Avancerad JavaScript

Laboration 3

Laboration 3

- Deadline för labben är fredag 12:e april
- Den här gången ska vi implementera användarregistrering och autentisering med JSON Web Tokens (JWT)
- Vi ska även skapa en enkel Todo-applikation där inloggade användare har en egen att-göra-lista

JSON Web Tokens (JWT)

- JSON Web Tokens (JWT) är en öppen standard för att skicka säkra meddelanden på nätet
- JWT finns implementerat i de flesta moderna programmeringsspråk
- Med JWT kan en webbserver skicka data till en klient som är svår att förfalska
- Det används ofta vid autentisering på webben
- För att skapa en JWT-token behövs en hemlig nyckel. Den hemliga nyckeln används även för att verifiera om en token är giltig
- En stor fördel med JWT är att servern inte behöver hålla koll på vilka klienter som är inloggade utan man behöver bara kontrollera om en token är giltig

JWT i JavaScript

- I JavaScript kan vi använda modulen **jsonwebtoken** för att skapa en JWT-token
- För att skapa en token använder vi funktion **sign(payload, secret)**

```
import jwt from 'jsonwebtoken';
const token = jwt.sign({ a: 10 }, 'secret');
console.log(token);
```

- Det går även att göra så att token bara är giltig under en viss tid.
- De tokens som ni kommer få i labben slutar fungera efter en tid

JWT i JavaScript

- För att verifiera att en token är giltig används funktion **verify(token, secret)**
- Funktionen returnerar även själva objektet
- Om en token är ogiltig eller om den är för gammal kommer funktionen kasta ett fel

```
const decoded = jwt.verify(token, secret);
```

- Informationen i en JWT-token är inte krypterad och om vi inte har tillgång till den hemliga nyckeln kan vi använda funktionen **decode(token)** för att hämta ut informationen

```
const decoded = jwt.decode(token);
```

Backend

- Precis som i tidigare labbar i kursen finns en backend med ett API tillgänglig
- Adressen är http://ec2-13-53-32-89.eu-north-1.compute.amazonaws.com:3000
- Backenden har API för att registrera användare, logga in och hantera element i en personlig todo-lista
- Till skillnad från tidigare labbar delar ni inte på samma data, utan varje användare har sin egen lista

API-referens

POST /register
Används för att skapa en ny användare. En email-adress och ett lösenord behöver skickas.

Exempel

```
axios.post(API_ROOT + '/register', { email, password });
```

Lösenordet sparas inte på servern utan det kommer att hashas men använd ändå inte några personliga lösenord! Lösenord skickas i klartext eftersom servern inte använder HTTPS

API-referens

POST /auth

Används för att logga in. Om email och lösenord är giltiga kommer den att svara med en JWT-token

Exempel

```
axios.post(API_ROOT + '/auth', { email, password });
```

API-referens

GET /todos

Används för att hämta en lista av todo-element för en användare. En JWT-token måste skickas med en Authorization-header.

Exempel

```
axios.get(API_ROOT + '/todos', {
  headers: {
    Authorization: 'Bearer ' + token,
  },
});
```

API-referens

POST /todos

Används för att skapa en ny todo. Ett objekt med en nyckel “content” måste skickas.

Exempel

```
axios.post(API_ROOT + '/todos', { content: 'Water the
plants' }, options);
```

API-referens

DELETE /todos/:id

Används för att ta bort ett todo-element.

Vyer

Applikationen ska ha minst tre olika vyer

- En registreringssida med ett formulär för att registrera en ny användare
- En loginsida där användare kan logga in med email och lösenord
- En “att-göra-lista” där en inloggad användare kan hantera en lista

Alla sidor ska dela ett gemensamt sidhuvud som innehåller innehåller länkar till registreringssida, inloggningssida och en knapp för att logga ut om man redan är inloggad.

Sidhuvudet ska även visa email-adressen för den inloggade användaren.

Registreringssida

- Sidan ska innehålla ett formulär med fält för att mata in email och lösenord
- När användaren trycker på knappen ska API:et användas för att lägga till en användare
- Visa att felmeddelande om API:et svarar med ett fel

Loginsida

- Sidan ska innehålla ett formulär där användaren kan mata in email och lösenord
- När användaren trycker på knappen ska API:et användas för att hämta en JWT-token
- Spara JWT-token i localStorage så att användare fortsätter vara inloggade även om de laddar om sidan
- Visa ett felmeddelande om API:et svarar med ett fel

Att-göra-lista

- Inloggade användare ska kunna lägga till och ta bort element i en att-göra-lista
- Sidan ska innehålla ett formulär där användaren kan mata in saker att göra
- Sidan ska visa en lista av “todos” och varje element i listan ska ha en knapp som går att använda för att ta bort element
- Visa ett felmeddelande om API:et svarar med ett fel

Krav

- Applikationen ska skrivas som en SPA i React
- Ni ska implementera korrekt routing
- Användaren ska vara fortsatt inloggad även om sidan laddas om
- Sidan ska innehålla minst tre vyer
  - Registreringssida
  - Loginsida
  - Att-göra-lista
- Email-adressen i sidhuvudet ska tas från den JWT-token som skickats från servern. Spara inte email-adressen i localStorage

Tips

- Använd **react-router** för routing
- Försök dela upp applikationen i små, enkla delar
- Implementera varje sida som en liten applikation med sitt eget state
- Hjälp varandra och fråga om ni kör fast
- Lycka till!