	**Politechnika Opolska** **Wydział Elektrotechniki, Automatyki i Informatyki** **Katedra Informatyki**
Rok akademicki	2023/2024
Przedmiot	Bazy danych w praktyce
Prowadzący zajęcia	Dr hab. inż. Mariusz Rząsa
Nr grupy	1

Bezpieczeństwo API REST

Nazwisko i imię	Nr indeksu
Dominik Boguszewski	**98784**

Uwagi

1. Wstęp

Architektura Representational State Transfer (REST) to jeden z najpopularniejszych i powszechnie stosowanych modeli projektowania interfejsów programistycznych (API). Zasady REST, opracowane przez Roya Fieldinga, stanowią fundamentalną metodologię dla komunikacji pomiędzy systemami komputerowymi, umożliwiając rozwój efektywnych, elastycznych i łatwo zrozumiałych aplikacji.

API REST to zestaw zasad i protokołów, które służą do projektowania i budowy interfejsów programistycznych. Nazwa "REST" oznacza "Representational State Transfer" i odnosi się do idei przenoszenia stanu zasobów między klientem a serwerem poprzez reprezentacje. W praktyce oznacza to, że zasoby, takie jak dane czy usługi, są reprezentowane w formie zrozumiałej dla klienta, który może następnie wykorzystać te reprezentacje w celu manipulacji stanem zasobów.

Kluczowe zasady REST:

- **Reprezentacja zasobów**: zasoby (np. dane, obiekty, usługi) są reprezentowane w formie zrozumiałej dla klienta, zazwyczaj w formacie JSON lub XML,
- **Jednolity interfejs**: interfejs API powinien być jednolity, co oznacza, że zasoby są identyfikowane jednoznacznie i dostępne za pomocą jednolitego interfejsu, niezależnie od ich rodzaju,
- **Bezstanowość**: każde żądanie od klienta do serwera zawiera wszystkie informacje potrzebne do zrozumienia i przetworzenia tego żądania. Serwer nie przechowuje żadnych informacji o stanie klienta między żądaniami,
- **Możliwość buforowania**: odpowiedzi serwera mogą być buforowane, co zwiększa wydajność poprzez unikanie ponownego pobierania tych samych danych,
- **Warstwowość**: systemy REST mogą być warstwowo zorganizowane, co oznacza, że każda warstwa ma określone zadania i nie musi być świadoma struktury pozostałych warstw,

2. Dlaczego należy zabezpieczać API REST?

Bezpieczeństwo API REST jest krytycznym aspektem w budowaniu nie tylko stabilnych, ale przede wszystkim bezpiecznych aplikacji. W przypadku, gdy API obsługuje dane użytkowników, ich prywatność staje się priorytetem. Bezpieczeństwo jest kluczowe dla zapewnienia, że prywatne informacje nie padają w nieautoryzowane ręce, chroniąc wrażliwe dane przed niepożądanym dostępem.

API REST jest narażone na różnorodne ataki, takie jak ataki SQL injection, ataki typu cross-site scripting (XSS) czy ataki CSRF (Cross-Site Request Forgery). Zabezpieczenia pomagają zapobiegać takim zagrożeniom, utrzymując integralność systemu.

Bezpieczeństwo API wymaga pewności, że tylko uprawnione osoby mają dostęp do zasobów. Mechanizmy autentykacji sprawdzają tożsamość klientów, podczas gdy autoryzacja definiuje, jakie działania dany użytkownik może podjąć. To kluczowe dla ochrony wrażliwych danych przed nieautoryzowanym dostępem.

Bezpieczeństwo API pomaga w utrzymaniu integralności danych, uniemożliwiając nieautoryzowaną modyfikację zasobów. To szczególnie ważne w kontekście zapewnienia dokładności danych i uniknięcia sytuacji, w których zasoby zostają naruszone.

3. Projekt

By wykonać projekt API REST, należy najpierw zainstalować Node.js (https://nodejs.org/en) . Po pomyślnym zainstalowaniu Node.js, trzeba utworzyć folder, gdzie będzie fizycznie znajdować się nasze API.

```
D:\>mkdir api-projekt

D:\>cd api-projekt\
```

Utworzenie folderu dla projektu

Następnie trzeba zainicjalizować projekt Node.js:

```
D:\api-projekt>npm init -y
Wrote to D:\api-projekt\package.json:

{
  "name": "api-projekt",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

Zainicjowanie projektu Node.js

Kiedy projekt zostanie zainicjowany, należy zainstalować w naszym projekcie express oraz jsonwebtoken.

Express jest minimalistycznym i elastycznym frameworkiem dla aplikacji internetowych opartych na Node.js. Został zaprojektowany, aby ułatwić tworzenie interfejsów API (API REST) oraz aplikacji stron internetowych. Express dostarcza zestaw funkcji i narzędzi, które znacznie upraszczają proces obsługi żądań HTTP, zarządzania trasami, oraz renderowania widoków.

Jsonwebtoken umożliwia proste tworzenie tokenów JWT zdefiniowanych przez programistę, zawierających informacje o użytkowniku lub innych informacjach potrzebnych do bezpiecznej autentykacji.

```
D:\api-projekt>npm install express jsonwebtoken

added 78 packages, and audited 79 packages in 2s

11 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Zainstalowanie bibliotek do projektu

Kolejnym krokiem jest utworzenie pliku JavaScript o dowolnej nazwie (w tym przypadku index.js) oraz otwarcie go w edytorze tekstu według uznania. W nowo utworzonym pliku na samym początku powinny się znaleźć importy zainstalowanych bibliotek.

```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  const jwt = require('jsonwebtoken');
```

Importy bibliotek

Następnie trzeba zainicjalizować aplikację Express:

```
5  const app = express();
6  const port = 3000;
7  const secretKey = 'tajnyKlucz123';
8
9  app.use(bodyParser.json());
```

Inicjalizacja aplikacji Express

Aby udało się włączyć aplikację należy rozpocząć nasłuchiwanie na określonym porcie:

```
40  app.listen(port, () => {
41      console.log(`Serwer nasłuchuje na porcie http://localhost:${port}`);
42  });
```

Rozpoczęcie nasłuchiwania

Następnie podczas obsługi żądania logowania „/login”, generowany jest token JWT przy użyciu funkcji jwt.sign.

```
26  app.post('/login', (req, res) => {
27      const { username, password } = req.body;
28      // Tutaj powinno odbyć się czy login oraz hasło
29      // są prawidłowe np. z bazą danych
30      const token = jwt.sign({ username }, secretKey);
31      res.json({ token });
32  });
```

Tworzenie tokenu JWT

Kolejnym krokiem jest funkcja middleware, która jest używana jako pośrednik dla zabezpieczonego endpointu „/secure”. Sprawdza ona, czy żądanie zawiera poprawny token w nagłówku „Authorization”.

```
11  function authenticateToken(req, res, next) {
12      const token = req.header('Authorization');
13      if (!token) return res.status(401).send('Brak tokena.');
14
15      jwt.verify(token, secretKey, (err, user) => {
16          if (err) return res.status(403).send('Błąd autentykacji.');
17          req.user = user;
18          next();
19      });
20  }
```

Middleware dla zabezpieczonego endpointu /secure

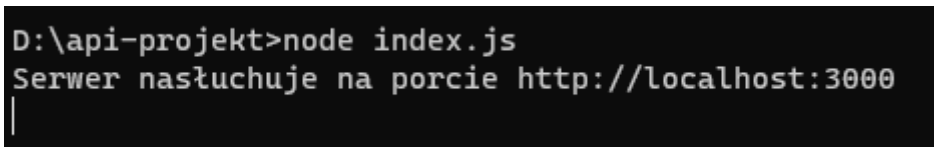
Dostęp do kodu poniżej jest zabezpieczony, ponieważ wymaga poprawnego tokenu JWT.

```
22  app.get('/secure', authenticateToken, (req, res) => {
23      res.json({ message: 'Dostęp do zabezpieczonego zasobu udzielony.' });
24  });
```

Zabezpieczony endpoint funkcją authenticateToken

4. Testowanie API

By przetestować nowo utworzone API należy otworzyć konto terminala, oraz włączyć nowo utworzony plik index.js komendą: node <nazwa_pliku>.js



Włączenie utworzonego API

Następnie trzeba otworzyć nowe okno terminala oraz wpisać komendę logowania z odpowiednimi danymi:

curl -X POST -H "Content-Type: application/json" -d '{"username": "<username>", "password": "<password>"}' http://localhost:<port_nasłuchu>/login

Poprawnie wykonana komenda

Poprawnie wykonana komenda powinna zwrócić token, który należy podać w komendzie:

Curl -H "Authorization: <token>" http://localhost:3000/secure

Udzielenie dostępu do endpointu /secure

5. Zadanie

Na zadanie samodzielnie należy sprawdzić co zwróci serwer w wypadku, gdy będziemy chcieli uzyskać do endpointu secure, bez uprzedniego utworzenia tokenu.