GREENSWAY

Inlämningsuppgift 3: Från koncept till prototyp

DEN 30 APRIL 2021

WIVIANNE GRAPENHOLT
grwi20ql@student.ju.se

Innehåll

Länk

WiviWonderWoman/GreenSway

Mitt bibliotek

react-minimal-pie-chart

Det är ett kompakt paket för att implementera SVG pie charts, med costum-labels och css-animeringar i React. Jag använder det för att visualisera ”förbrukningen”av de olika avfalls-fraktionerna för användaren i vyn ”Min översikt” .

Om extern datalagring och React

Det är viktigt när i komponentens livscykel informationen hämtas och att data synkas med Reacts tillståndshantering, det vill säga sparas i antingen state eller skickas vidare som props innan den renderas. Som exempel kan jag ta min PieChart komponent, om datan hämtas i componentDidMount för Overview så syns inte animationen för användaren. Jag hade kunnat använda en loading boolean i state för att motverka detta men då få en viss fördröjning av renderingen. I stället har jag valt att spara undan datan i Apps state och skicka ner den som props till Overview.

Om URL:en

Det är viktigt att se till att de angivna URL:erna går att besöka trots de förändringar som gjorts.

Mitt val av bibliotek

Jag vill citera utgivarnas motivation på npm:

” **Why?** Because Recharts is awesome, but when you just need a simple pie/donought chart, 2kB are usually enough.”

Precis därför valde jag react-minimal-pie-chart för att det löste det specifika ”problem” som jag ville lösa: att visualisera ett ”dougnot-chart” för användaren. Det är ett litet paket, relativt enkelt att förstå sig på och tillräckligt väl dokumenterat. Jag kontrollerade när det senast uppdaterades, antal nedladdningar/vecka och licensen på npm. Paketet är skrivit i TypeScript.

Min kodstruktur

Jag har försökt implementera ”colocation”, filer som används/ändras tillsammans nära varandra, samt ”atomic design”, små komponenter bildar större som till slut blir ”sidor” som får egna mappar... Bästa exemplet är mappen details. Dit har jag flyttat fraction-mappen, med all buissnies-logic för fractions , eftersom denna logik endast används här.

*När det gäller kommunikationen med REST API:er har jag valt en service-mapp i roten som innehåller själva caller.js, här skapas en instans av axios.create som har bas url:en. Min första intuition var att använda user-services som ”controller” men jag fick det inte till att fungera och placeringen av caller.js har varit svår. Caller importeras till de komponenter som gör anrop. I de komponenter som gör GET-anrop sker detta i componentDidMount, datan hämtas alltså bara en gång, och datan sparas i komponentens state vilket triggar en re-rendering av komponenten. UserForm, som endast renderas om ingen användare är sparad, använder en setUserEmail funktion, som endast anropas när användaren klickar på spara, för att uppdatera användarens email adress med PATCH, respons datan sparas i Apps state som därför renderas på nytt. Alla anrop är alltså väl ”isolerade” och själva bas-url:en är inkapslad i caller.js. Anledningen till att det är lämplig struktur förklarar jag delvis under rubriken ”**Om extern datalagring och React** ”, en annan anledning är om man ska ändra bas-url:en så räcker det att göra det på ett och samma ställe.*

Helt ärligt känner jag mig rätt ”lost” angående kod/fil-struktur i react projekt och det blir värre ju mer googlar.

Min prototyps externa datalagring

Att lagra data externt möjliggör ett dynamiskt resultat.
Jag använder fortfarande localStorage för att spara användarnamn och id, för att användaren skall slippa registrera sig varje gång samt för att kunna rendera användarnamnet om inte detta hämtats ännu från API:et. Jag sparar även id:et i URL:en via Routern och läser av det i Table för att hämta API-data.

All mock-data för användare lagras på https://retoolapi.dev/BOnmI8/greenswayusers. Används främst till att få slumpade siffror som anger ”förbrukning”/avfalls-fraktion. Jag tänker mig att det framöver ska gå att registrera förbrukning och även i månads/årsskifte kunna nollställas.

Data för fractions finns lokalt men tanken är att flytta detta till ett API som motsvarar alla fraktioner för varje ”garbagehouse” som finns angivet i användarobjekten. Då tänker jag även att dessa kan ange hur ”fulla” de är och därmed indikera när de behöver tömmas.

Drömmålet är att ha all buisness logic på en webbserver för att möjliggöra att använda både mobil och dator.