierungspotential bei der Payload größe. Das Vorgehen beim Einbinden des Token in den Header ist von cassiomolin[4].

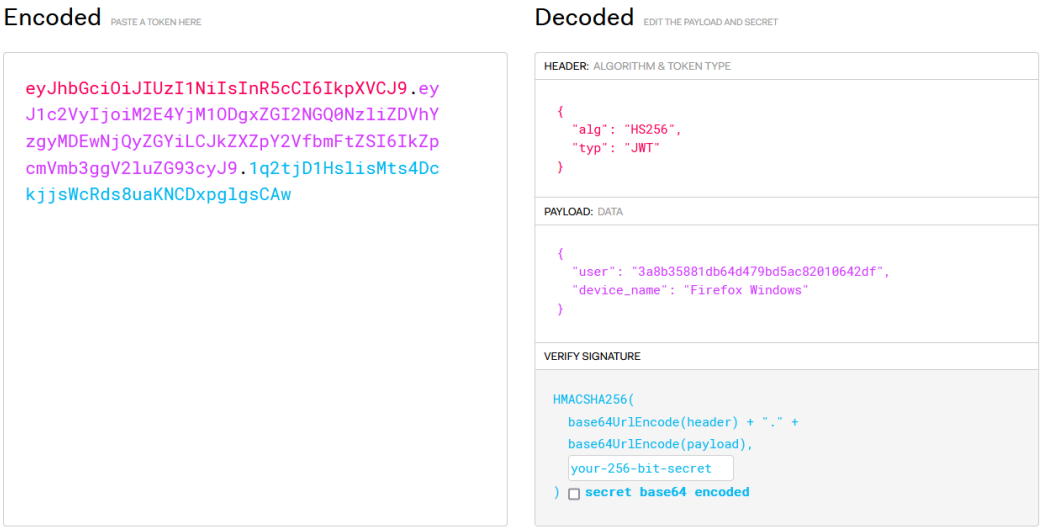


Abbildung 3.1: Beispiel jwt

## 3.5 QR-Code

Die QR-Codes werden mit der qrcode Bibliothek erstellt und im Anschluss mit OpenCV nachbearbeitet. Zum Versenden müssen sie noch ins PNG Rastergrafikformat konvertiert werden. Dazu wird die Python Imaging Library (PIL) names Pillow verwendet. Um die Datei nicht im System speichern zu müssen, wird in einen binary stream geschrieben. Dieser wird von der io Library über BytesIO durch einem in-memory byte buffer zur Verfügung gestellt. Mehr dazu unter `https://docs.python.org/3/library/io.html#`

`io.BytesIO`. Schließlich muss vor dem Senden der "'Lesekopf" noch mit $fp.seek(0)$ an den Anfang gesetzt werden.

### 3.5.1 Encoding

Eine Schwierigkeit war hier die Address in der URL an die API mitzugeben. Eine URL kann nicht einfach in eine URL geschrieben werden, da diese dann nicht mehr zu unterscheiden wären. Dieses Problem hat auch die Token API aus dem vorherigen Abschnitt. Auch ein Device-Name kann in einer URL unerlaubte zeichen wie ;/?:@&=+$,# enthalten. Die Zeichen müssen also escaped werden. Beim Client wird das im JavaScript mit $encodeURI()$[5] durchgeführt. Während in Python die $unquote\_plus()$[2] Funktion aus der urllib verwendet wird.

## 3.6 Cookies

Als Visitor soll die Seite ohne Cookies verwendet werden können. Denn so muss er keinen Cookie Banner wegklicken. Als eingeloggter User wird ein Session-Cookie benötigt. Optional kann ein User auch noch einen Remember-Me-Cookie bekommen. Um die Cookies von Flask für Visitors zu unterbinden wird ein CustomSessionInterface benötigt. Dieses überschreibt die $should\_set\_cookie()$ Funktion, so dass kein Cookie gesendet wird, wenn der user nicht authentifiziert ist. Leider bringt das einige Schwierigkeiten mit sich. Die Cross-Site-Request-Forgery (CSRF) Tokens für die Flask-Forms benötigen eigentlich Sessions. Ich musste daher extra ein SessionLessCSRF Token verwenden, der über die IP-Adresse funktioniert. Dieser ist allerdings nicht sehr sicher. Daher kommt der spezial Token nur für die register, signin und share_now Forms zum Einsatz. Um diese Forms zu verwenden, muss der User ohnehin nicht eingeloggt sein. Ein Angreifer kann auch so einen Account registrieren oder eine Adresse teilen.

Gelegentlich funktionierte in meinen Tests der Logout nicht richtig. Dann musste ich die Cookies manuell löschen. Mir ist nicht bekannt, woran das liegt, da ich Schwierigkeiten habe den Fehler zu reproduzieren. Es hat aber möglicherweise etwas mit dem CustomSessionInterface zutun.

## 3.7 APScheduler

Flask führt normalerweise nur Funktionen aus, um auf eine Anfrage zu reagieren. Um alte Visitor Adressen nach einer gewissen Zeit zu entfernen wird daher der flask_apscheduler benötigt. Mit ihm lässt sich die $clear\_old\_visitor\_addrs()$

Funktion jede Minute aufrufen. Diese kontrolliert dann, ob eine Visitor Adresse älter als 42 Minuten ist und löscht diese gegebenenfalls.

## 3.8 Scramble Effect

Um die Seite etwas spannender zu gestalten zeige ich einen Scramble Effect, wenn die Maus über Links und Buttons ist. Der Effect ist in *scramble.js* implementiert und wurde durch HYPERPLEXED inspiriert, welcher sich wiederum von KPR hat inspirieren lassen. Der verlinkte codepen zum vergleich. Im Prinzip werden die Zeichen durch zufällige Zeichen ersetzt und dann langsam die tatsächlichen Zeichen wieder angezeigt. Die Zeichen können durch Zahlen, Buchstaben oder Punkte ersetzt werden. Die *now.js* und *register.js* implementieren eine ähnliche Funktion, um einen Typewriter Effect für die Informationen zu erzeugen.

## 3.9 Checkbox

Da die Standard Checkboxen optisch nicht zur Webseite passten, habe ich welche mit css erstellt. Eine Schwierigkeit hierbei war, dass css nicht zwischen absoluter und relativer Positionierung animieren kann. Im unchecked Zustand (Abbildung 3.2) sollte die Box relativ zum Text positioniert werden. Im checked Zustand (Abbildung 3.3) sollte die Box absolut um den Text positioniert werden. Um ein saubere Animation zu erhalten, habe ich mich für absolute Positionierung entschieden. Die Box wird im unchecked Zustand mithilfe einer Variablen `--chars` positioniert. Diese Variable muss für jede checkbox im html-Code auf die Anzahl der Zeichen gesetzt werden. In css wird die Positionierung der linken Kannte im unchecked Zustand dann mit `left: calc(11.7rem - var(--chars) / 2);` berechnet. Der html-Code für eine Checkbox ist dann entsprechend so:

```
1     <div class="checkbox-container" style="--chars: 10ch;">
2         {{ settings_form.remember }}
3         <label for="remember">REMEMBER ME</label>
4     </div>
```


Abbildung 3.2: unchecked Checkbox


Abbildung 3.3: checked Checkbox

## 3.10 Limiter

Um sich gegen DOS-Attacken zu wehren kann mit dem Flask-Limiter für jede Route ein Limit gesetzt werden. So können für rechenintensivere Funktionen wie z.B. die QR-Code Erzeugung strengere Grenzen eingestellt werden. Die Limits sind über die Dekoratoren @*limiter.limit*() vor den Routen festgelegt. Diese Funktionalität habe ich auch leicht zweckentfremdet um die Login Versuche zu limitieren.

## 3.11 Tests

Auch an den Tests wollte ich mich zumindest kurz versuchen. Am Besten sind dabei die Tests für die API. Der Test für die */now* Route in *test_views* ist leider nicht mehr aktuell. Die Unit-Tests sind nicht ganz leicht, da in den Funktionen auf die Datenbank zugegriffen wird. Auch die Authentifizierung muss berücksichtigt werden. Ob und wie sich das JavaScript automatisiert testen lässt, ist auch noch unklar. Recht hilfreich empfinde ich die coverage Reports die in der Konsole mit `python -m pytest --cov-report=html --cov` erstellt werden können.

| **Blueprint** | **Route** | **Forms** | **Authentication** | **Methods** |
|---|---|---|---|---|
| *ip_share_views* | */* | | | HEAD, GET, OPTIONS |
| *ip_share_views* | */now* | ShareNowForm | | HEAD, GET, POST, OPTIONS |
| *ip_share_views* | */impressum* | | | HEAD, GET, OPTIONS |
| *login_views* | */register* | RegisterForm | | GET, OPTIONS, HEAD, POST |
| *login_views* | */signin* | LoginForm | | GET, OPTIONS, HEAD, POST |
| *login_views* | */signout* | | login required | GET, OPTIONS, HEAD |
| *login_views* | */me* | ChangePseudonymForm<br>ChangePasswordForm<br>InvalidateTokensForm<br>SettingsForm<br>DeleteForm | fresh login required | GET, OPTIONS, HEAD, POST |
| *api_views* | */v1/* | | | HEAD, GET, OPTIONS |
| *api_views* | */v1/* | | | PUT, OPTIONS |
| *api_views* | */v1/* | | | DELETE, OPTIONS |
| *api_views* | */v1/token/ < device_name >* | | | HEAD, GET, OPTIONS |
| *qr_views* | */qr/ < addr >* | | | HEAD, GET, OPTIONS |
| | */static/ < filename >* | | | |

Tabelle 3.1: Routen

# Kapitel 4

# Inbetriebnahme

Zur Inbetriebnahme auf einem lokalen Rechner sind die nachfolgenden Schritte notwendig.

## 4.1 Dateien Herunterladen

Die Dateien über `https://github.com/janluisho/ipshare` als ZIP herunterladen oder clonen und in das Verzeichnis navigieren.

## 4.2 Pakete Installieren

Ich benutze Anaconda und würde daher ein neues Environment anlegen.

```
1     conda create --name myenv --file .\environment.yml
```

Dies ist aber abhängig vom Paketverwaltungssystem. Vorausgesetzt man hat Python bereits installiert, können die Pakete auch mit pip installiert werden.

```
1     pip install -r requirements.txt
```

## 4.3 Datenbank anlegen

Um die Datenbank anzulegen muss die *db.py* Datei einmal ausgeführt werden. Dabei kann es zu einem *sqlalchemy.exc.InvalidRequestError* kommen. Als Workaround müssen in der *app/__init__.py* die Blueprints, gezeigt in Listings 4.1, temporär auskommentiert werden.

Listing 4.1: views.py

```
44 from app import Socket
45 from app.APScheduler import scheduler
46 from app.views import ip_share_views
47 from app.login_views import login_views
48 from app.api import api_views
49 from app.qr import qr_views
50
51 app.register_blueprint(ip_share_views)
52 app.register_blueprint(login_views)
53 app.register_blueprint(api_views, url_prefix="/v1")
54 app.register_blueprint(qr_views, url_prefix="/qr")
```

## 4.4 Ausführen

Ist die Datenbank angelegt, kann die Datei *run.py* ausgeführt werden, um die Flask Anwendung zu starten. Als Einstiegspunkt kann auf der Webseite die IP-Adresse mittig im Bild angeklickt werden, um die Adresse als Visitor zu teilen. Danach kann z.B. ein Account über die REGISTER-Schaltfläche rechts oben angelegt werden. Nach dem Teilen einer Adresse als eingeloggter User lohnt sich ein Klick auf das Stiftsymbol neben der Adresse. Es sollte dann ein QR-Code und der API-Token angezeigt werden.

# Kapitel 5

# Fazit

Die ursprünglichen Anforderungen wurden umgesetzt. Im Laufe der Zeit hatte ich noch zusätzliche Ideen, die noch nicht alle umgesetzt sind. Was genau erledigt oder verworfen wurde ist den UserStories in Abschnitt **??** zu entnehmen. Ob mein Projekt große Verwendung für andere hat, bleibt wie schon in der Einleitung erwähnt, abzuwarten. Momentan ermöglicht die Anwendung eine Abgeschwächte form des DynDns. Anstelle eines Domainnamens wird der Server nach der IP-Adresse zu einem Device Namen gefragt. DynDns ist natürlich ausgereifter, kostet aber evtl. etwas. Möchte man nur etwas ausprobieren oder hat eine Anwendung ohne Domainnamen ,könnte IPShare aber durchaus sinnvoll sein. Eine praktische Erweiterung für Websites, wäre hier eine automatische Umleitung von z.B. `ipshare.de/redirect/MeineAnwendung` zu `oeffentlicheip:5000/`. Eine weitere Idee, die ich hätte, ist ein "reverse Proxy", mit dem ohne Port-Forwarding Kontakt zum Server aufgebaut werden könnte.

# Literaturverzeichnis


[1] Anaconda. URL https://www.anaconda.com/.

[2] unquote plus. URL https://docs.python.org/3/library/urllib.parse.html#urllib.parse.unquote_plus.

[3] Whatismyip.com. URL https://www.whatismyip.com.

[4] cassiomolin. Best http authorization header type for jwt, 7. October 2021. URL https://stackoverflow.com/a/47157391.

[5] mozilla. encodeuri. URL https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURI.

[6] JosÃ© Padilla. Pyjwt on readthedocs. URL https://pyjwt.readthedocs.io/en/latest/.

[7] Chris Parker. Whatismyipaddress.com. URL https://whatismyipaddress.com/.

[8] Tirtha Rahaman. The below code always gives the public ip of the client (and not a private ip behind a proxy)., 10. April 2018. URL https://stackoverflow.com/a/49760261.

[9] Miguel Grinberg Â©. Flask-socketio, 2018. URL https://flask-socketio.readthedocs.io/en/latest/getting_started.html.