CS 312 Software Development

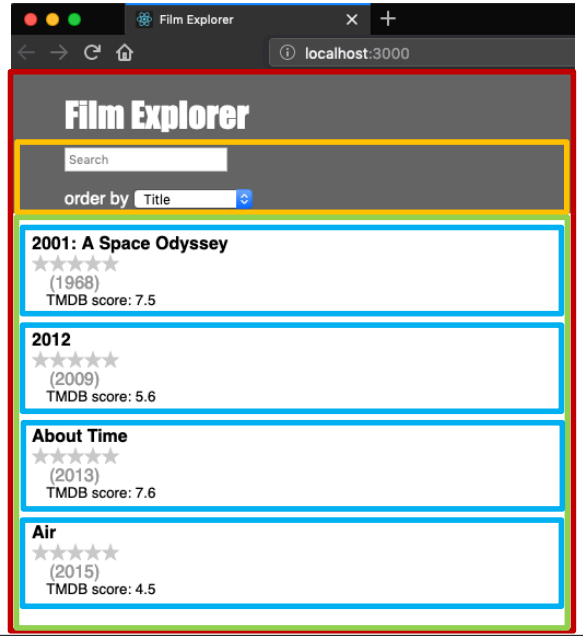
Designing in React

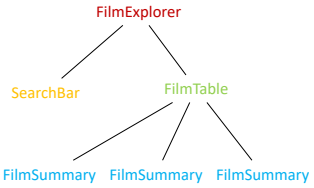
Recall: “Thinking in React”

1. Break the UI into a component hierarchy
2. Build a static version in React
3. Identify the minimal (but complete) representation of state
4. Identify where your state should live
5. Add “inverse” data flow (data flows down, callbacks flow up)

https://reactjs.org/docs/thinking-in-react.html

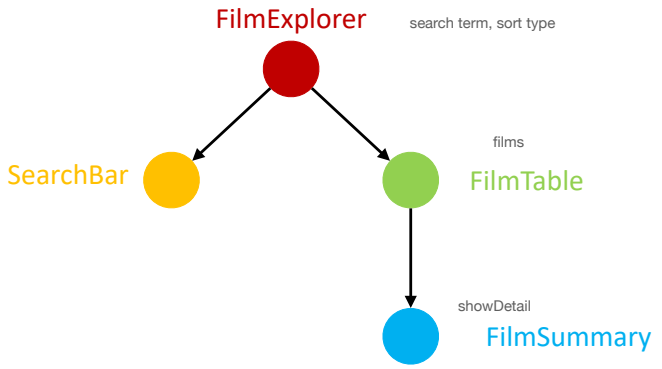
What is the component hierarchy?





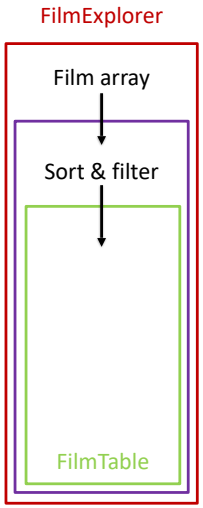
Review: React state placement

What state do we need, and where should it be stored?



Recall data flows “down” via props and data flows “up” via callbacks

Container components: Separating logic from UI

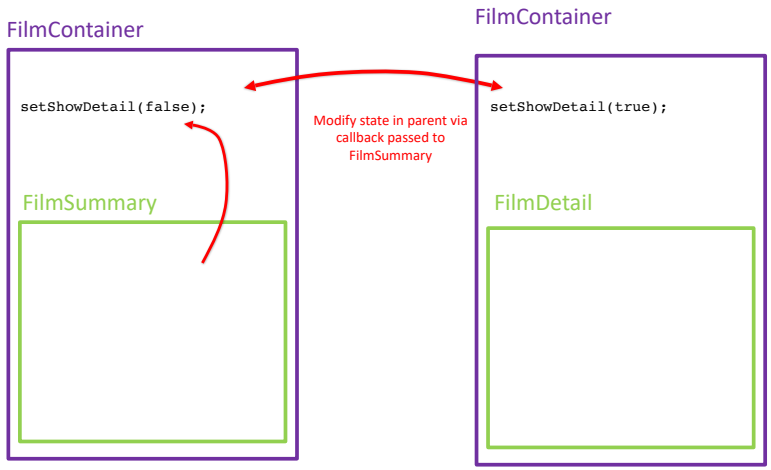


Separation of Concerns:

- *Container Component (CC):* Concerned with how the application works, i.e. implements logic
- *Presentation Component (PC):* Concerned with how the application looks. Typically generates DOM.

*"Remember, **components don't have to emit DOM.** They only need to provide composition boundaries between UI concerns."* Dan Abramov

CC applied: FilmContainer



Working with React

Conditional rendering

```
function FilmContainer (props) {
  const [showDetail, setShowDetail] = useState(false);
  if (showDetail){
     return <FilmDetail {…props} onClick={()=> setShowDetail(false)} />
   } else{
     return <FilmSummary {…props} onClick={()=> setShowDetail(true)} />
   }
}
```

```
function FilmContainer (props) {
  const [showDetail, setShowDetail] = useState(false);
  const View = showDetail ? FilmDetail : FilmSummary;
  return (
    <View {...props} onClick={()=>{setShowDetail(!showDetail);}} />
  );
}
```

Some common conditional patterns:

```
{boolean && <Component … />}
{boolean ? <Component1 … /> : <Component2 … />}
```

Working with React

Sequences

```
function FilmTable({ films, setRatingFor })
{
  const keyedFilms = films.map(film => (
    <FilmContainer
      key={film.id}
      {...film}
      setRatingFor={setRatingFor}
    />
  ));
  return <div>{keyedFilms}</div>;
}
```

"Arrays" need a key to uniquely identify components

"Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity. Most often you would use IDs from your data as keys" -ReactJS Docs

Working with React

Mutating data

What might go wrong with this code?

```
const [films, setFilms] = useState([]);
…
const setRating = (filmid, rating) => {
  const index = films.findIndex((film) => film.id === filmid);
  films[index].rating = rating;
  setFilms(films);
}
```

films is the same object — this may not trigger a re-render since it doesn't appear anything has changed

Don't mutate state or props objects!

Working with React

Don't mutate, make copies

map() creates a new array

```
const setRating = (filmid, rating) => {
  const newFilms = films.map((film) => {
    if (film.id === filmid) {
      // or return Object.assign({}, film, { rating: rating});
      return { ...film, rating };
    }
    return film;
  });
  setFilms(newFilms);
}
```

creates a new object instead of mutating it