Planned agenda for exam questions

- Når vi snakker passwords kan vi ikke bare gemme brugerens rå password nogle steder. Problemet er hvis vi bliver compromised kan folk se brugerens password og få adgang til den bruger. Derfor hasher vi passwords. Det er ikke bare nok med hashing da det stadig kan knækkes så derfor bruger vi BCryptPasswordEncoder som også tilføjer salting samt en work factor, strength. Den har 4 dele. Den første del er vores version, den anden del er vores strength value, den tredje del er vores salting og den sidste del er vores hashede password.

- Fordi alle vores kald med http til vores rest api er uafhængige er der et problem med authentication. Vi skal have tilføjet en måde vi kan holde på brugeres credentials når de er logget ind. Vi bruger JWT til at lave tokens der authenticater brugeren. Vi vælger at gemme dem i localStorage i hans browser, men der er også nogle problemer hvis nogen får adgang til hans browser kan de se den token han har.

- Et problem med session cookies er de ikke er stateless, de fungerer fint hvis det er server centreret application da vi ikke har noget mod at refreshe fra serveren hele tiden. Cookies har dog fordelen at de kan markeres med HttpOnly og Secure der gør at de kun kan sendes gennem Https connections som er et ekstra lag af beskyttelse. JWT kan dog bruges stateless da vi kan gemme brugerens oplysninger andre steder. JWT har fordelen at man ikke behøves server-side looksup hver gang du bruger applicationen som gør den kan være hurtigere. Begge deles kan bruges, vælg hvad der passer din app.

- Når vi logger ind får vi tildelt en JWT streng. Den indeholder vores header som fortæller hvad vi bruger (HS256), payload som er den data den indeholder fx, navn, rolle etc, samt en verify signature der er en combination af at hashe payload og header sammen med din secret. Kan vises med Postman Login, tag din token og smid den i jwt.io hjemmesiden.

- Når vi laver access rules på de forskellige endpoints kan vi endten preauthorize vores endpoints med en annotering eller vi kan gå ind i security config og tilføje hvilke roller vi gerne vil have på. Se controller IntelliJ og security config IntelliJ.

- Gå ind i Postman og brug login user, derefter tag strengen og sæt den i Authorize med ''Bearer STRENG''.

- Når vi logger ind i vores login.js kan man se at det vi får retur fra vores login kald er username, role og token. Vi gemmer vores token i localStorage, og hvis vi logger ud fjerner

vi den fra localStorage. Dvs hver gang vi skal bruge vores informationer ligger de i localStorage til nye kald.

- Når vi skal lave et fetchkald der skal bruge vores har vi en makeOptions der modtager parametrerne, method, body og token. Vi sætter method til den vi skal bruge fx ”GET” body til det vi skal sende med og token til true/false. Hvis den er true henter vi token fra localStorage.getItem så den kan sendes med i vores fetch(URL, makeOptions(X,X,X).

- Når vi logger ud og bruger JWT er der det problem at vores token stadig er aktiv i den duration vi har givet den i backend. Det betyder hvis en anden får fat i den token kan de bruge den til at lave kald til vores backend og blive accepteret selvom brugeren er logget ud. Når vi logger ud i vores løsning fjerner vi dem bare fra localStorage men den eksistere stadig som en accepteret token på backend.

- Hvis man vil få ulovlig adgang kan man bruge ting som JWT builder og gætte sig til din secret hvis du nu har valgt en dårlig en som ’secret’. Derinde kan man reverse din token string ved at skrive data ind, så hvis du kender nogle af data fra andre kald med den json du modtager kan du gætte på secrets indtil du rammre rigtigt. Man kan også bare tage en PC der er logget ind og lave en alert på den localStorage man skal bruge som fx alert(window.localStorage.token) så vil den give dig token. Det kræver dog brugeren er logget ind og nogen har adgang til hans connection.

- Jeg har ikke helt styr på CORS tror jeg, her er hvad jeg forstår. Vis CorsConfig ALLOWED_ORIGINS. CORS bliver brugt til at beskytte mod at kald til api kan laves fra andre steder. Så når vi har en frontend der hedder noget andet end vores hostede backend fx Cars_Frontend.net (front) vs Cars_Backend.net (API) så vil den ikke godkende det da vores kald kommer fra et anden domain. Men med CORS setup kan vi vælge at acceptere kald fra andre domains så vi kan sætte at Cars_Frontend.net bliver accepteret som en url der kalder vores API. Hvis man sætter den til ’’*’’ vil alle domains kunne kalde den. Det kan være et problem hvis vi har følsomme data som andre ikke skal kunne kalde, men kun skal kunne kaldes via vores API. Så CORS headers configurationer gør altså at vi kan vælge hvilke domains der skal have adgang til vores API og ikke få en CORS fejl fra Same-Origin-Policy (SOP) fordi web browsers som standard kun tillader request fra samme domain.

Det er hvad jeg har tænkt mig at gennemgå på mine 15 minutter til teknik eksamen.