

PG6300 – Webutvikling og API-Design: Eksamens

Av: Robert Mattias Molin.

Min Lösning:

Egentligen är applikationen i samma status som vid Oppgave2, då Web applikationen fungerar nästan helt som tilltänkt med fungerande funktionalitet för att spara en ny användare och logga in denne med autentisering i form av ett JWT token. Vid inloggning med token så visas också användarens personliga lista med spel som finns lagrat i mongo Db.

Inspirations sidan har jag inte hunnit att implementera dessvärre, då jag fick lägga allt krut på att få grundfunktionaliteten att fungera. Även WebSockets och inpackning av API i Docker-container är utelämnat av samma skäl som inspirationssidan.

Testing av applikationen är heller inte implementerad. Har gjort ett försök med att skriva en test klass för server-testing med Jest, men är väldigt osäker på hur det hur man löser biten med Auth när man testar endpoints som kräver att ett token finns. Det var inte så här jag önskade att lämna in min "Finale" men det blir dessvärre så. Mitt mål var att testa endpoints på servern för att säkerställa att integreringen mellan stack komponenterna är på plats, som att spara ett spel och hämta spel från Db.

Funktionaliteten för att logga ut användaren på rätt vis är fortfarande inte på plats, I nuläget så tas bara token bort ur LocalStorage och användaren tas ej tillbaka till startsidan.

Har försökt att lösa detta med en funktion som skulle ändra state från loggedIn: true till false men utan framgång.

Inloggad användares namn syns inte heller i dagsläget, tänkte att detta kunde lösas genom att skicka namnet från formuläret där användaren fyller i sitt namn till en komponent som visas fram på spelsidan.

Gällande mobil applikationen så finns möjlighet att lägga till en ny användare, men fick problem med att generera användarens personliga lista.

Här ligger problemet i att jag inte använder rätt react-native komponenter, så listan visas nu med tags. För att lösa problemet så måste nog listan skrivas ut i en ListView eller liknande native komponent, men fick inte till att implementera detta korrekt. För att applikationen inte ska krascha har låtit dessa komponenter vara kvar.

Körning av applikation och HTTP val:

På lokalmaskin har jag använt IntelliJ IDE som verktyg och startat server, mongo Db och dev servern från terminalen med: npm start (port: 3000), node server.js (port: 3001) och kommandot mongod för Db som kör på standard port. För visuella effekter används BootStrap 3.

Mobil app startas med: react-native run-ios där simulator körs från Xcode, annars gäller samma förutsättningar som för web applikationen ovan.

Gällande applikationens URL struktur och HTTP val.
endpoint's:

```
post('/addUser', (req, res) => {...})
```

För att lägga till en ny användare. HTTP verbet POST används när något skall skapas.

```
post('/saveGame', (req, res, next) => {})
```

För att spara nytt spel

```
post('/login', (req, res) => {...})
```

För att logga in som existerande användare. Här måste en body skickas med användares username och password så POST används som en säker metod här.

```
get('/getGames', (req, res, next) => {})
```

Används för att hämta lista med spel från Db. HTTP verbet GET används när något ska läsas.

```
put('/updateGame/:id', (req, res, next) => {})
```

För att uppdatera ett existerande spel. HTTP verbet PUT ska användas vid uppdateringar.

```
delete('/deleteGame/:id', (req, res, next) => {})
```

För att radera ett existerande spel. HTTP verbet DELETE ska användas vid radering.

Jag har valt att kalla dessa endpoints med självföklarande namn så att protokollet skall vara så självdokumenterande som möjligt.

Diskussion:

1.)

En av de största fördelarna med att bygga en stack endast med JavaScript är att det leder till bättre effektivitet i team som utvecklar applikationen tillsammans på grund av bättre förståelse av källkoden som skrivs. Därav blir det inget gap mellan olika team som jobbar med back-end och front-end utveckling, som istället kan ske om olika teknologier används.

Jag tänker att man här även kan spara på kostnader i utvecklingsprocessen, då man kanske klarar sig med ett team istället för två. En till fördel kan också vara att det blir lättare att hitta rätt personer med rätt kunskap då JavaScript är ett av eller kanske det mest populära programmeringsspråket idag.

Åter användning av kod blir också lättare när hela stacken är skriven med samma teknologi (JS) i detta fallet.

Node.js är också en faktor som gör att en fullstack i JS är fördelaktigt, det är både snabbt och har högre prestanda än andra vanliga alternativ för back-end.

Några nackdelar kan vara att Node.js inte är lämpad för tyngre datahantering på server sidan, och även den faktorn att JavaScript på server sidan fortfarande är en relativt ung teknologi jämfört med exempelvis Java eller PHP. Detta leder i sin tur till begränsade integrations möjligheter med andra teknologier.

2.)

Ett Web API är ett system där en klient kan kontakta en web server och utföra operationer på data som finns på servern, en klient kan definieras som allt från en web browser till en mobil applikation.

Det består av en Back-end exempelvis mongo Db, en web server (Node.js) och klienter (web, mobil). Kommunikation mellan dessa komponenter sker typiskt som HTTP transaktioner och som direkta anslutningar.

Om en endast skriver en liten simpel applikation så finns kanske inte behovet för ett web API. Men om man behöver ett så kan det både öka hastighet och tillgängliggöra mobila enheter. Om det inte finns något klart syfte att skapa ett API bör man avstå från detta, då det tar både tid och resurser att skapa på rätt vis.

3.)

Den kanske största fördelen med att använda tokens över cookies är det att ett token är stateless, kontra cookies som är stateful.

Så back-end behöver inte komma ihåg tokens, då ett token redan innehåller all data som behövs för att kontrollera dess validitet och det kan även ge ut användare information genom efterfrågningar. Vid bruk av tokens får alltså servern betydligt mindre att göra.

Till skillnad från en cookie baserad autentisering då en session eller en autentisering lista måste lagras både på server sidan och på klient sidan.

Tokens fungerar också bättre vid Cross Domain med CORS i bruk, då cookies är bättre ägnat för single domain och sub-domains.

Med en cookie baserad metod så sparar bara en session id, men i ett JWT token kan alla typer av metadata sparar bara den är giltig JSON vilket är väldigt fördelaktigt.

Vid en cookie baserad autentisering så måste också back-end göra ett uppslag i en databas och roundtrip tar därför längre tid än det tar att avkoda ett token.

En annan fördel med tokens före cookies är att de fungerar bättre för (native) mobila plattformar som IOS och Android.

Den kanske största svagheten med autentisering med token (JWT) är dess storlek. En session cookie är relativt liten i jämförelse med ett litet JWT token.

4.)

Huvudpoängen med REST arkitekturen är att ett API skall vara resursbaserat, och att klienter interagerar med servers genom att begära saker hellre än handlingar.

Som jag ser det så finns det klart mer fördelar än nackdelar med att implementera nivå 2 och 3 i Richardsons maturity model.

Genom att implementera nivå 2 så börjar man ju använda HTTP specifikationen som den är definierad. Samspelet mellan server och klient blir också mera klart, ett GET kall ska returnera någonting och ett POST kall används till att skapa något och så vidare, Så egentligen ser jag inga direkta nackdelar med nivå 2. Man kan ju utan att ha en djup förståelse av HTTP specifikationen förstå samspelet mellan server och klient bara av HTTP orden, då man "GETting" någonting från servern eller "PUTting" någonting på servern.

En av de stora fördelarna med hypermedia controls (nivå 3) är att hypermedia kontrollerna i en respons berättar för klienten vad man kan göra härnäst och hur man ska göra det med URI till resursen som man vill modifiera.

Nivå 3 tillåter också servern att ändra sitt URI schema utan att klienter blir negativt påverkade (breaks). Så genom att implementera nivå 3 får man klart mera flexibilitet men också mera komplexitet, detta bygger ju på att kraven för nivå 1 och 2 också är

uppfyllda. Om applikationen är väldigt liten och enkel eller har en kortlevnadstid så finns det troligen inget behov för att implementera nivå 3 i modellen.

5.)

Vid automatiserad testning körs ju vanligen ett och samma test script många gånger i rad, så genom att automatisera testningen så sparar man både tid, resurser och pengar. Man behöver heller inte planera testning efter resurser (folk som testar) om man automatiserar testerna.

När det kommer till att välja en testmetod av antingen unit-tester, integrations-tester eller end-to-end så måste man väl ta i beräkning faktorer som tid och pengar.

Alla testmetoderna är viktiga i olika stadier i utvecklingsprocessen, men om jag måste välja en av dessa så väljer jag i detta fallet integrations testning.

Anledningen till detta är väl så simpel som det att när vi utvecklar en full-stack applikation så låter integrations testningen oss testa de individuella komponenterna kombinerat som en grupp, så front-end (web), server (Node.js) och Mongo DB fungerar tillsammans som tilltänkt.

6.)

Några fördelar med WebSockets (WS) så har man full-duplex vid utväxling av meddelanden, både klient och server kan strömma (stream) meddelanden till varandra asynkront och självständigt.

WebSockets kan även passera igenom de flesta brandmurar utan att behöva re konfigureras, vilket kan vara fördelaktigt.

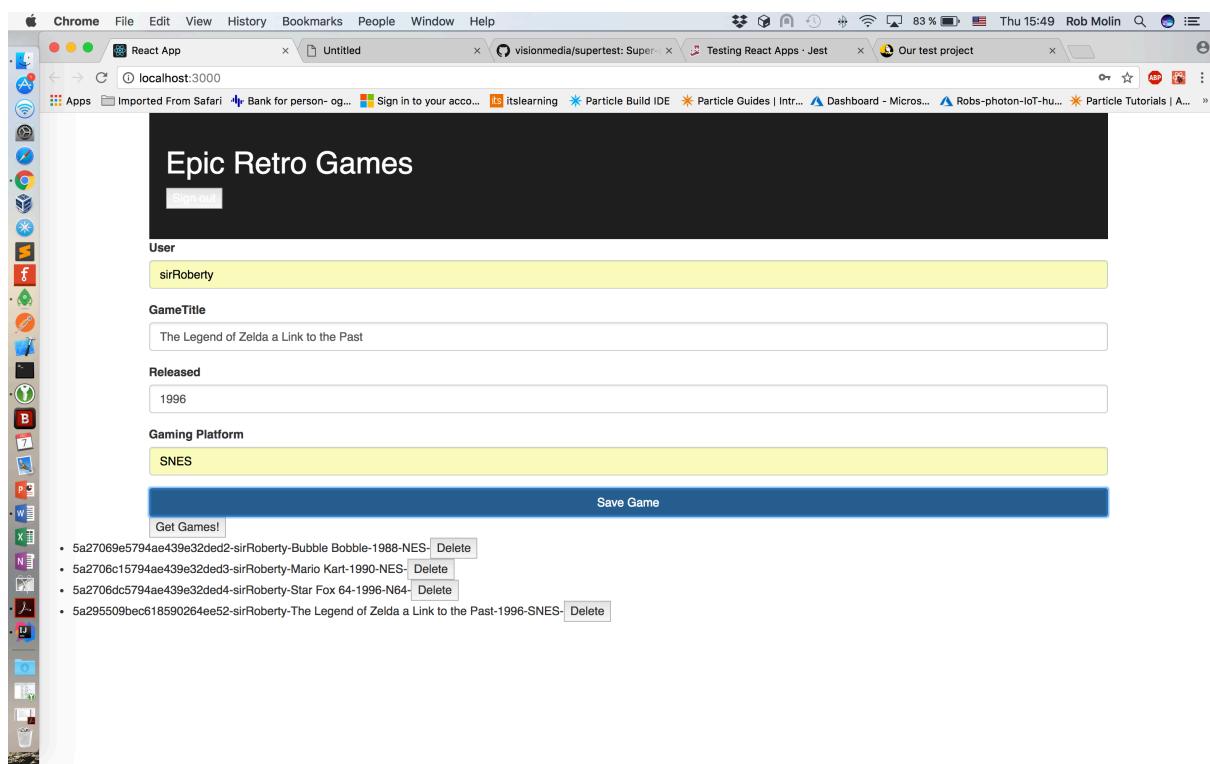
För real time apps är WS rätt val att ta, som exempelvis en live chat app. En nackdel med WS kanske kan vara att de kräver att en anslutning hålls öppen över en längre tid. Samt det att det också finns flera proxies som inte har support för WS än.

WebSockets bör inte användas om man kräver en mer omfattande support för error scenarion, då WS protokollet endast erbjuder error scenarion knutna till upprättandet av en anslutning. Så här är en HTTP anpassad lösning bättre.

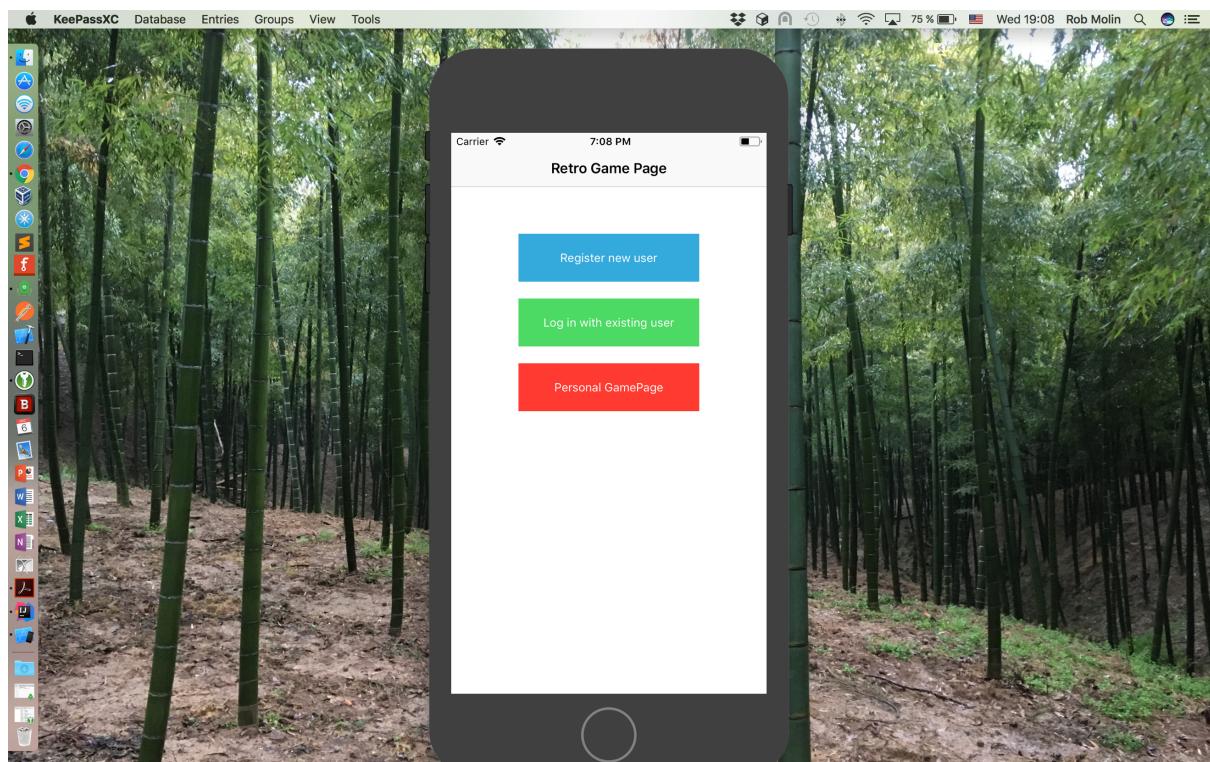
WS bör inte heller användas vid synkroniserade events därför att WebSockets protokollet inte erbjuder någon garanti att ett meddelande kommer att bekräftas i någon form.

Även när det kommer till "safe requests" så har inte WebSocket protokollet några direkt breda industristandarder.

Skärmbilder:



Inloggad som sirRoberty, personlig lista med spel visas.



Startsida mobil app.