Projekt 2, Gruppe 3

Shorty – URL-Shortener

Andreas Brühl[*] Deniz Durmaz[†] Sebastian Hufeld[‡]
Jason Krimmel[§] Markus Rennings[¶]

23. August–6. September 2023

Betreut durch: Martin Dubb, Fabio Keller

[*] Datenbank
[†] Design, Frontend, Dokumentation, Korrektorat
[‡] Design, Datenbank, DB-Dokumentation
[§] Design, Frontend
[¶] API-Design, Backend, OpenAPI, Dokumentation, XeLaTeX, Korrektorat

Inhaltsverzeichnis

1 Einleitung

1.1 Kontext der Arbeit

Das Projekt wurde im Rahmen einer geförderten Weiterbildung erstellt. Der Träger der Weiterbildung ist die Techstarter GmbH[1]. Es gab mehrere Projekte zur Auswahl mit freier Gruppen- und Themenfindung. Das gewählte Projekt ist ein URL Shortener.

1.2 Motivation für diese Arbeit

In der Weiterbildung wurden verschiedene Methoden und Themenbereiche vermittelt. Diese sollen und werden in diesem Full-Stack-Projekt zusammengeführt und genutzt. So soll zum einen gezeigt werden, wie wir diese vielen Teilbereiche nutzen, um das Projekt umzusetzen.

Die größte Problematik ist es, sich erst einmal als Gruppe zu finden, aufzuteilen und zu sortieren, um die Anforderungen umzusetzen.

1.3 Zielstellung für diese Arbeit

Unsere Lösung wandelt lange Uniform Resource Locator (URL) in kurze und eindeutige URLs um. Bei URLs kann die Zeichenlänge je nach Darstellung in mehreren Zeilen ausarten. Das kann zu Problemen in Dokumenten, E-Mails oder Social-Media-Posts führen. Durch die gekürzten URLs wird dieses Problem behoben oder aber auch, wenn eine Zeichenbegrenzung vorhanden ist, wird diese nicht durch die lange URL kannibalisiert. Optisch sind Short-URLs ebenfalls nicht so aufdringlich.

1.4 Anforderungen für diese Arbeit

URL-Validierung Es ist erforderlich, eine URL-Validierung zu implementieren, um sicherzustellen, dass die eingegebenen URLs gültig sind.

URL-Kürzung Die Lösung muss in der Lage sein, lange URLs in kurze URLs umzuwandeln.

Generierung von Kurzlink-Codes Die Kurzlinks müssen eindeutig sein.

Benutzeroberfläche Es muss eine benutzerfreundliche Oberfläche sein, über die Nutzer URLs eingeben und Kurzlinks generieren können.

[1] https://techstarter.de/

Anzeige von Kurzlinks Benutzer bekommen bei Generierung des Kurz-Links ein Passwort mitgeteilt, mit dem sie sich die Statistik zu dem Link anschauen können. Diese Statistik umfasst eine Auswertung, wie oft auf den Link zugegriffen wurde, mit welchen Browsers und mit welchem Operating System (Betriebssystem) (OS). Auch der Zeitpunkt des letzten Zugriffs wird ausgewertet.

Datenbank Die Anwendung muss die ursprünglichen URLs und die zugehörigen Kurzlinks in einer Datenbank speichern und die Verknüpfung zwischen ihnen sicherstellen.

API Ein Application Programming Interface (Programmierschnittstelle) (API) muss entwickelt werden, um die Interaktion zwischen Frontend und Backend zu ermöglichen. Dies ermöglicht das Erstellen und das Abrufen von Kurzlinks. Das gleiche gilt für die dazugehörigen Statistiken.

Statistik und Analyse Die Anwendung muss die Daten über die Nutzung der generierten Kurzlinks sammeln und speichern. Einschließlich Anzahl der Klicks, Zeitpunkt der Nutzung und Informationen zu OS und Browser der Nutzer, müssen generiert werden.

Passwortschutz für Statistik Jeder generierte Kurzlink muss über ein eindeutiges Passwort verfügen, mit dem Benutzer den Link bearbeiten und die Statistiken einsehen können.

Programmiersprache Als Programmiersprache wurde Javascript (JS) vorgegeben. Es durften Bibliotheken verwendet werden.

2 Technische Grundlagen

Grundsätzlich sind weitläufige Kenntnisse aus der Webentwicklung erforderlich, um die Architektur und die Komponenten des Full-Stack-Projekt umzusetzen und zu verstehen.

Frontend- und Backend-Technologien Grundlegende Kenntnisse über die Technologien, die im Frontend und Backend verwendet werden, wie z. B. JS oder react.js, um die Benutzeroberfläche und die Serverseite zu entwickeln und gestalten.

Datenbanken Verständnis von Datenbankkonzepten und -operationen, insbesondere der Speicherung und Abfrage von Daten.

API Schnittstellen-Bildung und Integration zur Sicherstellung einer fehlerfreien Kommunikation zwischen den einzelnen Elementen.

Sicherheit Sichere Speicherung von gestellten Anfragen, Benutzern und Passwörtern.

Statistik und Daten Speicherung, Abruf und Visualisierung der vorhanden bzw. angefragten Daten (URL, Klicks) für die Erstellung der Statistik.

2.1 Benutzeroberfläche

Die geforderte benutzerfreundliche Web-Oberfläche (siehe Abb. 2.1) zur Eingabe und Generierung von Kurzlinks wurde vorgeplant mit den Design-Tools Balsamiq[1] und Figma[2]. Zur Umsetzung wurde React genutzt, um die Vorteile der modularen Komponenten zu nutzen.

2.2 Datenbank

Als Datenbank haben wir uns zum Schluss für eine SQLite3 Datenbank entschieden. Eigentlich war das Projekt mit einer Firebase Datenbank geplant, jedoch gab es bei unserer Implementierung einige Fehler, welche wir leider nicht innerhalb des Projektzeitraums beheben konnten. So haben wir uns schließlich dazu entschlossen, statt einer Online-Datenbank auf eine Offline-Datenbank zu switchen.

Dabei haben wir Firebase nicht gänzlich verworfen und sollte es eine Version 2.0 geben, so ist geplant dort dann auf den Firestore zu wechseln, sodass eine Online-Version der Datenbank zur Verfügung steht.

Die `url.db` sowie die `database.js` wurden unter folgendem Pfad gespeichert:
`server/src/db/url.db`
Dies wurde als genereller Speicherort für die Arbeiten der Datenbank ausgewählt.

[1] https://balsamiq.cloud/
[2] https://www.figma.com/

Abbildung 2.1: Startseite der Anwendung

Erklärung der einzelnen Felder, welche beim erstellen der Datenbank erstellt werden:

longURL hier wird die Ausgangs-URL hinterlegt

shortURL enthält die gekürzte URL

browser_* Anzahl der Aufrufe über einen bestimmten Browser

os_* Anzahl der Aufrufe über ein bestimmtes OS

lastClick Zeitpunkt des letzten Aufrufs der shortURL

clicks Anzahl der Clicks der jeweiligen ShortURL

timestamp Zeitpunkt zu dem der Link/Datenbankeintrag erstellt wurde.

expireDate Datum wann der Link erlischt (90 Tage nach Erstellung)

2.2.1 Datenbank-Struktur

Die Datenbank ist wie bereits erwähnt eine SQLite3 Datenbank. Das Schema für diese Datenbank haben wir zusammen erarbeitet (siehe Zeile 4 bis 20 in Lst. 2.1). In unserem Code wird als erstes überprüft, ob die Datenbank bereits existiert. Sollte dies nicht der Fall sein, so wird diese über die dritte Zeile im Code unter Lst. 2.1 erstellt.

2.2.2 Erstellen der Datenbank

Als Defaultwert werden für die Browser- und OS-Daten sowie für `clicks` und `lastClick` `0` verwendet. Diese haben wir initial gesetzt, da nach dem Erstellen des Links ja noch keinerlei Informationen vorhanden sind.

```
db.serialize(() => {
    db.run(`
        CREATE TABLE IF NOT EXISTS url (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            longURL TEXT NOT NULL,
            shortURL TEXT NOT NULL,
            passwd TEXT NOT NULL,
            browser_chrome INTEGER NOT NULL default 0,
            browser_firefox INTEGER NOT NULL default 0,
            browser_edge INTEGER NOT NULL default 0,
            browser_safari INTEGER NOT NULL default 0,
            browser_opera INTEGER NOT NULL default 0,
            browser_sonstige INTEGER NOT NULL default 0,
            os_win INTEGER NOT NULL default 0,
            os_mac INTEGER NOT NULL default 0,
            os_linux INTEGER NOT NULL default 0,
            lastClick INTEGER NOT NULL default 0,
            clicks INTEGER NOT NULL default 0,
            timestamp INTEGER,
            expireDate INTEGER
        )
    `);
});
```

Listing 2.1: Die Statistiken werden initial auf 0 gesetzt, der Timestamp ist der Datumswert der Erstellung des Eintrags

Speichern des Links in der Datenbank

Nun geht es darum, dass wir den Link auch entsprechend mit allen Informationen in der Datenbank speichern. In Lst. 2.2 sieht man dazu die entsprechende Logik. In Zeile 2 und 3 werden die jeweiligen Zeitstempel gesetzt. `currentTime` setzt dabei den Zeitstempel beim erstellen des Link. Das `expireDate` haben wir anschließend auf 90 Tage gesetzt. Die Formel »`currentTime + 90 * 24 * 60 * 60 * 1000`« berechnet dabei genau diesen Zeitraum.

Danach werden in der Zeile 6 des Codes die Daten für `longURL`, `shortURL`, `passwd` und der bereits erwähnte Zeitstempel sowie das Verfallsdatum des Links in die Datenbank geschrieben. Die Daten für `longURL`, `shortURL` & `passwd` werden uns dabei von der API übermittelt. Wie wir in Lst. 2.1 ja schon gesehen haben, werden alle anderen Werte, welche hier nicht aufgezählt sind, mit `0` gesetzt.

Sollte es ein Problem beim Speichern der Daten in die Datenbank geben, so wird dieser Fehler in den Zeilen 10 bis 12 ausgegeben. Wenn hingegen kein Fehler auftritt, so wird die zuletzt erstellte ID für den Eintrag zurückgegeben.

```
1  const saveURL = async (result) => {
2      const currentTime = Date.now();
3      const expireDate = currentTime + 90 * 24 * 60 * 60 * 1000;
4      return new Promise((resolve, reject) => {
5          db.run(
6              'INSERT INTO url (longURL, shortURL, passwd, timestamp,
       expireDate) VALUES (?, ?, ?, ?, ?)',
7              [result.longUrl, result.shortUrl, result.passwd,
       currentTime, expireDate],
8              function (err) {
9                  if (err) {
10                     reject(err);
11                 } else {
12                     resolve(this.lastID);
13                 }
14             }
15         );
16     });
17 };
```

Listing 2.2: Speichern der URL in der Datenbank

Statistiken auslesen und an die API übergeben

Damit die entsprechenden Daten aus der Datenbank über die `/api/stats`-Route abgerufen werden können, haben wir die Logik in Lst. 2.3 geschrieben. Diese ruft in Zeile 3 alle Informationen zu einem Kurzlink auf. Mit »`'SELECT * FROM url WHERE shortURL = ?', [shortURL]`« übergeben wir der Datenbank den entsprechenden Link. Die Datenbank gibt uns anschließend alle Infos zurück, die zu dem gespeicherten Link vorhanden sind.

Sollte es einen Fehler geben, so wird dieser wieder ausgeworfen. Dies passiert in Zeile 4 & 5. Sollte es hingegen keinen Fehler geben, greift der `else` Zweig und gibt uns die entsprechende Spalte der Tabelle aus.

Update der Statistikdaten in der Datenbank mit den Infos von der API

Um die Daten entsprechend zu aktualisieren, haben wir die Logik in Lst. 2.4 geschrieben. Mit dem »`Update`« Befehl in Zeile 4 aktualisieren wir in unserer Datenbank die Daten, die wir von der API bekommen. Dabei werden die jeweiligen Statistiken für die aufgezählten Browser (Zeilen 8 bis 13), das entsprechende Betriebssystem (Zeilen 14 bis 16) sowie der letzte Klick (Zeile 17) und die generellen Clicks (Zeile 18).

Auch hier haben wir wieder das Fehlerhandling so gebaut, dass er bei Fehler die entsprechende Fehlermeldung auswirft und bei keiner Fehlermeldung die Daten in die Datenbank schreibt (Zeile 21 bis 25).

```
const getStats = async (shortURL) => {
    return new Promise((resolve, reject) => {
        db.get('SELECT * FROM url WHERE shortURL = ?', [shortURL], (
    err, row) => {
            if (err) {
                reject(err);
            } else {
                resolve(row);
            }
        });
    });
};
```

Listing 2.3: Daten werden von der Datenbank anhand des Kurzlinks abgerufen

2.3 Statistik und Analyse

Für die Statistik nutzen wir die Express.js-Middleware `express-useragent`[3], welches die User-Agent-Kennungen zusammenfasst, so dass man nicht für jede Browser-Version einzeln checken muss. Diese Daten werden in der Datenbank kumuliert und bei Anfrage entsprechend über den API-Endpunkt `/api/stats/{shortUrl}` an den Client gesendet. Die Statistiken sind mit einem Passwort geschützt, dass dem Benutzer bei Erstellung der Kurz-URL angezeigt wird.

2.4 API

Die API bietet verschiedene Endpunkte für den Zugriff. So wird zur Erstellung eines Kurz-Links der http-POST-Endpunkt »/« genutzt, zur Weiterleitung (Abfrage des Kurz-Links) wird der Endpunkt »`/{shorturl}`« verwendet. Und zur Abfrage der Statistiken dient der POST-Endpunkt »`/api/stats/{shortUrl}`«.

2.5 URL-Validierung

Zur Validierung der URL nutzen wir das Paket `url-http`[4] mit einer kleinen Wrapper-Funktion (siehe Lst. 2.5), wobei die URL inklusive Schema (`http://` bzw. `https://`) erwartet wird.

2.6 URL-Kürzung

Die eigentliche URL-Kürzung entsteht durch einen Datenbank-Eintrag, der die generierte Kurz-URL (ID) mit der langen URL verknüpft und zusätzlich die statistischen Daten zu jedem Kurzlink speichert.

[3] https://www.npmjs.com/package/express-useragent
[4] https://www.npmjs.com/package/url-http

2.6.1 Algorithmus

Wir haben uns für die Verwendung eines 58-stelligen Alphabets (Base-58, siehe Lst. 3.1, Zeile 1) für die Generierung der ID entschieden. Hierzu werden von einem Base62-Alphabet (`[0–9A–Za–z]`) die Zeichen »0« (Null), »I« (Großbuchstabe i), »O« (Großbuchstabe o) und »l« (Kleinbuchstabe L) entfernt, da diese – je nach Schriftart – verwechselt werden können. Hierdurch wird zwar die Anzahl der möglichen IDs eingeschränkt, aber $58^7 = 2\,207\,984\,167\,552$ Möglichkeiten sollten diese immer noch ausreichen.

Die eigentliche Kurz-URL ist dann ein siebenstelliger base58-kodierter String aus einer Kombination aus Teilen des Unix-Zeitstempels (Lst. 2.6 Zeilen 1–3) und einer zweistelligen Zufallszahl (Zeile 4). Zum einfacheren Verbinden beider Zahlen werden diese zunächst als String generiert und erst bei Übergabe an die Funktion `base58()` als Integer umgewandelt (ebenfalls Zeile 4 in Lst. 2.6).

2.7 Passwort für Link-Bearbeitung und Statistik

Das Passwort für die generierte Kurz-URL lassen wir uns als zufällige, zwölfstellige alphanumerische Zeichenkette durch das Modul »generate-password«[5] erstellen.

[5] https://www.npmjs.com/package/generate-password

```
const writeStats = (shortURL, stats) => {
    return new Promise((resolve, reject) => {
        db.run(
            'UPDATE url SET browser_chrome = ?, browser_firefox = ?,
   browser_edge = ?, browser_safari = ?, browser_opera = ?,
   browser_sonstige = ?, os_win = ?, os_mac = ?, os_linux = ?,
   lastClick = ?, clicks = ? WHERE shortURL = ?',
            [
                stats.Browser.Chrome,
                stats.Browser.Firefox,
                stats.Browser.Edge,
                stats.Browser.Safari,
                stats.Browser.Opera,
                stats.Browser.Sonstige,
                stats.OS.Windows,
                stats.OS.MacOs,
                stats.OS.Linux,
                stats.lastClick,
                stats.clicks,
                shortURL,
            ],
            (err) => {
                if (err) {
                    reject(err);
                } else {
                    resolve();
                }
            }
        );
    });
};
```

Listing 2.4: Die Daten werden von der API an uns übermittelt und wir speichern die Daten in der Datenbank beim entsprechenden Kurzlink ab

```
const isValidUrl = (url) => !!httpUrl(url);
```

Listing 2.5: Wrapper-Funktion zur URL-Validierung

```
let date = Date.now()
    .toString()
    .slice(1,-2);
const shortUrl = base58(parseInt(getRnd() + date));
```

Listing 2.6: Aufbau der Basis vor der base58-Konvertierung

3 Beschreibung der Lösung

Das Backend ist eine RESTful API, welche von den Frontend-Komponenten gezielt abgefragt wird.

3.1 Backend

Wie bereits in Kapitel 2.6.1 beschrieben, haben wir uns für eine `base58`-Kodierung der Kurz-URLs entschieden. Hierzu wurde das entsprechende Alphabet definiert (siehe Lst. 3.1, Zeile 1). Nach einem Check, ob es eine Ganzzahl < 0 ist (Zeilen 4–8), wird innerhalb der `while`-Schleife (ab Zeile 11, Lst. 3.1) dann nach dem Restwertverfahren die Eingabezahl konvertiert.

Zwar wäre diese Umrechnung auch wieder umkehrbar (bijektiv), aber da es eine generierte Zahl ist, die keine Verbindung mit der zu kürzenden URL hat, verzichten wir darauf und nehmen zur Identifikation die `base58`-kodierte Zahl als Schlüssel.

Aufgrund der in Lst. 2.6 gezeigten Generierung der Ausgangszahl, kann es unter seltenen Umständen dazu kommen, dass bei mehreren gleichzeitigen Zugriffen auch die gleiche Zufallszahl generiert wird. Dazu müsste allerdings in der gleichen Zehntelsekunde auch noch die gleiche Zufallszahl durch `getRnd()` erzeugt werden. Auch wenn dies nicht komplett ausgeschlossen werden kann, haben wir uns dazu entschieden, dies zu vernachlässigen. Diesen Fall kann man – in einer späteren Erweiterung (siehe auch Kapitel 4) – auch durch eine Datenbankabfrage und evtl. Neugenerierung der Zufalls-ID umgehen.

3.1.1 Schnittstellen der Komponente

Es gibt drei Schnittstellen-Pfade der API:

/ POST-Endpunkt, um eine neue Kurz-URL zu generieren

/{shortUrl} Endpunkt für alle HTTP-Methoden, um eine Kurz-URL abzurufen. Der Abrufende wird automatisch, mittels http-Status 307, an die Zieladresse weitergeleitet.

/api/stats/ POST-Endpunkt zum Abruf der Statistik zu der angegebenen Kurz-URL. Das Passwort und die Kurz-URL müssen im Request-Body als Javascript Object Notation (JSON)-Objekt gesendet werden.

Für eine genaue (technische) Dokumentation haben wir die API-Dokumentation im OpenAPI-Format[1] erstellt, welche über das Backend[2] abrufbar ist.

[1] https://swagger.io/specification/
[2] Backend:Port/api/doc/api

```
const CHARS58 = '123456789
   ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz';

function encode(num) {
    if (isNaN(num) || num % 1 !== 0 || num < 0) {
        // Keine Zahl, oder Float, oder < 0
        return false;
    } else if (num === 0) {
        return '0';
    } else {
        let id = '';
        while (num > 0) {
        id = CHARS58[(num % 58)] + id;
        num = Math.floor(num / 58);
        }
        return id;
    }
}
```

Listing 3.1: base58-Konvertierung

3.1.2 Weiterleitung

Wir leiten mit dem HTTP-Status-Code »307« weiter, so dass sichergestellt ist, dass der Client seine Anfrage mit der gleichen HTTP-Methode an den Server stellt, auf den die Short-URL referenziert.

3.1.3 Datenstrukturen

Wir haben uns für eine recht einfache Struktur entschieden, siehe Lst. 3.2. Ursprünglich geplant war, wie bereits in Kapitel 2.2 geschrieben, die Daten in einer Document-Database (Google Firebase Firestorm[3]) abzulegen, in der sie hätten im JSON-Format geschrieben und gelesen werden können. Allerdings musste das Datenbank-Team – wegen Problemen bei der Umsetzung – am letzten Tag dann doch noch auf SQlite3[4] wechseln, so dass die untenstehende Struktur (siehe Lst. 3.2) nicht mehr eingehalten werden konnte. Damit das Frontend-Team nicht auch noch alles ändern musste, bildet die API die neue Struktur auf die alte ab.

3.2 Frontend

Unser Frontend, das in React[5] entwickelt wurde, besteht aus mehreren Schlüsselkomponenten, die nahtlos miteinander interagieren, um die gewünschten Funktionen unserer

[3] https://firebase.google.com/
[4] https://www.sqlite.org/index.html
[5] https://react.dev/

```
{
    "clicks": 0,
    "lastClick": timestamp,
    "OS": {
        "Linux": 0,
        "Windows": 0,
        "MacOs": 0,
    },
    "Browser": {
        "Chrome": 0,
        "Edge": 0,
        "Firefox": 0,
        "Opera": 0,
        "Safari": 0,
        "Sonstige": 0,
    },
    "createDate": timestamp,
    "expireDate": timestamp,
    "longURL": "http://ex.amp.le/",
    "shortURL": "aB1Cd2e",
    "Password": "password"
}
```

Listing 3.2: Aufbau der Datenstruktur für jede Kurz-Url

Anwendung bereitzustellen.

3.2.1 Startseite

App.js ist die Hauptkomponente unserer Frontend-Anwendung, sie bildet die Benutzeroberfläche ab. Hier haben Benutzer die Möglichkeit, URLs einzugeben, die dann an den Server gesendet werden. Die Antwort des Servers, bestehend aus der gekürzten URL und einem Passwort zur Identifikation, wird übersichtlich angezeigt. Darüber hinaus bietet diese Komponente die Möglichkeit, zur Statistikseite zu navigieren.

BasicModal.js fungiert als eine Modal-Komponente, die eine Hilfefunktion für unsere Anwendung bereitstellt. Sie wird aktiviert, wenn Benutzer auf das Fragezeichen-Symbol klicken, um zusätzliche Informationen zu erhalten.

Password.js ist verantwortlich für die Generierung eines Passwortfelds, das Benutzer in die Zwischenablage kopieren können. Zusätzlich zeigt es eine benutzerfreundliche Snackbar-Nachricht zur Bestätigung des Kopiervorgangs an.

ShortURL.js hat die Aufgabe, die gekürzte URL anzuzeigen und Benutzern die Möglichkeit zu geben, diese bequem in die Zwischenablage zu kopieren. Dabei nutzt sie das Basis-URL-System und Material-UI-Elemente, um die Darstellung und Interaktion benutzerfreundlich zu gestalten (siehe Abb. 3.1).

PWModal.js ist eine weitere Modal-Komponente, die eine ausführliche Erklärung zur Verwendung des Passworts zur Identifikation bietet. Diese Komponente wird ebenfalls durch Klicken auf das Fragezeichen-Symbol aktiviert.

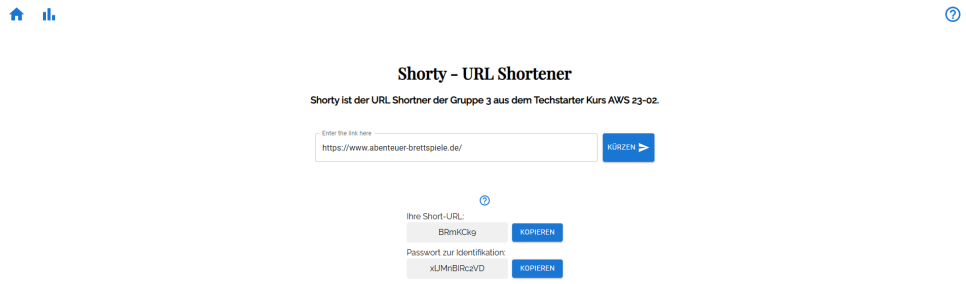


Abbildung 3.1: Anzeige der generierten Kurz-Url und des Passworts

Die drei letzten Komponenten werden erst sichtbar, nachdem eine Kurz-URL generiert wurde.

Abschließend ermöglicht `Snackbar.js` das Anzeigen von Snackbar-Nachrichten zur Bestätigung des erfolgreichen Kopiervorgangs von Text in die Zwischenablage. Dabei kommt das `navigator.clipboard-API` (Document Object Model (DOM)) in Kombination mit Material-UI zum Einsatz.

PWDModal.js ist eine Modal-Komponente, die Benutzern das Eingeben einer Kurz-URL und eines Passworts ermöglicht. Die eingegebenen Daten werden über eine API-Anfrage überprüft, und bei erfolgreicher Überprüfung wird der Benutzer zur Statistikseite weitergeleitet, um detaillierte Statistiken anzuzeigen.

3.2.2 Statistikseite

Auf unserer Statistikseite (siehe Abb. 3.2) werden verschiedene Daten visualisiert, darunter kleine Balken- und Kreisdiagramme, um dem Benutzer ein umfassendes Bild der URL-Statistiken zu vermitteln. Hier sind die Schlüsselkomponenten, die zur Umsetzung dieser Funktionen verwendet werden:

Stats.js ist die zentrale Seite für die Anzeige von Statistikdaten. Diese Seite empfängt Daten von der Hauptseite und stellt sie visuell ansprechend dar. Benutzer haben auch die Möglichkeit, von hier aus zur Startseite zurückzukehren.

Chartpie.js und TinyBar.js sind die zwei optischen Schlüsselkomponenten der Statistikseite. `Chartpie.js` erstellt ein Kreisdiagramm, das die Betriebssystemnutzung in Prozenten anzeigt, während `TinyBar.js` ein kompaktes Balkendiagramm für die Anzahl der Klicks in verschiedenen Browsern darstellt. Beide verwenden die `Recharts`-Bibliothek für die Diagrammerstellung und spielen eine wichtige Rolle bei der Visualisierung von Statistikdaten.

Diese Komponenten ermöglichen es dem Benutzer auf der Statistikseite alle Informationen auf einen Blick und leicht verständlich abzulesen.

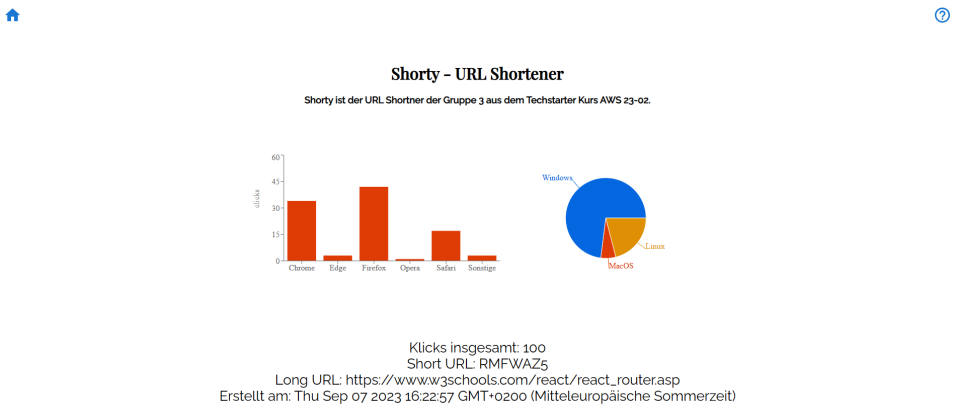


Abbildung 3.2: Statistiken zu einer generierten Kurz-Url

3.3 Übersicht Verzeichnisse und Dateien

Auf Top-Level wurde für jede Komponente ein Verzeichnis erstellt. Innerhalb dieser wurde von den jeweiligen Teams eine eigene Struktur etabliert.

3.3.1 Backend

Im Backend-Hauptverzeichnis `/server` wurde für die API-Endpunkte das Verzeichnis `api/` erstellt, welches wiederum die Unterverzeichnisse `doc/` und `stats/` enthält, welche die jeweiligen Pfade der Endpunkte abbilden. Die Controller der Endpunkte befinden sich unter `/server/controller/`.

Im Verzeichnis `src/` im `/server`-Verzeichnis befinden sich JS-Dateien, die (Hilfs-)Funktionen für die Endpunkte bereitstellen, z. B. die `base58()`-Kodierung. Unterhalb des `src`-Verzeichnisses liegen die Datenbank-Funktionen in ihrem `db`-Verzeichnis.

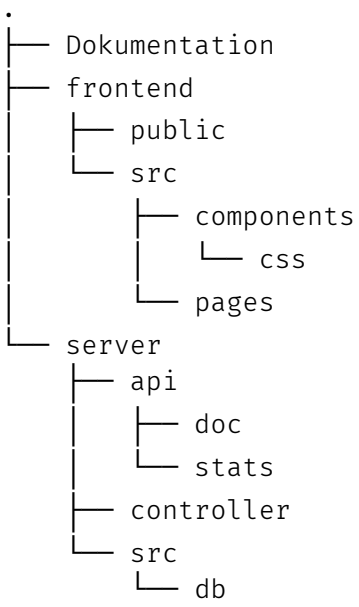
Der Einstiegspunkt für das Backend ist `/server/index.js`.

3.3.2 Dokumentation

Da die Dokumentation das komplette Projekt, mit allen (Sub-)Komponenten umfasst, hat diese ihr eigenes (projektbezogenes) Top-Level-Verzeichnis: `/Dokumentation`. Hier befinden sich sowohl die `.tex`-Dateien, als auch die `.yaml`-Datei der OpenAPI-Dokumentation.

3.3.3 Struktur

Daraus ergibt sich dann folgende Verzeichnisstruktur:


.
Dokumentation
frontend
public
src
components
css
pages
server
api
doc
stats
controller
src
db

4 Möglichkeiten zur späteren Anpassung und Weiterentwicklung

Es bestehen diverse Möglichkeiten, das Projekt weiterzuentwickeln:

- Die Statistiken können erweitert werden, sodass man auch nach mobilen (Smartphone, Tablet) oder stationären (Desktop/Laptop, Smart-TV) auswerten kann.
- Auch können die API-Endpunkte erweitert werden, um z. B. nur Werte zu einer bestimmten Statistik abgerufen werden können.
- Es kann auch dahingehend erweitert werden, dass sich Benutzer registrieren und anmelden können. Dann kann auch für den eingeloggten Nutzer ein Verlauf der generierten Kurz-URLs angezeigt werden und die jeweiligen Statistiken mit einem einfachen Button aufrufbar gemacht werden.
- Es kann ein DELETE-Endpunkt implementiert werden, damit ein Benutzer seinen generierten Link vorzeitig löschen kann.
- Die generierten Links sollten auch nach einer voreingestellten Zeit automatisch gelöscht werden. Diese Zeit kann bis zu einer Obergrenze von Nutzer festgelegt werden. Das Löschen selbst kann beispielsweise durch einen Endpunkt erfolgen, der von außen mittels `cron` und `curl` angesprochen wird. Das Feld `expireDate` ist schon in jedem Datensatz enthalten und wird standardmäßig auf Erstellungszeit + 90 Tage gesetzt.
- Es kann (sollte?) eine Sicherheitsabfrage eingebaut werden, die das Erzeugen – und damit das Überschreiben – identischer Kurz-IDs unterbindet.
- Zum Einsatz »in production« sollten auf jeden Fall noch die Benutzereingaben auf Vollständigkeit und Schadcode überprüft werden. Der Anfang hierzu ist mit der Funktion »`isValidBase58` bereits gemacht.
- Ebenfalls für den produktiven Einsatz können entsprechende Log-Meldungen eingepflegt werden. Hierzu kann auch die Logging-Bibliothek geändert werden, z. B. zu »Winston«[1].

[1] https://github.com/winstonjs/winston

5 Gewonnene Erkenntnisse für zukünftige Projekte

Wir haben einiges zu Zusammenarbeit im Team und zum Zusammenspiel verschiedener Komponenten gelernt, woraus jeder von uns seine Rückschlüsse für kommende Projekte ziehen wird.

Liste der Abkürzungen

API Application Programming Interface (Programmierschnittstelle)

DOM Document Object Model

JS Javascript

JSON Javascript Object Notation

OS Operating System (Betriebssystem)

URL Uniform Resource Locator

Abbildungsverzeichnis

Listings