Web-school

Home task

React

Navigation

Definition

The idea is to create a single page application, which will allow users to search, add and delete movies from the Movies DB database.

Local Movies API Service

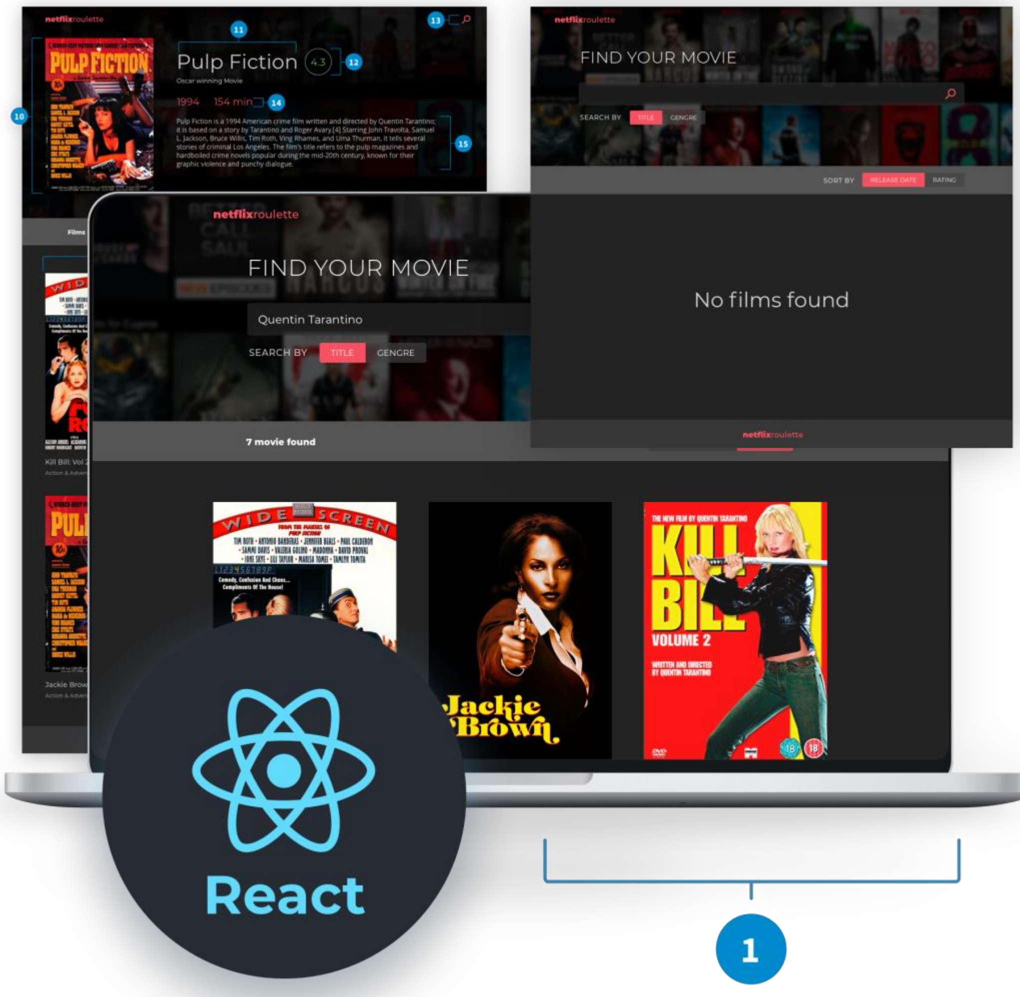
GitHub Repository with local Movies API Service implementation:

https://github.com/VarvaraZadnepriak/MoviesAPI.ReactJS

To start local Movies API service, you should:
1. Clone Movies API Service repository
2. Run the following command to install node modules:
```
npm i
```
3. Run the following command to start Movies API Service:
```
npm start
```
Movies API Service will be available on http://localhost:4000/.
API Swagger documentation will be available on http://localhost:4000/api-docs.

Interface

Design is available via InVision prototype. **There's no need to implement pixel perfect responsive design. Design can be implemented schematically.**



! NOTE
Each mark in evaluation criteria includes previous mark criteria

Task 1: Core concepts

Setting up the environment

Create a repository. Install an application created via `create-react-app.` Commit the changes to repository. Install extensions mentioned in the lecture (react developer tools, etc.)

Create components in different ways, using:

```
1.   React.CreateElement
2.   React.Component
3.   React.PureComponent
4.   Functional component
```

NOTE

!

To implement this task you could use react online sandbox [optional], or start to configure basic project build from task 2. Share sandbox link with your mentor or commit to your repository

Evaluation criteria*

2	Install blank `create-react-app` application
3	Render blank message (Hello World) with `React`
4	Use at least 2 methods of creating react components
5	Use all methods which mentioned in task, to create `React` components, installed and used all extensions

*** Each mark includes previous mark criteria**

Task 2: Webpack

Create `package.json` file and install `React, Redux, React-Redux, React-Router, Jest.` Install and configure `Webpack` & `Babel` to get build artifact by running `npm` command.

Set DEV and PROD build configuration. Use env variables, dev server, optimizations for `PROD build.` Set up testing. You should have test command in your `package.json` file, which will run your future tests. **Don't use any React boilerplate (like `create-react-app`) for this task**

Evaluation criteria*

2	Installed `React, Redux, React-Redux, React-Router, Jest`
3	Configured `Webpack`
4	Configured `Babel`. Configured tests script
5	Have DEV and PROD build commands (use env variables)

*** Each mark includes previous mark criteria**

Task 3: Components part 1

Write components implementing HTML markup for required design **for home page** of **InVision prototype (Only UI part)**. For this part, no need to implement API calls and routing, the task can be done with mocked data.

Use `<ErrorBoundary>` component for catching and displaying errors https://reactjs.org/docs/error-boundaries.html. You could create one component and wrap all your **application** or use several components.

Evaluation criteria*

2	Markup is done with `React Components and React Fragments` (parent-child)
3	Apply styles (no need to do pixel perfect and strict colors following)
4	Use `PropTypes`
5	Use `<ErrorBoundary>` component for catching and displaying errors

*** each mark includes previous mark criteria**

Task 4: Components part 2

Implement markup and styles for “**Add movie**”, “**Edit**”, “**Delete**” modal windows and “sorting”. **No need to implement real API calls. Only add pages with mocked data**. No need to implement hooks in this task.

Evaluation criteria*

2	Implementation of markup and styles
3	Use stateless/stateful approach
4	Use React synthetic events
5	Use lifecycle methods (discuss with mentor)

*** each mark includes previous mark criteria**

Task 5: Hooks

Implement markup and styles for “Movie details” page.

In your project, change Class components into Functional components and use hooks where applicable.

Evaluation criteria*

2	Implement “Movie details” page. Use “`useState`” hooks
3	Use “`useCallback`” hooks
4	Use “`useEffect`” hooks
5	Usage of custom hooks (discuss with your mentor)

*** each mark includes previous mark criteria**

Task 6 (Redux)

1. Go through API docs in swagger: http://localhost:4000/api-docs
2. Make your components perform real AJAX requests. Implement data fetches as async actions and pass data to your components with `redux`.
3. Implement creating, editing and updating films according to the design operations as redux actions.
4. Implement filtering and sorting (by genre, rating, and release date) as redux actions.

Evaluation criteria*

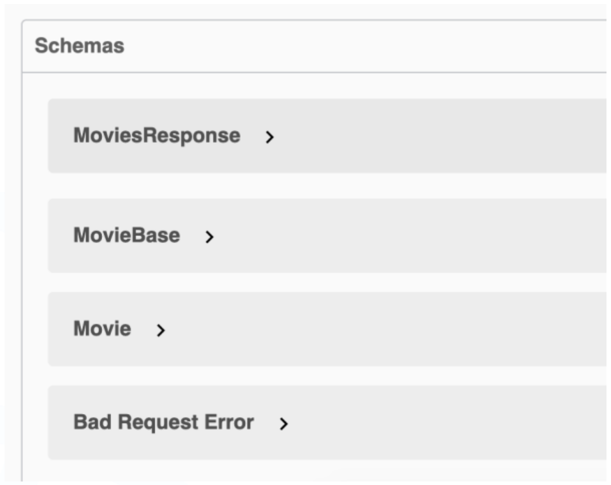
2	All data fetches done as `actions` and received from store by components
3	Creating, editing, deleting and updating (CRUD operations) of films are done as `redux` actions.
4	Filtering by release date and rating done as `redux` actions.
5	Sorting by genre is done as `redux` actions

*** each mark includes previous mark criteria**

Task 7 (Forms)

1. Implement «**Add movie**» form.
2. Implement «**Edit movie**» form.

Required validation criteria: http://localhost:4000/api-docs



Evaluation criteria*

2	Installed `formik`
3	Implementation of "Add movie" form with validation
4	Implementation of "Edit movie" form with validation
5	Use hooks from `formik`

*** each mark includes previous mark criteria**

Task 8 (Routing)

Create routing for your application (via `React Router`).

Add **404** (error page) and **‘No movies found’** page.

Link app states between each other with `React Router`.

By default, user lands on a new page with empty results state

When user clicks on a film item, redirect them to:

```
localhost/film/id
```

Handle invalid URLs, display a 404 page, where user will be redirected in case of invalid URL. On switching search type or sorting type you shouldn’t switch any routes.

When user performs a new search, you should redirect them to:

```
localhost/search/Search%20Query
```

When a new user lands on the page with such URL, you should perform search and display results.

Evaluation criteria*

2	Add 404 and “No movies found” pages with markup Have 2+ pages which displays on different URLs
3	Implement displaying 404 page for all invalid URLs
4	By default, user lands on a new page with empty results page
5	- When user performs a search on the page, change URL and show search results - When new user enters the site using direct link with search parameters – show search results

*** each mark includes previous mark criteria**

Task 9 (Testing lecture)

Write unit tests for your application (consider using `Jest, @testing-library/react` or `Enzyme, react-test-renderer, React-test-utils, etc.`)

Subtasks:

1. Cover 1 simple presentational component with snapshot tests
2. Cover 1 reducer and all its actions with unit-tests
3. Measure coverage level with coverage report
4. Cover “Add movie” modal dialog components with unit-tests, mock all external dependencies using Jest mocks

Evaluation criteria*

2	Subtask 1 is implemented
3	Subtasks 2,3 are implemented
4	Subtask 4 “Add movie” modal dialog and all its components coverage > 70%
5	Global coverage > 90% Add unit tests for hooks

*** each mark includes previous mark criteria**

Task 10 (Server-Side Rendering)

Implement server rendering in your NodeJS application.

Use async actions, `redux` should provide initial state from server.

Server should handle query parameters to compute correct initial state.

Implement route masking. `Next.js` provides this functionality out of the box, but its usage is not required. Example of route masking:

```
localhost/search/Hello%20Friend
localhost/search/Search%20Query
```

When a new user lands on the page with such URL, you should perform search and display results.

Add code splitting to your app. You can use dynamic **import()** syntax in conjunction with webpack and React.lazy, some library (react-loadable), or next.js.

Evaluation criteria*

2	Async actions, `redux` provides initial state from server
3	Server app handles query params to compute initial state
4	Route masking. Can be implemented, or mentee can switch to `next.js` for SSR
5	Code splitting for optimized performance. Goes out of the box with `next.js`

*** each mark includes previous mark criteria**

Task 11 (Patterns, tips & tricks, Useful libraries.)

After lecture, review your app.

Find libraries that are applicable for your app and can simplify it. Integrate those libraries. Discuss your choices with mentor. Mentor evaluates your work on his own choice.

Evaluation criteria*

2	Apply Airbnb best practices to your project: clean up the code of your application: follow the required naming convention, make sure code has proper alignment and there are no redundant spaces, order all your methods in a right way
3	Add storybook to your project; Add 3 components to storybook with knobs
4	Highlight patterns which were used in the homework by comments in code “// PATTERN: {name}”
5	1. Apply React optimization techniques. 2. Run performance audit (by lighthouse) of your application and increase performance index (target value estimated by mentor)

*** each mark includes previous mark criteria**