09 JWT Tokens

Multimedia Engineering (PS siehe 36609b)

M.Sc. Nils Hellwig
Lehrstuhl für Medieninformatik
FAKULTÄT FÜR INFORMATIK UND DATA SCIENCE

Universität Regensburg

User Management mit MongoDB/Mongoose und JWT

Was ist User Management?

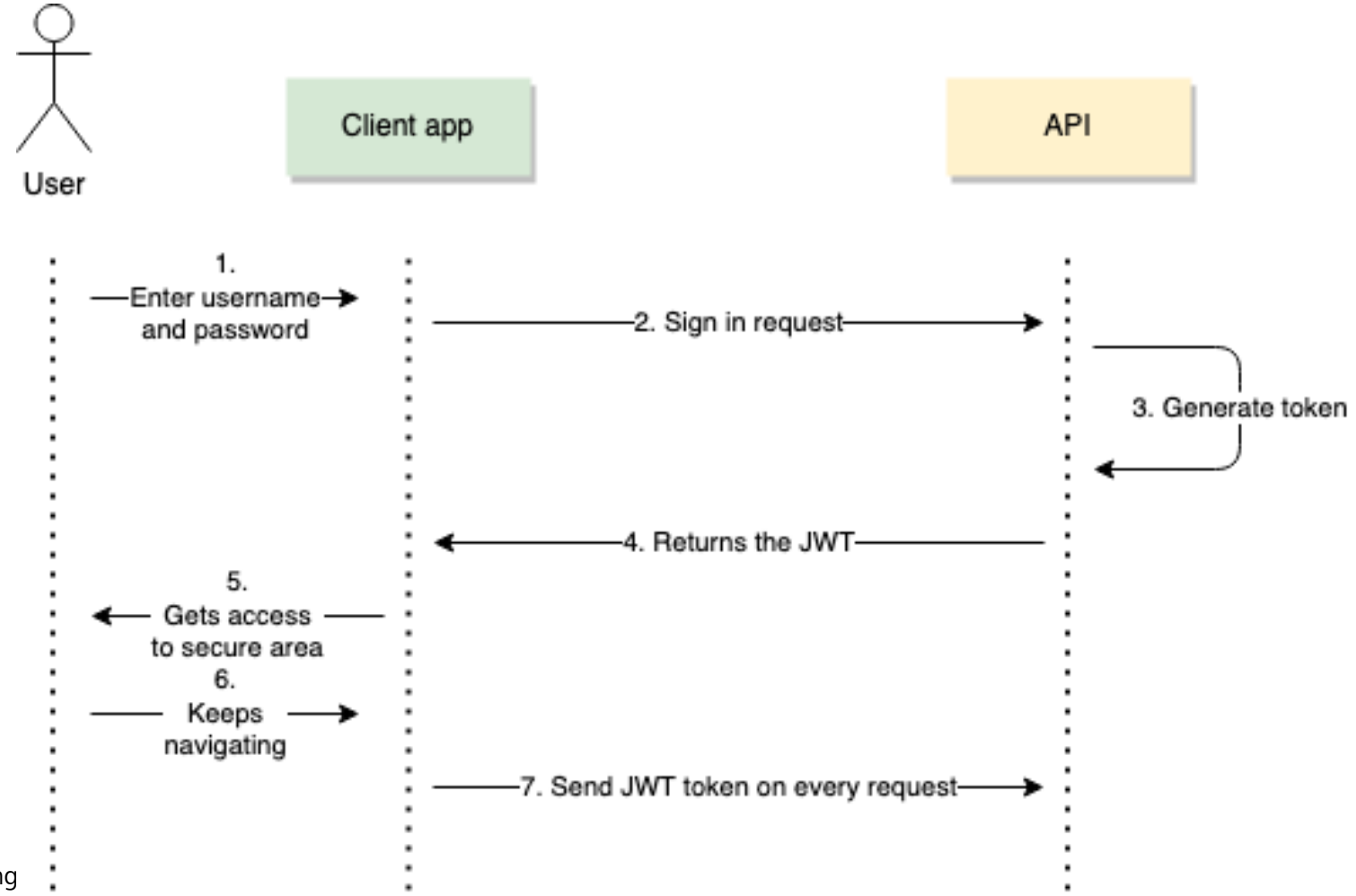
- Verwaltung von Benutzerkonten in einer Webanwendung.
- Funktionen wie Registrierung, Login, Passwortverwaltung und Authentifizierung.
- Ziel: Nur autorisierte Nutzer Zugriff auf bestimmte Ressourcen haben.

User Management mit MongoDB/Mongoose und JWT

- Ein **JWT (JSON Web Token):** Benutzeridentität zwischen Client und Server kommunizieren.
- Benutzeridentität wird in einem **digital signierten Token** gespeichert und zwischen Client und Server übertragen werden kann.
- Beispiel:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ik1heGEgTXVzdGVybWFubiIsInJvbGUiOiJhZG1pbiIsImlhdCI6MTY1MzM5NTUyMn0.
SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

User Management mit MongoDB/Mongoose und JWT



User Management mit MongoDB/Mongoose und JWT

Header: Enthält Metadaten wie Algorithmus (HS256) mit dem der Token signiert wird und Typ (JWT).
Payload: Beinhaltet Benutzerdaten (z. B. Benutzer-ID, Rollen).
Signatur: Verifiziert die Integrität der Daten mit einem SECRET_KEY.

Header und Payload sind codiert (Base64), können einfach z.B. mit Online-Tool ausgelesen werden.

(Base64 ist eine Methode zur Kodierung von Daten in ein ASCII-String-Format)

User Management mit MongoDB/Mongoose und JWT

- Der Server nimmt:
  1. den **Header** (Teil 1 des JWT-Tokens)
  2. den **Payload** (Teil 2 des JWT-Tokens)
  3. und einen „**geheimen Schlüssel“** (String, möglichst anspruchsvoll), den nur der Server kennt.
- Daraus wird eine **Signatur** (Teil 3 des JWT Tokens) mit einer Rechenregel wie „HS256“ berechnet.
- Zusammengefasst: Die **Signatur** ist ein digitales Siegel, das der Server ursprünglich aus Header, Payload und dem geheimen Schlüssel berechnet hat.
- Angenommen, man ändert den Payload, dann würde eine neue, nicht gültige Signatur erstellt werden

HS256 ist ein kryptografischer Algorithmus, der bei JWTs verwendet wird, um eine **digitale Signatur** zu erstellen.

„HS“: **HMAC mit SHA-256**:

HMAC ist eine Methode, die eine Nachricht mit einem geheimen Schlüssel kombiniert, um eine sichere Prüfsumme (Signatur) zu erzeugen.
SHA-256 ist eine Hash-Funktion, die aus beliebigen Daten einen eindeutigen, festen *256-Bit-Wert* berechnet.

User Management mit MongoDB/Mongoose und JWT: Secret Key

Wie bereits in der vorherigen Folie erwähnt, ist der `SECRET_KEY` ist ein **geheimer Schlüssel**, der verwendet wird, um JWTs (JSON Web Tokens) zu signieren und zu validieren.

Signierung:

```javascript
jwt.sign({ userId: 123 }, SECRET_KEY);
```

Verifizierung:

```javascript
jwt.verify(token, SECRET_KEY);
```

Header Default (3. Argument von `.sign()`): `{ "alg": "HS256", "typ": "JWT"}`

Wo speichert man einen Secret Key? Environment Variables!

- **Environment Variables (Umgebungsvariablen)**: Sensible oder konfigurierbare Daten, die außerhalb des Programmcodes in der Laufzeitumgebung definiert sind.
- Diese Variablen werden oftmals in einer Datei `.env` gespeichert.
- **Wichtig:** Veröffentlicht man sein Repository, sollte die `.env` Datei nicht in dem Repository enthalten sein.

Beispiel:

```plaintext
SECRET_KEY=MeinGeheimerSchlüssel
DATABASE_URL=mongodb://localhost/myDatabase
```

User Management mit MongoDB/Mongoose und JWT: Wo speichert man einen Secret Key? Environment Variables!

- Variablen aus der .env können mithilfe von dotenv in Node.js geladen werden:

```bash
npm install dotenv
```

- Integration im Code:

```javascript
require('dotenv').config();
const secretKey = process.env.SECRET_KEY;
console.log(secretKey);
```

Die Methode .config() liest die **.env-Datei** und lädt ihre Inhalte als **Environment-Variablen** in die Laufzeitumgebung der Anwendung.

process ist ein globales **Node.js-Objekt**, das Informationen und Funktionen zur Steuerung des aktuellen Node.js-Prozesses bereitstellt.