**Från Koncept till Prototyp rapport av Jonas Rosberg**

Github Repo: <https://github.com/joro1993/Create-A-ScoriaAB/tree/ThirdVersion>

Min inlämning är en fortsättning från källkodsprojektet från inlämning 1 och 3.   
Denna gång använde jag create-react-app då jag ansåg att det var rätt svårt med inlämning 3. Det var mycket syntax-problem som jag vet egentligen borde funka, då dem har funkat i mina tidigare övningsprojekt. Så jag körde create-react-app i mitt repo, placerade om filerna i olika mappar och tog bort vissa filer som jag inte behövde.   
  
För att få tillgång till denna uppdaterade Single-page applikationen måste man vara inne i git branchen ”ThirdVersion”. Man startar projektet med npm kommandot ”npm start”, precis som tidigare.

Det finns en README.md fil tillsammans med källkoden som ger en tydlig beskrivning hur man startar upp projektet.  
Det finns även en Api.md fil tillsammans med källkoden som visar vilka paket jag har använt och så stor det även under rubriken ”Tjänster” vilket REST api jag använder.

På uppgift 3 – The Proof of Concept så använder jag mig utav 10 komponenter om man räknar med App.js. 1 utav dem(orders.js) var bortkommenterad för att jag inte fick mitt projekt att fungera med den koden som fanns i orders.js  
Jag använder nu orders.js, men döpte om den med ett stort O, alltså till Orders.js.   
Detta för att jag ville ha mitt projekt lätt att navigera mellan filerna, och inte ha det rörigt.   
Så jag skaffade 2 mappar under mappen Components. En mapp som heter logic, alltså att logisk kod som skapar funktionerna på hemsidan. Och en mapp som heter views, där jag har alla mina UI-komponenter.   
Komponenterna från The Proof of Concept är följande:  
business.js, contact.js, home.js, menu.js, myIdea.js, orders.js, product.js, prototype.js, worker.js.   
Alla dessa komponenter förutom orders.js och workers.js är view-kompoenter, orders.js och workers.js är en logic-komponent och därav i logic-mappen. Så jag la in dem andra i view mappen.

Mina nya komponenter som jag har lagt till i view-mappen är copyright.jsx, footer.jsx och header.jsx.  
Mina nya komponenter som jag lagt till i logic-mappen är Danger.jsx, errorBoundry.jsx, Filter.jsx, Products.jsx(ska inte blandas ihop med product.jsx i view-mappen).  
Totalt 7 nya komponenter där en av dessa är en error boundry-komponent.  
  
Uppdaterat:  
Ändrade JSX-syntaxen i React view komponenterna till mer Semantic HTML(JSX).  
Lade till en handleError funktion till min fetch.

**d1 - Benämn och beskriv kort 3 st olika platser för datalagring som finns tillgänglig i en webbapplikation på *klientsidan*. Beskriv enbart de typer av platser där information kan sparas som är kvar mellan de tillfällen applikationen används av användaren.***(uppdaterad)*  
localStorage är väldigt vanligt att använda sig av datalagring. Om jag skriver tillexempel localStorage.setItem(”key”, ”value”) så bestämmer jag med en nyckel och ett värde vad jag ska placera i localStorage. Man kan även skriva removeItem eller clear för att ta bort objekten som är sparade i localStorage.  
För att hämta data i localStorage kan jag istället skriva getItem.  
  
IndexedDB är ytterligare ett API som går att användas till lokal datalagring på klientsidan. Precis som i localStorage måste man använda en nyckel för att lagra och hämta data. Asyncron kod används med IndexedDB för att undvika att blockera applikationer.  
  
Webbkakor är den äldsta metoden för datalagring.  
Webbkakor kan man skriva själv genom tillexempel JavaScript.   
Sekretessinställningar och autentisering data är två vanliga områden man ofta kan tänkas att använda webbkakor till.

**d2 - Vad är viktigt att tänka på vid val av datalagringsplats ur en utvecklares perspektiv?***(uppdaterad)*Det viktigast att tänka på är att man väljer det som passar bäst till sin lösning. En unik applikation kräver en unik lösning. LocalStorage kanske inte är passande i en applikation, men IndexedDB passar bättre.   
ÎndexedDB stöds av dem flesta nyare webbläsare, om en utvecklare har detta i åtanke är indexedDB ett bättre alternativ.

Om en utvecklare skapar en youtube-kopia(annan videoplattform) kanske hen vill kunna spara en specifik tid i videon, på något riktigt roligt för att kunna skicka den till sin bästa kompis, så den personen får skratta lika mycket. Då är Query Parametrar ett bättre alternativ i den situationen.

**d3 - När vi gör förändringar i vår applikation, vad är viktigt att tänka på om vi använt webbläsaradressen (URL) som en datalagringsplats för klienten?***(uppdaterad)*När man sparar information lokalt i <URL:n>, så bör man alltid se till att ändringarna man gör i applikationen även är up-to-date så att rätt komponent visas i rätt URL.   
Det är viktigt när man skapar en Query Parameter att den har rätt matching path(match.path/myComponentURL). Om vi sedan skulle ändra namnet på komponenten till notMyComponentURL, skulle en fel match komma, och <URL:en> skulle bli fel.

**r1 - Vilka designmönster/strukturer har du använt dig av i koden för att separera applikationens UI från kommunikationen med REST API:er i källkoden, varför är detta en lämplig struktur?**Jag använde mig utav redux, redux använder sig av fyra komponenter: store, reducers, actions och views.  
Jag använde mig av redux’s action för att seperera kommunikationen med mitt REST Api.

|  |
| --- |
| **r2 - Vilken syntax och metod/tankesätt har du använd för att se till att den asykrona koden felhanteras på ett korrekt sätt. Beskriv vilka potentiella fel och problem du har identiferat och hur dessa hanteras i koden, varför är detta lämplig felhantering?**  *(uppdaterad)* Jag skapade en handleError function som jag implementerade i min fetch-metod. Tanken är att denna functionen ska ge ett error när servern inte körs. Och då kommer denna även fångas upp utav errorBoundry-komponenten. Fetch används även för att hämta db.json filen, då i denna applikationen så används det en json-server.  Ifall servern inte är igång så funkar applikationen precis som den ska, enda skillnaden är att produkterna laddas långsammare när servern inte är på, än vad dem gör när servern är på.  Detta kanske inte är korrekt felhantering, men jag ser det som en smart grej att ha en statisk ”backup” json-fil som ger samma resultat som om servern är på.  **r3 - Vilka platser använder du för lokal datalagring i din applikation, varför är dessa lämpliga val? Om du inte har implementerat någon lokal datalagring, vad ser du för potentiellt användningsområde i din applikation?** *(uppdaterad)*  Jag använder mig av redux’s store som jag använder till localstorage och där jag skapar en createStore till mina reducers.  localStorage används i min applikation för att spara produkter i min varukorg. Så man kan bläddra runt på hemsidan och produkterna är fortfarande kvar.  Detta var ett lämpligt val för Redux är för tillståndshantering. Detta gör så att ”State”:en till min applikation sparas i en Redux store, vilket gör att mina komponenter har tillgång till alla state’s som är nödvändiga, från store. I applikationen används det en json-server, om det inte skulle finnas en server i bilden, och det enbart var en simpel applikation, så skulle Redux inte vara nödvändigt. Så i detta fall var det nödvändigt, och därmed ett lämpligt val. |
|  |

Jag ansåg att det skulle se mycket mer renare, mer lättsamt för ögat när man ska kolla i

projektet. Därför tycker jag det är en bra kodstruktor, det blir inte lika rörigt.**r4 - Med vilken metod har du uteslutit filer som kan genereras eller installeras i ditt källkodsprojekt, av vilken anledning har du valt att utesluta dessa filer? - Motivera ditt val!**

*Inget svar – kunde inte fixa detta krav.*