Loggbok – Frontend 2 - Alicia Petersson

7e januari

Laddade ner startmallen från GitHubRepo-länken och skapade ett eget GitHub repo med denna startmall. Installerade ESlint till projektet, var osäker om ESlint skulle commitas till mitt repo så avvaktar till imorgon och frågar på lektionen.

8e januari

Commitade Eslint-filerna till repot. Började med Vanilla JavaScript och hantera klicka på sätena. Gjorde en dev-branch och sedan en feature-branch ”feature-seat-selection”.

Komplicerade till det med byta ut eventlisterns callbackfunktionen beroende om sätet är valt eller ej. Men kunde inte ta bort en eventlistener som hade en anonym funktion korrekt. Jag gjorde det till en anonym funktion för att jag behövde ha en parameter med callbackfunktionen, men då visade det sig att jag inte kunde ta bort den som callbackfunktion. Sedan ändrade jag till så att callbackfunctionen inte byttes ut och hade en och samma callbackfunktion men hade mer logik i den funktionen istället. Det blev endast 1 eventLsitener callbackfunktion nu, **clickSeat** som tog emot en parameter som var Node-elementet som klickades på (vilket säte som klickades).

Skapade en variabe **selectedSeatsCount** som håller reda på hur många säten som har klickats på, 0 som startvärde. I functionen clickSeat så lade till så att klassen ’selected’ togglades, om den klassen nu finns med i elementets klass-lista så adderas det med 1 till selectedSeatsCount l. Om den klassen inte finns i klasslistan så har man avmarkerat det sätet och det subtraheras 1 från selectedSeatsCount. selectedSeatsCount skrivs ut till spannen med id’t #count. Den totala summan beräknas och skrivs ut till spannet med id’t #total. Så när användaren tryckerpå ett säte så uppdateras antalet säten och totala priset nere i skärmen.

Om användaren skulle välja att byta film i rullistan så uppdaterades inte priset, då lade jag till en eventListener på select-elementet som lyssnar efter ”change”, och anropar callbackfunktionen **displayTotalCost** som beräknar om priset och skriver ut det.

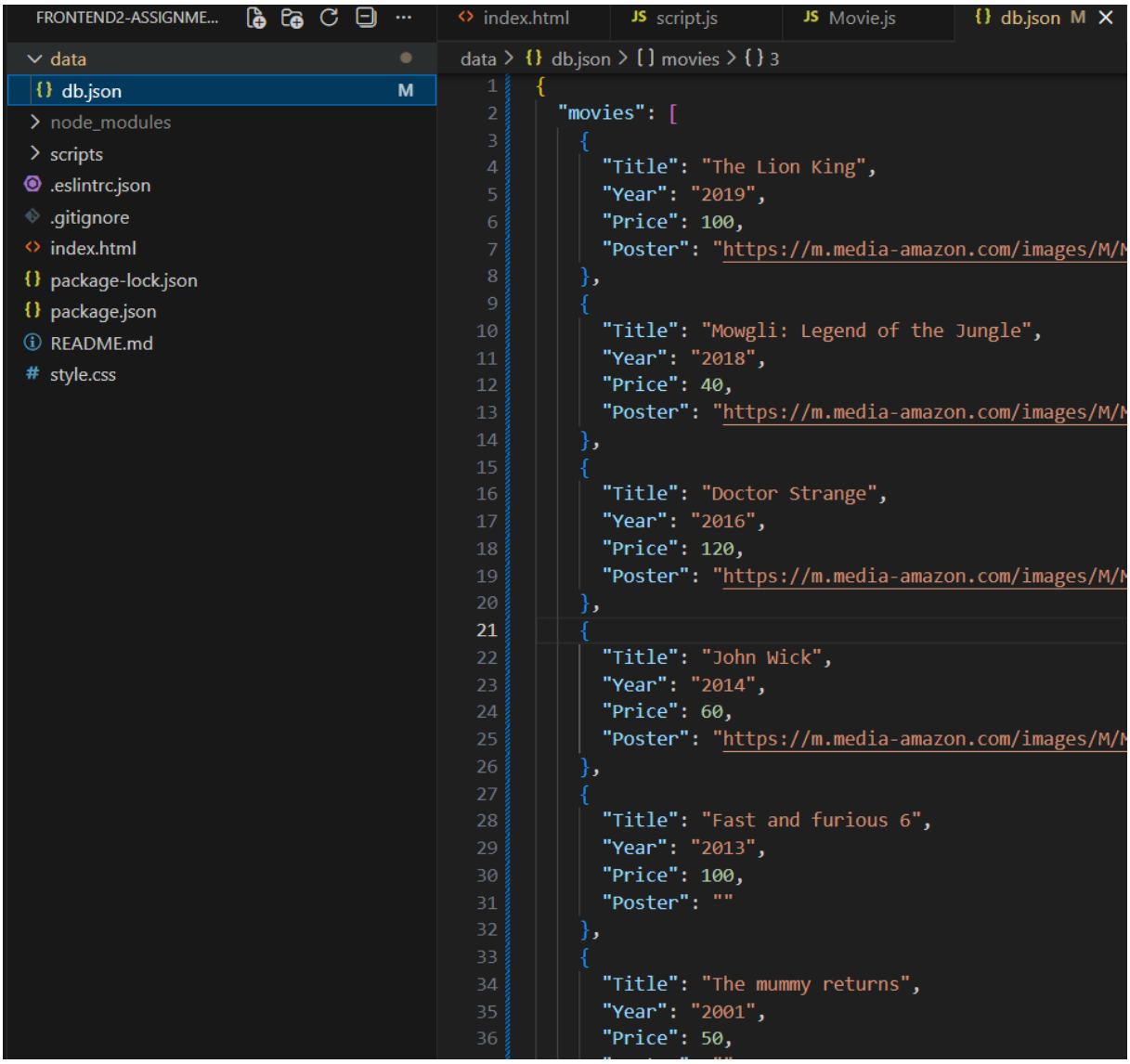
9e januari

Hade affärsmannaskaps-föreläsning idag så inte så mycket gjort. Jag försökte fixa lite errors som ESlint flaggade på, men fick väldigt många errors om att jag använder CRLF linebreak och inte LF linebreaks. Så funderar på att ha det som en rule att ändra det till windows, men är lite osäker på om det är okej så får fråga vid nästa lektionstillfälle. Bland annat så fick jag även flytta kodblocket där jag lägger till eventlisteners längst ner för ESlint klagade på att funktioner som inte har deklarerat har används, tycker det är konstig för att alla funktioner hoistas så det är ju inget direkt problem egentligen. Men för att minimerar ESlint errors så gör jag så.

Skapade en klass ’Movie’. Installerade json-server, skapade en db.json fil som jag lade i en mapp som heter data. Då var jag tvungen att ändra kommandot som startar json-server för att sökvägen för filen skulle hittas, ändrade till:

```
npx json-server data/db.json
```

Lade in information i databasen, baserat från inlämningsuppgiftens exempel men lade även till filmerna som var hårdkodade i startmallens HTML-fil.



12e januari

Lektion och övningsuppgift under förmiddagen/ en del av eftermiddagen.

Gjorde klar det sista med db.json: Tog bort Poster för varje film, sedan märkte jag också att jag hade skrivit fel att nyckelvärdena började med stor bokstav, så justerade det så att nyckelvärdena började med små bokstäver. Tog bort alla options-element i index.html:

```
12  -       <select name="movie" id="movie">
13  -         <option value="100">Fast and furious 6 (100 kr)</option>
14  -         <option value="50">The mummy returns (50 kr)</option>
15  -         <option value="70">Jumanji: Welcome to the Jungle (70 kr)</option>
16  -         <option value="40">Rampage (40 kr)</option>
17  -       </select>
    12  +   <select name="movie" id="movie"></select>
```

Mergade branchen **feature-seat-selection** till branchen **dev**.

Skapade en ny branch: **feature-movie-selection.** Skapade funktionen displaySelectOptions, som hämtar från json-server-filen db.json med hjälp a fetch. Den hämtade datan mappar över och det skapas en Movie-instans av varje hämtad post från ”databasen”. Som sparas i en array **movies**. För varje movie i movies skapade jag en option-element där värdet var filmens pris och innertexten var filmens namn och pris inom parentes.

Gjorde lite småfix, som att jag hade glömt att använda url-variablen i fetch-anropet, det rättade jag till och sedan så extraherade jag ut det som hörde till movies, Movie-klassen och att hämta filmerna från json-server lade jag i en egen funktion **loadMovies** som även det fick vara i samma fil som Movie-klassen, detta blev en modul där jag bara exporterar funktionen **loadMovies**.

13e januari

Kollar morgondagens lektions-material.

Döper om funktionen displaySelectOptions till displayMovieOptions för att göra det extra tydligt vad functionen gör.

Mergar branchen **feature-movie-options** till branchen **dev**, men denna gång via en pull request på GitHub. Nu ville jag prova att dela upp koden i ännu flera moduler. Jag skapade 2 nya filer, en seat.js som ska innehålla klick-logiken för när man trycker på ett säte och en ui.js, som bara ska rendera grejer till DOMen.

Ui.js hämtar 3 element från DOMen, och exporterar 3 funktioner; displaySeatCount, displayTotalCost och displayMovieOptions.

Seat.js exporterar funktionen clickSeat, som är lite modifierad. Koden som skriver ut antalet till DOMen har flyttats ut till en egen funktion **displaySeatCount** (ui.js). Denna modul håller också reda på hur många säten som är markerade, den exporterar även den variabel för att den används i script.js.

14e januari

Lektion på förmiddagen och React-övningsuppgifter på eftermiddagen.

15e januari

Fortsätter lite med React-övningsuppgifter.

Men nu vill jag arbeta med React i inlämninguppgiften, så jag skapar ett nytt projekt med en React-template med Vite och jag vill även använda TypeScript istället för JavaScript i mitt projekt.

Skapde en ny branch ”**dev-react**” för att separerar det från min tidigare dev-branch. Sedan bytte jag ut hela projektet till en React (Vite) applikation som använder TypeScript.

Skapade en feature branch ” **feature/project-ui-template**” där jag lade in gränssnittet som man fick med från inlämningsuppgiftens start-repo.

19e januari

Jag skapade en tom (än så länge) MoviePicker-komponent och integrerade den i App.tsx Jag skapade även klassen Movie (igen, bara kopierat från den jag gjorde tidigare). Men eftersom jag använder TypeScript fick jag även lägga till typning i Movie-klassen. Jag fick åter igen installera

json-server, till detta projekt men den här gången lade jag db.json i src-mappen. Så för att starta json-server skriver jag nu detta:

```
npx json-server src/db.json
```

Jag fyllde db.json med testdata igen. Jag skapade även en MovieEntity-klass som representerar filmdata från databasen. Jag gjorde så eftersom i databasens filmobjekt även hade år, men Movie-klassen endast skulle ha namn och pris.

20e januari

Skapade en mapp vid namn **api** som jag lade i src-mappen där jag tänkta ha filen som utför någon typ av API-anrop. Dels för att det är bra att samla alla API-anrop på ett och samma ställe och det gör även den mer ”skalbar” i längden om jag lägger en bra grund till mappstrukturen. Jag skapade även en movies.tsx där funktionen loadMovies kommer ligga. (Edit: Det är fel filändelse just nu på den, men upptäckte det någon dag senare). Jag började med att kopiera över funktionen loadMovies jag hade gjort tidigare i vanilla JavaScript, men det jag var tvungen att justera var typningen av variablerna etc.

Jag upptäckte även ett fel i MovieEntity-filen. Då jag hade kopierat strukturen från Movie-klassen så glömde jag ändra vad klassen hette och vad som faktiskt exporteras, så ändrade så att det verkligen exporterades en MovieEntity och inte en Movie.

21a januari

Lektion och grupparbete.

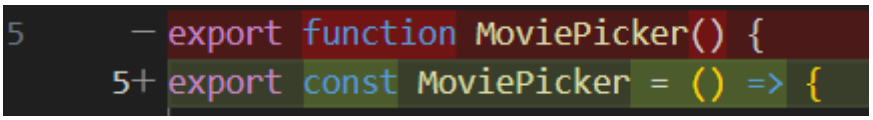
Jag gjorde den sista av typningen för funktionen loadMovies. Jag typade vad returdatatypen var, en array av Movie, men eftersom det är en async funktion så måste den returnera ett Promise (som ska resolvas till en array av Movie).

Upptäckte att jag hade använt fel filtyp för filen med funktionen loadMovies, jag hade först satt den till .tsx men det är fel eftersom att den inter returnerar någon React-komponent, så den korrekta filändelsen är .ts, som jag justerade till.

22a februari

Grupparbete fram till ca 14:30, sen:

Ändrade att MovieEntity är ett interface och inte en klass, som visades på lektionen igår. Tog då bort konstruktorn från MovieEntity-filen. Flyttade även den filen till en annan mapp vid namn ’interfaces’. Gjorde klar komponenten MoviePicker, den hämtar filmer från json-server databasen i en useEffect och använder useState för att lagra arrayen med filmer. Hittade också ett fel, där jag inte hade skapat react komponenten med en arrow-dunction utan en vanlig function, så justerade detta också.

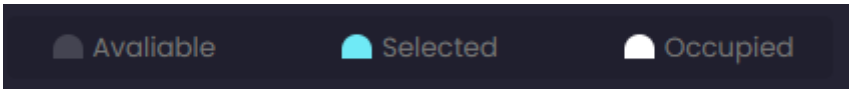


Sedan skapade jag BookingSummary-komponenten för att visa bokningssammanfattning som visar antalet valda säten och totala summan för bokningen. Lade till state-hantering med useState för att spåra antal valda platser och totalt pris.

För att visa ett säte skapade jag en Seat-komponent som är återanvändningsbar vid flera olika användningsområden (för att kunna använda den ikonbeskrvningen samt den faktiska klickbara sätet). Seat-komponenten tar nu emot en size-prop (small/large) och en isOccupied-prop samt en isSelected-prop och baserat på dem så läggs rätt CSS-klass till för att indikera sätets status. Jag lade också till CSS-regler för storleksvarianterna. Jag ville göra Seat-komponenten så generisk som möjligt så att jag kunde använda den så enkelt som möjligt också, likadant med size. Att jag kan ha mer valmöjlighet och då kunna justera sätena mer till hur jag ville ha dom. Detta gjorde jag också för att försöka tänka på skalbarheten, att man lättare ska kunna justera ett säte och även kunna lägga till fler props i framtiden på ett smidigt sätt.

Samtidigt med detta skapade jag en SeatLegend-komponent för att visa vad varje sittplatstyp representerar. Jag valde att göra det möjligt att ha den beskrivande texten som child till Seat-komponenten så att man kunde visa någon typ av label för varje säte som bara skriver ut det om det finns en child-props med till Seat-komponenten. Och valde också att sätta storleken till small på dessa säten som skulle ha en label. Jag tog mig också friheten att ändra texten N/A tillAvaliable för att göra det mer tydligt för användaren vad de gråmarkerade sätena betyder.

```
<Seat size="small">Avaliable</Seat>
```



Slutligen fokuserade jag på layout och styling. Jag centrerade alla element på sidan, justerade bredden på sittplatsraderna för att matcha varandra för när jag importerade start-mallen till React-appen så betedde layouten sig lite annorlunda från vanilla-projektet av någon anledning. och löste några mindre style-buggar. Sedan bytte jag ut alla div-seats till min egna Seat-komponent och avslutningsvis extraherade jag alla sittplatsrader till en ny SeatGrid-komponent för bättre kodorganisering. Började fundera på hur jag skulle strukturera upp det med bokningsfunktionaliteten och försökte tänka ut vad bästa sättet att koppla en bokning till en specifik sittplats kunde vara för att vet om sätet ska vara occupied eller inte. Det första jag tänkte på var en 2D-array samt att varje säte måste ha något typ av ID. Samt hur jag kunde hantera hur användaren klickar på ett säte och ändra utseende på det sätet.

23e januari

Grupparbete fram till ca 15. Sedan:

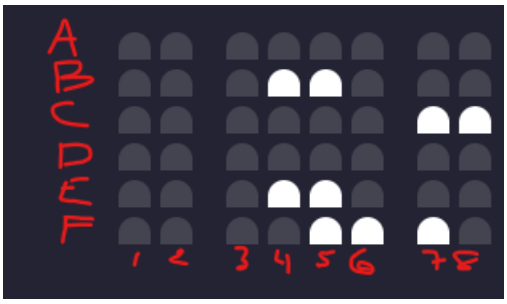
Jag ändrade hur Seat-komponentens props fungerade med att ange om den var klickad eller occupied. För nu kunde man skicka in både isSelected och isOccupied och det är inget beteende som jag vill ha. Så jag bytte ut propsen isOccupied och isSelected med en ny optional property *'status'*, som är av en string union type där man antingen kan välja *'selected'*, *'occupied'* eller *'avaliable'*, men den defaultar till 'avaliable' om inget status-propery anges. Då blev det enklare att ange klassnamnet också eftersom selected och occipied är även klassnamnen som bestämmer sätets utseende. Justerade i de olika komponenterna som använder Seat-komponenten p.g.a dessa ändringar Seat-komponenten fick.

Jag hittade även ett typo där jag inte hade använt camelCase på ett korrekt sätt i useState set-funktionen ' setSelectedSeatsCount'.

Sedan mergade jag branchen **feature/project-ui-template** till branchen **dev-react**. Skapade en ny branch **feature/booking-db-display-booked-seats**. Kanske inte bästa namnet, hade kanske nu i efterhand enbart kunnat döpt den till ’feature/display-booked-seats’ eller nått.

Skapade en ny **tom fil** *bookings.ts* i mappen *api*, som kommer att hämta bokningarna och returnera bokningarna för en specifik film.

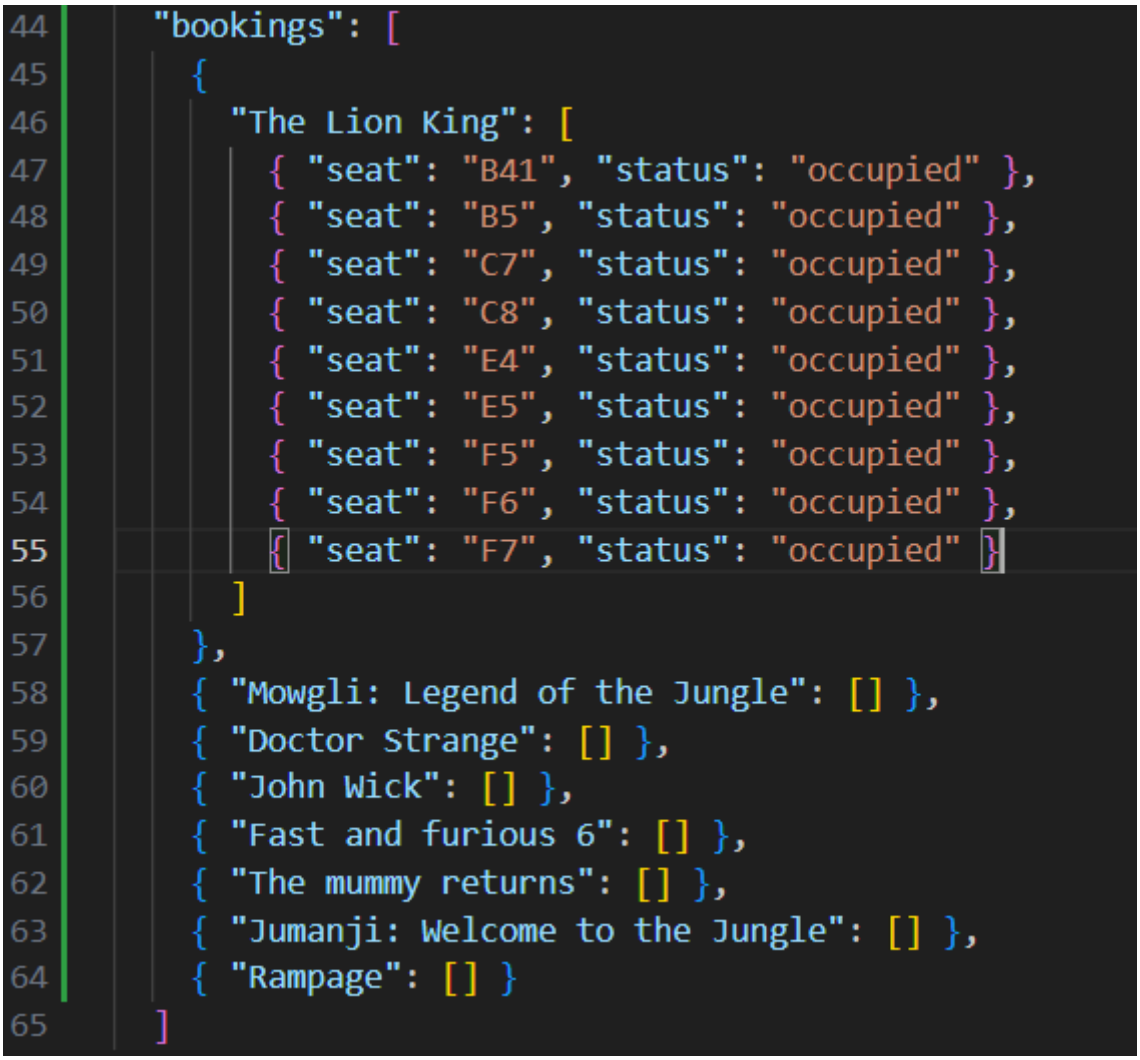
Jag funderade hur jag ville strukturera upp och märka de olika sätena, jag funderade mellan att ha raderna som A-F och varje rad har säten 1-8. Eller enbart ha siffror för alla sätena, 1-48.



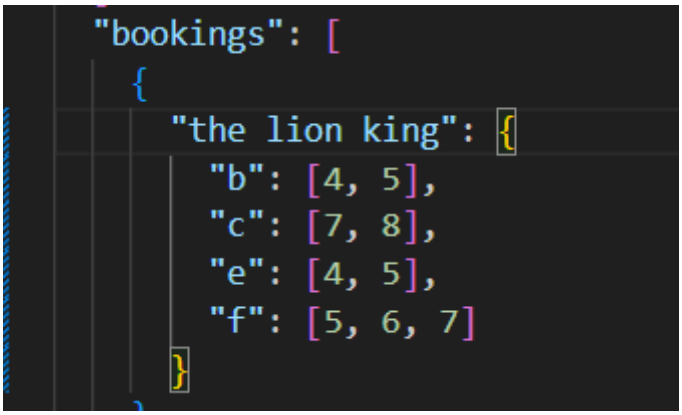
ELLER

Jag bestämde mig för att gå med varianten till vänster ovan i bild. Började då fundera på hur ag ville strukturera upp hur databasens för bokningarna skulle se ut så att jag visste på förhand innan jag började koda hur databasen skulle indexeras i koden (säger man så?) Jag gick igenom en rad olika varianter på hur bokningarna kunde struktureras upp.

Det första jag provade var detta:



Sedan ändrade jag mig och funderade på att strukturera upp det på detta sätt, så att jag enklre kunde indexera till en specifik rad i koden, och att jg faktiskt inte behövde ha med status, för alla som finns i denna json-fil är bokade och kommer ha status occupied när sätena renderas ut.



Men sedan kom jag på att jag ville ju även koppla en specifik kund till bokningen, så då jag jag tillbaka till varianten innan där jag kunde lägga till en kund till bokningen, se nedan. (Glömde att ta bort ”status”: ”occupied” innan jag tog skrämdumpen).

Men funderade jag på att den optimalaste lösningen om jag skulle ha detta alternativ var att kanske ha en separat databas för kunder och bara koppla ett kundID till bokningen så att jag slipper skriva kundinformationen flera gånger. Något liknande detta:

Men för att inte göra det för komplicerat och invecklat så gör jag inte detta nu utan kan justera senare om jag känner att jag har tid. Då det inte stod något i inlämningsuppgifterna om att det endast får finnas en kund med samma telefonnummer exempelvis.

Sen funderade jag på att ha denna struktur för att enklare kunna hämta en specifik rad lättare. Och att sätet indexeras med en siffra så kan jag möjligtvis hitta sätet enklare också med index-värde.

Men sedan kom jag på att jag kanske inte skulle ha mellanrum i nyckeln för objektet så då bytte jag ut mellanrummen till bindestreck. Så när jag skapar en ny film sedan får jag göra om

filmtiteln till små bokstäver och byta ut mellanrum mot bindestreck när bookings-databasen uppdateras med den nyskapta filmen. Jag insåg sen också att om jag på ett enkelt sätt ville komma åt bokningarna för en specifik filmtitel via exempelvis:

```
bookings[movieTitle]
```

Så var sättet jag gjorde i bilden ovan fel, så justerade så att det såg ut som bilden nedan.

```
"bookings": {
  "the-lion-king": {
    "a": [],
    "b": [
      {
        "seat": 4,
        "customer": { "name": "John Doe", "phone": "070 123 45 67" }
      },
      {
        "seat": 5,
        "customer": { "name": "John Doe", "phone": "070 123 45 67" }
      }
    ],
    "c": [
```

24e januari

Sedan funderade jag på om det inte var bättre att varje film fick ett ID, och funderade om jag kunde använda för att generera ett ID när en ny film skapas:

```
Date.now();
```

Men efter lite googling så fanns det en chans att detta kunde i vissa fall skapa dupletter om 2 filmer skapades EXAKT samtidigt. Så bestämde mig för att behålla film-titeln som nyckeln i bookings-databasen. (så länge).

Jag såg att jag hade fel filändelse på MovieEnitiy interfacet, så jag döpte om filen MovieEntity från .tsx till .ts för att reflektera dess faktiska syfte som ett interface och inte en React-komponent. Och gjorde en till fix i ”bookings-databasen” där jag märkte att jag inte hade ändrat till det nya format för alla filmer som jag hade lagt till tidigare.

Nästa steg var att skapa de nödvändiga interfaces och typer för att representera bokningsobjekten. Jag skapade SeatBookings, Booking och Customer interface som skulle så att jag kunde typa mina objekt vilket även ger mig fördelen med en del intellisens. En ny API-funktion implementerades i bookings.ts för att hämta alla bokningar från databasen. Den var mer eller mindre väldigt lik hur jag hämtade filmerna, bara att jag inte behövde skapa om dom till nya klass-objekt som jag gjorde med Movie-objekten.

För att hämta alla bokningar från APIet i applikationen lade jag till en useEffect-hook som fetchar bokningarna vid mount och lagrar dem i ett useState i App.tsx. Därefter började jag strukturera upp komponenterna genom att skicka alla bokningar som prop till SeatGrid-komponenten. Under denna process lade jag också till kommentarer för att förtydliga de olika raderna i SeatGrid-layouten, samt en variabel för att definiera antalet säten per rad.

Sedan lade jag till en selectedMovie-useState för den valda filmen, det värde gjorde jag först som en sträng där jag tänkte ha filmtitel som värde. Jag lade till setSelectedMovie som prop till MoviePicker-komponenten. Jag ändrade sedan useStaten från att lagra en sträng till att lagra ett Movie-objekt istället. Det är att jag tänkte att jag enklare även kan hämta priset från den valda filmen om det sparas som ett Movie-objekt som jag kan skicka med till BookingSummary-komponenten i framtiden.

En viktig uppdatering gjordes på filmmodellen genom att jag lade till ett id-fält till varje filmobjekt i databasen, samt uppdatera klassen Movie och MovieEntity-interfacen med en id-nyckel. IDt för filmen blir filmens titel i små bokstäver där mellanrum har bytts ut till bindestreck. Dettta är för att jag ska kunna hämta en specifiks films bokningar lättare via selectedMovie-useStaten. Jag insåg även att jag hade fel filändelse på Movie-klassfilen, så jag döpte om Movie-filen från .tsx till .ts. En bugg fixades där id:t inte inkluderades vid mappning av nya filmobjekt när filmerna hämtades från APIet i movies.ts filen.

En ytterligare optimering gjordes på SeatGrid-komponenten så att den endast fick bokningar för den valda filmen baserat på id, istället för alla bokningar för alla filmer. Därefter skapades en ny SeatRow-komponent och implementerades den i SeatGrid.

Fick problem med att componenter renderades 2 gånger. En första gång när sidan laddas in för en första gång och sedan en gång till när filmerna hade laddats in när selectedMovie useStaten ändrades. Vilket medförde att när SeatGrid renderades för första gången så var bokningarna undefined som medför problem i child-komponenterna. Så provade första att flytta över det som fanns i useEffecten i MoviePicker.tsx (som fetchar från movies APIet) in i useEffecten som finns i App.tsx (som hämtar bokningarna). Men det fick samma resultat så då provade jag att lägga till en ny useState som hanterar om all data har laddats in.

```
const [isLoading, setIsLoading] = useState(true);
```

Om isLoading är true så returneras bara en h2 element med texten ”Loading...”. Och det fungerade när jag laddade om sidan. Men vid nästa omladdning så var all data undefined även fast isLoading var sätt till false. Det var för att jag sätter setIsLoading till false innan all data har hämtats färdigt, för att jag inte kan kan skriva await loadMoviesData() och await loadBookingsData() inne i useEffecten. Då hittade jag något som hette Immediately Invoked Function Expression och att jag kunde lägga in await loadMoviesData och await loadBookingsData i en anonym funktion som körs omedelbart inne i useEffecten i ett eget scope, och där kan jag då awaita all data som behöver fetchas från APIet innan jag sätter isLoading till false.

Slutligen gjordes några mindre korrigeringar: filmets ID-formatering i bokningsdatabasen fixades, och redundanta kommentarer i SeatGrid.tsx removerades för att hålla koden ren och underhållbar.

25e januari

Jag tog bort useStaten isLoading och baserade renderingen på om bokningarna för den valda filmens ID var undefined, om den är det så har datan inte hämtats klart från APIet och då skrivs ”Loading..” ut. Men om det INTE är undefined, då skrivs alla komponenter ut. Jag valde att göra så för att då blir det för det första mindre kod och jag behöver inte ha en ”extra” useState vilket gör koden lite renare. Samt att if-satsen även visar ganska tydligt vad som inte har laddats in än.

```
43    if (bookings[selectedMovie.id] === undefined) {
44      return <h2>Loading...</h2>;
45    }
```

Jag mergade branchen ’**feature/booking-db-display-booked-seats**’ till branchen ’**dev-react**’.

Jag skapade branchen ’**feature/seat-selection**’. Jag började med funktionaliteten för att kunna välja och av-välja ett säte. Skapade en useState i Seat-komponenten. isSelected. Om isSelected är tru läggs CSS-klassen ’selected’ till i seatClass-variabeln, annars inte. Jag gjorde en function handleClick som anropade setIsSelected och inverterade värdet. onClick lades bara till på sätet om statusen var avaliable så att man endast kan välja säten som är lediga.

Insåg då att jag inte behövde ha ’selected’ i min Union type för propertyn ’status’, så tog bort det alternativet. Men insåg sen att jag behövde den för min SeatLegend-komponent för att rendera ett valt säte där, så i några commits senare lade jag till den igen. Gjorde några små buggfixar som att man inte ska kunna trycka på sätet som visar ledigt i SeatLegens-komponeten och ändra ett variabelnamn i Seat-komponenten för att göra det tydligare var label var för något:

```
21  -   const label = children ? <small>{children}</small> : null;
    21  +   const legendLabel = children ? <small>{children}</small> : null;
```

Lade till funktionaliteten att när man trycker på ett säte så uppdateras antalet i BookingSummary-komponenten, genom att skapa en ny useState ’selectedSeatsCount’ och skickar med dens setter till SeatGrid-komponenten och alla childs tills Seat-komponenten. Inne i Seat-komponenten anropades jag setSelectedSeatsCount i funktionen handleClick.

På liknande sätt skapade jag funktionaliteten för att updatera priset i BookingSummary-komponeten. Jag skickade med useStaten ’totalPrice’ setter funktion ner till Seat-komponenten. Men då behövde jag även skicka med vad biljetpriset var samt nuvarande antalet valda säten som props. Och inne i komponetnen Seat så beräknade jag det nya priset och med useStaten setTotalPrice uppdaterade jag även priset.

Sedan behövde jag ha något sätt att veta exakt vilka säten som var valda, vilken rad och vilket sätesnummer som har valts. Som jag kommer behöva i framtiden när jag implementerar att faktiskt boka dessa platser. Skapade en ny useState ’*seatSelection*’ där datatypen var ett objekt:

```
16  const [selectedSeats, setSelectedSeats] = useState({
17    a: [] as number[],
18    b: [] as number[],
19    c: [] as number[],
20    d: [] as number[],
21    e: [] as number[],
22    f: [] as number[],
23  });
```

Så att jag enkelt kan lagra för varje rad vilka sätesnummer som har valts och kan sedan i framtiden spara till databasen. Hade först problem med att lista ut hur jag uppdaterade staten om datatypen var ett objekt, med en nestlad array.

26e januari

Listade ut hur jag uppdaterade useState om värdet i useStaten var ett objekt, som min seatSelection är. Jag hittade på Reacts dokumentations sida hur man tar bort från en array och uppdaterar ett objekt-useState här:

https://react.dev/learn/updating-objects-in-state

https://react.dev/learn/updating-arrays-in-state

Så nu kan jag få fram till App.tsx vilka säten som har valts för en specifik rad.

Sedan bestämde jag mig för att ta bort komponenten **SeatGrid** då den egentligen inte medförde något nytt utan bara medförde att jag behövs skicka properties en nivå djupare för att komma till sätesraderna och de individuella sätena. Så jag tog bort den komponenten helt och flyttade upp SeatRow till App.tsx filen istället för att använda dom i SeatGrid.

Commitade dessa ändringar och sedan mergade jag branchen '**feature/seat-selection**' till branchen '**dev-react**'.

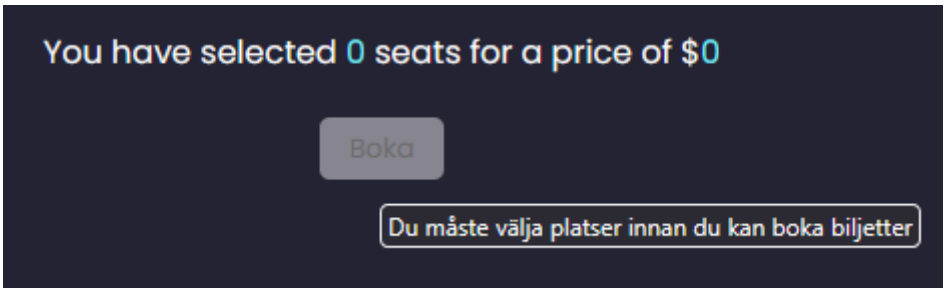
Skapade en ny branch '**feature/movie-selection**'. Implementerade funktionalitet för att markerade säten, antal och pris återställs när man väljer en ny film i rullistan. Jag listade även ut att jag kunde definiera en function i App.tsx som tog emot ett film-ID som parameter och i den funktionen kunde jag utföra all funktionalitet som att anropa setSelectedMovie, setTotalPrice, setSelectedSeatsCount och setSelectedSeats där jag återställde värdena. Hela denna funktion skickade jag med som en prop till MoviePickern. Detta istället för att skicka ner alla dessa setter-funktionerna ifrån de olika useStatsen till MoviePicker-komponenten och utföra logiken i i child-komponenten. Detta gjorde det mycket enklare och det var färre props jag behövde skicka med till MoviePickern.

Sedan stötte jag på ett problem med att valda säten inte återställdes när man ändrade film. Detta för att Seat-komponenterna har ett eget useState som håller reda på om de är klickade eller ej. Då var jag tvungen att strukturera om koden liten, för jag kom inte på något sätt hur jag skulle lösa problem med som koden var strukturerad nu. Det slutade med att jag använde useStaten selectedSeats för att kolla om sätet finns med i den useStateobjektet och om den är det ska Seat propertyn status sättas till 'selected' och jag tog bort useStaten isSelected från Seat-komponenten.

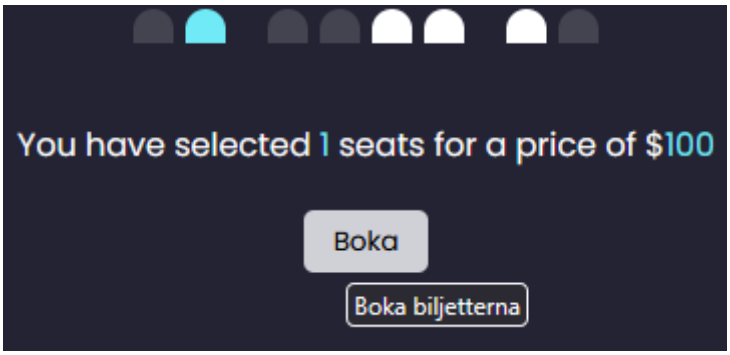
Sen tyckte jag att den feature-branchen var klar, så mergade branchen '**feature/movie-selection**' till branchen '**dev-react**'.

Skapade en ny branch '**feature/booking**'. Lade till en knapp 'Boka' under paragrafen som visar antalet säten och priset, som var disabled om antalet valde säten är 0, men är klickabar om antalet säten är över 0.

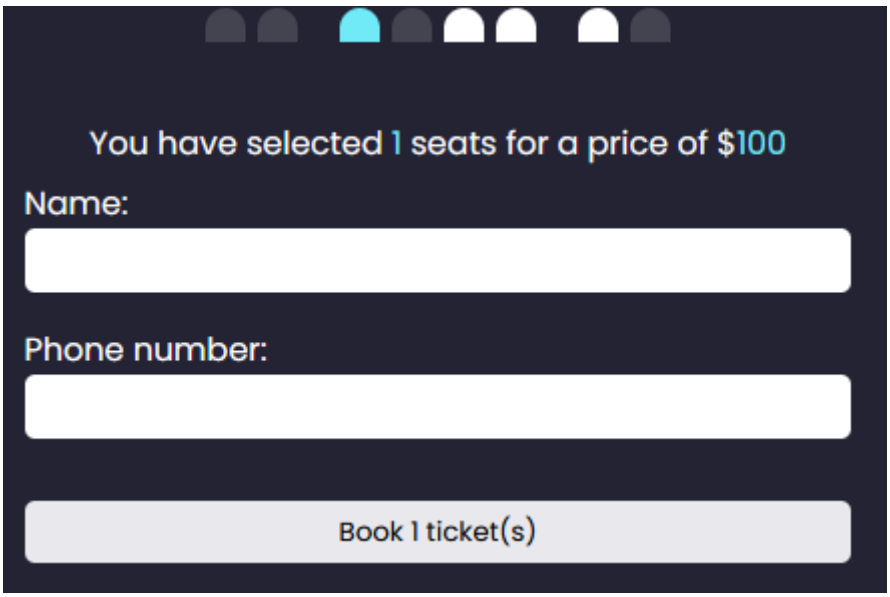
När antalet är 0 så visas även denna titel-text när man hovrar över knappen, nu syns inte min muspekare men den är av type 'not-allowed' när musen är över.



När det är ett eller flera säten valda (här hovrar jag med musen över, och då är muspekaren av typen ’pointer’):



Sen kom jag på att resten av sidan var skriven på engelska så översatte allt ovan till engelska. Sedan skapade jag ett form-element under boka-knappen som visas när man trycker på ’Book Tickets’- knappen, och då döljs ’Book Tickets’ knappen och detta visas:



27e januari

Men sen upptäckte jag en potentiell bugg, att man kan ju fortfarande markera och avmarkerar säten ovanför detta och om man då avmarkerar alla säten och har valt 0 säten så kunde man fortfarande klicka på knappen som visas i bilden ovan eftersom det är en helt ny knapp.

Så då var jag tvungen att lägga till checkar även på denna knapp så man inte kan trycka på den om antalet är 0. Då skapade jag en ny komponent ’BookingButton’ eftersom jag behövde upprepa mig. Så försökte göra den så generisk som möjligt i bokningssammanhanget, jag

skickade med en boolean om den är klickbar eller ej, skickar med en funktion som kan anropas vid klick samt children property så att jag kan välja vad som ska stå på själva knappen.

```
<BookingButton
  isClickable={selectedSeatsCount > 0}
  onClick={displayForm}
>
  Book Tickets
</BookingButton>
```

← Den första knappen som syns under sätesraderna

Andra knappen som syns när man har ”öppnat” formuläret →

(Med en temporär alert-funktion)

```
<BookingButton
  isClickable={selectedSeatsCount > 0}
  onClick={() => alert('Booking')}
>
  Book {selectedSeatsCount} ticket(s)
</BookingButton>
```

Sen funderade jag på om jag skulle ha gjort som en helt ny ”sida” för bokningsformuläret och använda mig av React Router. Så att man inte kunde markera och avmarkera när formuläret visades, och då göra det lite snyggare med att visa vilka sätesnummer man hade valt och priset igen och sedan formuläret. Men jag behåller som jag gjorde det ovan och kan ändra till det sättet sen om jag känner att jag har tid.

Sedan började jag på funktionaliteten med att validera bokningsformuläret, jag gjorde onClick-propertyn i komponenten BookingButton valbar eftersom den andra bokningsknappen hör till formuläret så kommer den inte behöva en onClick på sig själv utan eventet kommer hanteras av formulärets onSubmit-funktion i stället.

Skapade en funktion ’*onBookingSubmit*’ i App.tsx som kommer skickas ner som en property till BookingSummary och kommer triggas vid en onSubmit på formuläret. Men sen hade jag några problem med hur jag faktiskt skulle validera formuläret, eftersom jag inte kunde göra som jag gjorde i tidigare projekt i vanilla JavaScript. Så fastnade där ett tag för, det kom upp en mängd olika förslag på hur man gör detta i React, hittad ett bibliotek som hette React Hook Form. Men hittade också ett tillvägagångssätt där varje inputfält fick en egen useState som man satte värde på när inputfältet ändrades. Men sen hittade jag att det fanns något som hette useFormState men att den hade bytt namn i React 19 till useActionState. Läste i Reacts dokumentation hur useActionState fungerade, men det var fortfarande inte helt tydligt hur jag skulle göra. Sedan hittade jag denna video som förklarade useActionState:

https://www.youtube.com/watch?v=-aBKrvK5Vn8

Valde att använda useActionState för det såg faktiskt minst komplicerat ut för mig och mindre och mer kompaktare kod. Så färdigställde HTML/JSX-strukturen för formuläret, lade till nödvändiga attribut på inputfälten som; type, id, name, required,placeholder, men även pattern och titel för att använda det redan befintliga clientvalideringen som kommer med webbläsren. Jag använde regex-pattern för att säkerställa att namnet inte innehåller siffra, eller andra specialtecken eller att det är tomt, och att telefonnummret är skriven enligt formatet för ett svenskt mobilnummer (antingen börjar med +46 (eller 00 istället för ’+’) eller börja med 0). Då

tog jag bort onSubmit från formuläret och implementerade useActionState för att hantera formuläret istället. Om webbläsarens klientvalidering av någon anledning inte fungerar så valideras inputen igen i funktionen '*onBookingSubmit*', även där med regexPatterns och returnerar errors om fel skulle uppstå. Om så är fallet kommer ett rött felmeddelande under inputfältet att visas och inputfältens border blir röd.

Jag lade även till i funktionen '*onBookingSubmit´* att om inga fel hade uppstått så returneras ett *success message* istället, där tanken är att jag kan använda det för att se om bokningen har lyckats eller ej.

Sedan hittade jag en bugg med min komponent SeatLegend att Seat-komponenterna där inte hade alla obligatoriska properties. Så redigerade Seat-komponenten med att göra properties valbara för propertiesena: setSelectedSeatsCount, ticketPrice, currentSeatCount, setTotalPrice, row, seatNumber, setSelectedSeats. I funktionen '*handleClick*' så returnerar jag tidigt om någon av dessa är undefined, för att de krävs för att klick-logiken ska fungera när man markerar ett säte.

Insåg när jag började med funktionen som ska göra ett POST-anrop till bookings-databasen att jag som trodde att jag hade tänkt igenom så nog hur jag skulle strukturera upp dens json-text inte alls var så bra. För nu kommer jag inte åt en specifik bokning med URLen "*http://localhost:3000/bookings/the-lion-king*" om jag vill göra en ändring på en specifik film. Så justerade min db.json specifikt bookings-delen till att samlas i en array istället för objekt:

```
"bookings": [
  {
    "id": "the-lion-king",
    "a": [],
    "b": [
      {
        "seat": 4,
        "customer": { "name": "John Doe", "phone": "070 123 45 67" }
      },
      {
        "seat": 5,
        "customer": { "name": "John Doe", "phone": "070 123 45 67" }
      }
    ],
    "c": [
      {
        "seat": 7,
        "customer": { "name": "Jane Smith", "phone": "070 123 45 66" }
      },
      {
        "seat": 8,
        "customer": { "name": "Jane Smith", "phone": "070 123 45 66" }
      }
    ],
    "d": [],
    "e": [
```

Fick då redigera alla interface som berörde bookings för att matcha den nya strukturen, passade även på att exportera interfacena som default istället. Döpte om interfacet Booking till

CustomerBooking för att göra den mer tydlig. Jag skapade även en ny interface: MovieBooking:

```
import type CustomerBooking from './CustomerBooking';

export default interface MovieBooking {
  id: string;
  a: CustomerBooking[];
  b: CustomerBooking[];
  c: CustomerBooking[];
  d: CustomerBooking[];
  e: CustomerBooking[];
  f: CustomerBooking[];
}
```

Det är väl denna som egentligen ersätter SeatBookings-interfacet. Men valde att behålla SeatBookings-interfacet och justera det till följande:

```
1   import type MovieBooking from './MovieBooking';
2
3   export type SeatBookings = MovieBooking[];
```

Dels för att då behövde jag inte ändra typ-namnet i App.tsx i useStaten, bara byta ut default-datatypen från objekt till array:

```
23-  const [bookings, setbookings] = useState({} as SeatBookings);
23+  const [bookings, setbookings] = useState([] as SeatBookings);
```

Detta gjorde det ändå tydligare i min hjärna hur typen är strukturerad, för jag hade lika hjärna kunnat skriva att defaultvärdet var:

```
[] as MovieBooking[]
```

Men att göra om befintliga typen SeatBooking gjorde det tydligt för mig att här samlas alla sätesbokningar, då en array av MovieBooking låter lite mer generellt och inte specifikt att det syftar på sätena.

28e januari

Eftersom jag inte använde parametern '*previousState*' i funktionen onBookingSubmit ändrade jag dess namn till _ för att discarda den.

Jag började med en funktion *saveBookingData* i filen bookings.ts i mappen api, men då blev jag orolig igen att jag hade strukturerat upp min bookngs-databas på fel sätt igen för jag kan inte göra en POST i detta fall när jag vill spara en ny bokning för en film. Utan som jag har strukturerat upp det nu är en POST när jag vill skapa en helt ny boknings-post / resurs för en ny film man lägger till (i det kommande Admin-sidan).

Så jag funderade om jag skulle använda PUT, men slutade med att PATCH var det bästa alternativet, för jag uppdatera en befintlig bokning samt att jag då endast behöver uppdatera de

rader som har valts och behöver inte skicka med hela boknings-objektet för en film som jag behöver göra med PUT. Då kan jag bara skicka med de rader som har förändrats om jag använder PATCH.

Började lägga till funktionalitet i funktionen *'onBookingSubmit'* som loopar igenom de valda sätena (useStaten selectedSeats) och om det inte finns någon vald på en rad hoppar den till nästa iteration. Men stötte på problem:

```
Problem: " Element implicitly has an 'any' type because expression of
type 'string' can't be used to index type 'MovieBooking'.

  No index signature with a parameter of type 'string' was found on
type 'MovieBooking'."
```

```
79      for (const [row, seatNumbers] of Object.entries(selectedSeats)) {
80        if (seatNumbers.length == 0 || bookingsForSelectedMovie === undefined) {
81          return;
82        }
83
84        console.log(bookingsForSelectedMovie[row]);
85      }
```

Tänkte att det var för att det inte specifik visar att det kan vara någon av följande strängar: "a" | "b" | "c" | "d" | "e" | "f", så provade att skapa en Row-type, baserat på nyklarna i typen MovieBooking fast att inte inkludera nyckeln "id".

```
import type MovieBooking from './MovieBooking';

export type Row = Exclude<keyof MovieBooking, 'id'>;
```

Och typa selectedSeats useStaten till att det är ett objekt där nycken är av typen Row och värdet är en array av nummer.

```
18    const [selectedSeats, setSelectedSeats] = useState<Record<Row, number[]>>({
19      a: [],
20      b: [],
21      c: [],
22      d: [],
23      e: [],
24      f: [],
25    });
```

Men nu insåg jag att jag kanske har typat mina useStates felaktigt med att använda nyckelordet as, att då overridar jag typescripts typning och castar om hela objektet till den typen och då plockar inte TypeScript upp och ger mig varningar om jag hade skrivit in ett felaktigt datatyp. (som jag uppfattade det).

Men fick fortfarande samma error när jag försökte iterera över med hjälp av Object.entries. Men sen hittade jag att det fanns Object.keys så använde det istället. Hade problem med att en variabel möjligtvis var undefined (bookingsForSelectedMovie) men insåg att jag hade ju skapat denna funktion före jag deklarerade denna variabel, så flyttade ner denna funktion under variabeldeklarationen och det fixade problemet.

Sedan itererade jag igenom useStaten selectedSeats och för varje rad kollar jag om det finns valda säten i den raden, om inte så hoppar den till nästa nyckel. Annars mappar jag över alla valda säten i den raden och för varje valt sätesnummer skapar jag ett objekt av typen CustomerBooking. Men hade ett problem med att den raden jag faktiskt hade valt säten på hoppades över och jag hade lite problem med typningarna och sådär som jag satt med i evigheter till jag insåg mitt misstag. Jag hade skrivit fel variabel i if-satsen. Jag hade skrivit

```
for (const row of Object.keys(selectedSeats) as Row[]) {
  console.log(row.toUpperCase());
  if (
    bookingsForSelectedMovie === undefined ||
    bookingsForSelectedMovie[row].length == 0
  ) {
    continue;
  }
```

```
for (const row of Object.keys(selectedSeats) as Row[]) {
  console.log(row.toUpperCase());
  if (
    bookingsForSelectedMovie === undefined ||
    selectedSeats[row].length == 0
  ) {
    continue;
  }
```

För nu hoppade den över raden om det inte fanns redan EXISTERANDE bokningar på den raden. Så min justering rättade till logiken rejält.

Jag implementerade att göra ett PATCH-anrop med de uppdaterade raderna för den specifikt valda filmen till json-server databasen. Implementerade en till useState 'currentPage' som jag skapade en typ för där värdena kunde vara "booking", "thank-you" eller "admin", där den defaultar till 'booking' och gränsnittet för sätena med pris etc visas bara om currentPage är 'bookin'. Sedan gjorde jag en enkel JSX markup för et tack sid som endast visas om currentPage är 'thank-you'. Denn hade en knapp som tar tillbaka användaren till "boknings-sidan". Denn knapp har en funktion onClick sätter useStaten currentPage till 'booking' och återställer useStaten för valda säten, antalet säten och pris samt sätter valda filmen till den första filmen i listan.

Men nu stötte jag på ett till problem, att jag måste uppdatera useStaten bookings där alla bokningar från databasen sparas till när komponenten mountar. Först tänkte jag att jag behövde göra ett GET-anrop från databasen igen, men då funderade jag på att jag skulle behöva refaktorisera koden liten genom att endast hämta bokningar för den valda filmen när sidan laddas och när man byter film så görs ett nytt GET-anrop till databasen för den nya valda filmen. Detta kommer medföra fler anrop till databasen, men applikationen behöver håller mindre data i

minnet i den useStaten som sparar bokningarna (så inte hela databasen sparas i den). Vilket jag kände var ett mycket bättre alternativ, men då behövde jag ändra så mycket och jag har redan lite stress med att hinna färdigt. Så jag uppdaterade bokningarna med usestate-funktionen setBookings, där jag lade in de nya bokade sätena och detta gjorde jag i funktionen, jag hade problem med att uppdatera bookings korrekt i setBookings med att använda spread-operatorn och byta ut specifika nycklars värden med de uppdaterade raderna (som hade ändrats), men sen hittade jag denna länk: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object/assign

Som kändes vara precis vad jag behövde.

```
const updatedBookingsForMovie = Object.assign(
  bookingsForSelectedMovie,
  newBookingData,
);
```

Men upptäckte sen att den muterade useState-variabeln bookings, och tekniskt sätt så funkar det, då sätena var markerade som bokade när jag tryckte på knappen 'Back to booking'. Men eftersom man ska uppdatera useStaten med dens setter funktion så försökte jag hitta ett annat sätt.

Provade att ha den nya objektet '*newBookingData*' som target (som första parameter) men när jag consol-loggade så hade bookings-useStaten uppdaterats ändå. Men efter mycket om och men och testande hit och dit så hittade jag denna länk, åter igen från React-dokumentationen:

https://react.dev/learn/updating-arrays-in-state#updating-objects-inside-arrays

Men jag kom något närmare till hur det ska vara, men ändå fel:

```
161       setBookings(
162         bookings.map((movieBooking) => {
163           if (movieBooking.id === selectedMovie.id) {
164             return {
165               ...movieBooking,
166               newBookingData,
167             };
168           }
169           return movieBooking;
170         }),
171       );
172     }
```

```
id: "rampage"
▸ a: [{…}, {…}]
▸ b: [{…}, {…}]
▸ c: []
▸ d: []
▸ e: []
▸ f: []
▾ newBookingData: {f: Array(4)}
  ▸ f: [{…}, {…}, {…}, {…}]
    new entry: ""
new entry: ""
```

29e januari

Vaknade och tänkte att detta i bilden nedan skulle vara den enkla lösningen på mitt problem. Men så var icke fallet det gav exakt samma resultat fast det tillagda objektet nyckel var istället "movieBooking".

```
161         setBookings(
162           bookings.map((movieBooking) => {
163             if (movieBooking.id === selectedMovie.id) {
164               return {
165                 ...movieBooking,
166                 movieBooking: newBookingData,
167               };
168             }
169             return movieBooking;
170           }),
171         );
```

Sen kom jag på denna ”ful-lösning” nedan, den är inte så snygg, ville ha den mer ”dynamisk, men jag satt och provade så många olika varianter som inte fungerade att jag fick tillslut nöja mig med denna lösning nedan tyvärr för att slå ihop de 2 objekten:

```
setBookings(
  bookings.map((movieBooking) => {
    if (movieBooking.id === selectedMovie.id) {
      return {
        ...movieBooking,
        a: newBookingData.a,
        b: newBookingData.b,
        c: newBookingData.c,
        d: newBookingData.d,
        e: newBookingData.e,
        f: newBookingData.f,
      };
    }
    return movieBooking;
  }),
);
```

Sen blev jag så arg på mig själv att jag inte hade googlat ”*javascript merge objects spread operator*” tidigare för där hittade jag lösningen på problemet jag hade under en pinsamt lång tid igår och nu.

```
setBookings(
  bookings.map((movieBooking) => {
    if (movieBooking.id === selectedMovie.id) {
      return {
        ...movieBooking,
        ...newBookingData,
      };
    }
    return movieBooking;
  }),
);
```

TA DA :D

Mergade branchen ’**feature/booking**’ till branchen ’**dev-react**’. Skapade en ny branch ’**feature/admin-layout**’.

Skapade en knapp som byter useStaten currentPage till ’admin’, och på admin sidan finns det en knapp ’<- Back’ som sätter currentPage till ’booking (med setCurrentPage).

Visar alla filmer i en ul-lista. Skapade en Delete-knapp som jag lade bredvid filmens namn och pris.

Skapde funktionalieten för delete-knappen, när knappen Delete klickas så uppdateras useState och ett DELETE-anrop görs till movie-databasen och bookings-basabasen som tar bort all information om den borttagna filmen.

Skpade en EditButton.tsx komponent:

```tsx
type EditButtonProps = {
  onEditMovie: (movieId: string) => Promise<Boolean>;
  movieId: string;
};

export const EditButton = ({ movieId, onEditMovie }: EditButtonProps) => {
  return (
    <button className="btn" onClick={() => onEditMovie(movieId)}>
      Delete
    </button>
  );
};
```

Men sedan refaktoriserade jag lite, eftersom Delete och Editknappen är i princip identiska bara annan text och vad för typ av action den utför så skapade jag en ny komponent, **ActionButton.tsx** istället. Dels för att inte upprepa samma kod igen (DRY) men också så att komponenterna blir mer återanvändingsbara och enklare att underhålla, om man kanske vill lägga till ikon eller liknande.

Så bytte ut DeleteButton och EditButton komponenterna mot ActionButton:

```tsx
type ActionButtonProps = {
  onAction: (movieId: string) => Promise<Boolean>;
  movieId: string;
  className?: string;
  children: React.ReactNode;
};

export const ActionButton = ({
  onAction,
  movieId,
  className = 'btn',
  children,
}: ActionButtonProps) => {
  return (
    <button className={className} onClick={() => onAction(movieId)}>
      {children}
    </button>
  );
};

```

Ville göra det enkelt för mig och inte ha olika ”sidor” för att redigera och lägga till en film. Så jag ville göra att när man trycker på Edit så ska dom omvandlas till 2 inputfält där man kan redigera filmtiteln och priset. Och behövde då ha någon useState som indikerar om den är editer-bar eller inte. Men som jag har satt upp det nu så kan jag inte ha en enkel boolean useState för då kommer alla filmer bli redigeringsbar i listan om jag trycker på edit. Så så hade jag 2 alternativ, antingen skapa en komponent av li-elementet som har en egen useState ”isEditable” eller liknande. Eller så får jag ha ett lite mer komplext useState med objekt där nyckeln är film-ID och värdet en boolean. Men bestämde mig för att extrahera ut till en egen komponent *’MovieItem*.tsx’ för att då får varje komponent sin lokala useState, om den är rediger-bar eller ej och det blir lite enklare logik för state-hanteringen.
Flyttade över JSX-markupen till *MovieItem*-komponenten men fick justera propsen, lade till propsen *onEditMovie* och *onDeleteMovie*:

```tsx
export const MovieItem = ({
  movie,
  onEditMovie,
  onDeleteMovie,
}: MovieItemProps) => {
  const [isEditable, setIsEditable] = useState(false);

  return (
    <li className="movie-item" key={movie.id}>
      <span>
        {movie.name} - {movie.price}kr
      </span>
      <span className="action-group">
        <ActionButton movieId={movie.id} onAction={onEditMovie}>
          Edit
        </ActionButton>
        <ActionButton movieId={movie.id} onAction={onDeleteMovie}>
          Delete
        </ActionButton>
      </span>
    </li>
  );
};
```

Först såg jag till att funktionaliteten fungerade som det gjorde innan, sedan lade jag till funktionaliteten för Edit.

”Problemet” (se bilderna ovan) jag stötte på då var att listan blev bredare p.g.a input-fälten satt ett bra tag och försökte lista ut hur jag skulle få till dom lika breda som filmtiteln. Men sen fick det vara som det är till höger i bilden ovan trots allt, det är *good enough*.

Lade till en useState editableMovie, som gör det möjligt att för användaren att visa det den skriver i inputfältet, dess värde som sen också kommer skickas vidare med till funktionen onSaveEditMovie för att kunna uppdatera movies-useStaten i App.tsx samt att skicka en PATCH/PUT anrop till databasen med denna nya data.

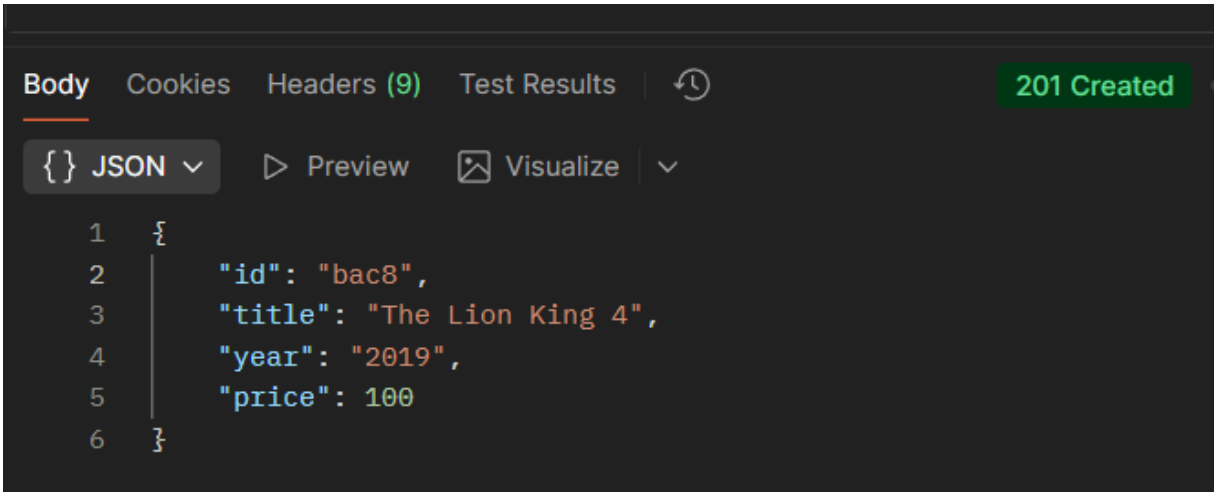
Implementerade funktionalitet för att avbryta redigerings-läget, genom att återställa editableMovie -useStaten till dess ursprungliga state (propertyn movie) och växla isEditable till false. Likadant gjorde jag för pris-inputen.

När användaren trycker på Save jämför jag först om det är någon förändring mot de initiala värdena, endast om det är en förändring görs ett PATCH-anrop till movie-databasen.

Men nu stötte jag på ett nytt problem, som åter igen är baserat på mina tidigare (dumma val), att eftersom filmens ID var baserat på filmtitel så känns det som att jag måste ändra IDt om filmtiteln updateras för annars om jag skulle döpa om The Lion King till The Lion King 2 och sedan skapa en ny film som heter The Lion King så kommer IDt 'the-lion-king' redan finnas kopplad till The Lion King 2 exempelvis. Så nu funderar jag ändå att sätta filmens ID till exempelvis Date.now() värde när en ny film skapas. Funderade också bara på att ha 1, 2, 3 etc. Men då måste jag hela tiden kolla vilket som är IDt på den senaste tillagda filmen, så kommer nog ändå implementera att ha Date.now() som ID.

30e januari

Men efter mer fundering så vill jag kanske ändå inte inte ha Date.now() som ID, eftersom jag påtalade tidigare i loggboken så är inte "säkert" att det blir ett helt unikt ID. Så funderade om det finns något liknande som GUID i C# eller om jag ska ha (som jag också tidigare har nämnt) 1,2,3 etc. Men då måste jag hela tiden kolla vilket ID den längst ner i listan har, men eftersom alla filmer redan finns sparade i en useState så kanske det alternativet är det bästa ändå. Sen hittade jag att json-server automatiskt genererar ett ID om det fältet utelämnas. Men jag hade problem med att det inte genererades en siffra utan en random sträng, även om jag ändrade befintliga Ids till siffror så fick jag tillbaka detta när jag gjorde ett test-anrop med Postman:



Så jag bestämde mig för att ändå använda alternativet där jag får titta vad senaste IDt är på filmen längst ner i listan.

Men nu gick jag tillbaka för att färdigställa uppdatera film funktionaliteten med att även uppdatera useStaten för filmerna. Gjorde på samma sätta som jag hade uppdaterat bookings, med spread-operatorn. Men stötte på problem med att det endast var priset som uppdaterade.

Men fattade sen att det var för att nyckeln för filmens titel sparas med nyckelordet ’title’ men i Movie-klassen så är den omdöpt till ’name’ och det är därför bara priset uppdateras i useStaten när jag gjorde på samma sätt:

```
Object { title: "The Lion King 3", price: 150 }
  price: 150
  title: "The Lion King 3"
```

Till slut kom jag på en lösning med att använda spread-operatorn och rest-operatorn och objekt destrukturering för att filtrera bort titel och lägga dit namn-nyckel istället (om filmtiteln har uppdaterats):

```
301     if (updatedData.title) {
302       const { title, ...filteredData } = updatedData;
303       useStateData = {
304         ...filteredData,
305         name: updatedData.title,
306       };
307     } else {
308       useStateData = {
309         ...updatedData,
310       };
311     }
```

Hittade även en bugg att om man tar bort filmen som är den aktivt valda filmen, kom man till skärmen ’Loading...’ för selected movie var undefined. Så i funktionen som uppdaterar useStaten efter att användaren tryckte ’Delete’ sätter jag en ny selected Movie som inte är den borttagna filmen.

Då kom jag att tänka på en till bugg, att om man tar bort alla filmer så kommer man komma till ’Loading...’ sidan igen eftersom selectedMovie är undefined eller null i detta läge. Så lösningen på detta problem var att jag fick gå tillbaka till min lösning som jag hade från början med att ha en isLoading-useState som jag tog bort förut. För att om det inte finns några filmer i databasen ska man inte kunna välja några säten och texten ”No movies to book at the moment” ska visas istället för ’Loading...’.

Sen implementerade jag funktionalitet för att lägga till en ny film. Skapade en ny komponent ’AddMovie’ som hade en ActionButton-knapp. När man trycker på den så försvinner den och 2 inputfält och en saveknapp visas. Jag implementerade även felhantering men hjälp av regex pattern för klient-validering sedan validerade jag även om det inmatade filmnamnet redan existerar i de befintliga filmerna. Om så är fallet visas detta:

När inget fel har uppstått så skickas ett POST-anrop till movies databasen och bookings databasen och skapar en ny ”entry” eller hur man ska kalla det för denna film och useStaten uppdateras med den nya filmen.

Körde kommanto ***npm run lint*** för att se om det fanns några ESlint errors, jag hittade några och åtgärdade dom. Det var ett fel som jag inte kunde fixa, för den klagade på att en variabel inte användes, men det var när jag filtrerade ut nyckeln ’title’ med rest-operatorn och jag ändrade variabel namnet till ’_’ för att se om det fungerade. Men det tog inte bort ESlint felmeddelandet tyvärr.

Hittade en bugg att om man ändrar priset på en film och går tillbaka till bokningssidan så används fortfarande det gamla priset om det är den ”valda filmen” som redigeras. Åtgärdade det.

Mergade branchen ’**feature/admin-layout**’ till ’**dev-react**’

Mergade **dev-react** till **main.**