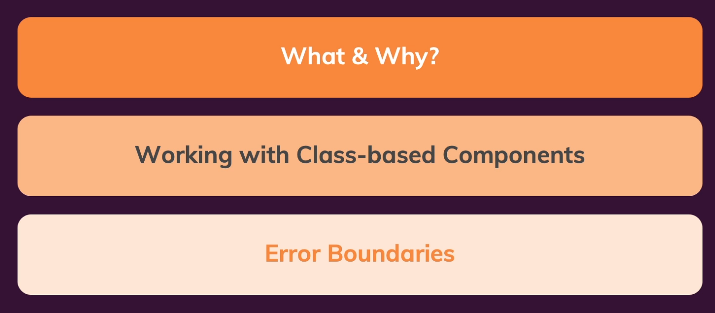
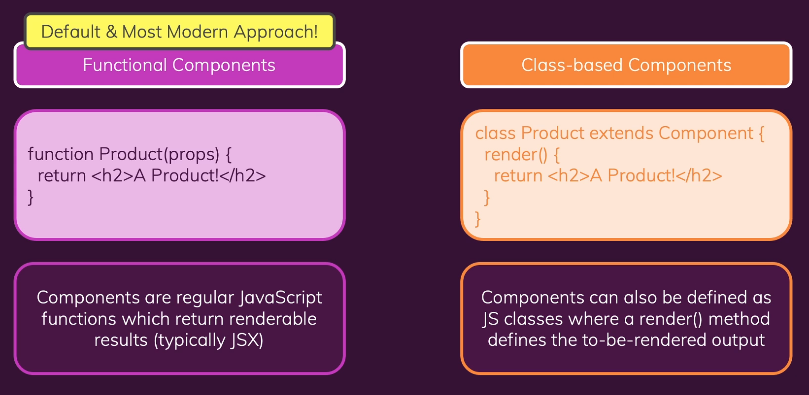
Section 13: An Alternative Way Of Building Components:..





165. Adding a First Class-based Component

Clone source from <https://github.com/academind/react-complete-guide-code/tree/13-class-based-cmp>

Step 1: Change User.js to class like this

import { Component } from 'react';

import classes from './User.module.css';

class User extends Component {

  render() {

    return <li className={classes.user}>{this.props.name}</li>;

  }

}

// const User = (props) => {

//   return <li className={classes.user}>{props.name}</li>;

// };

export default User;

Step 2: Done

166. Working with state & events

Step 1:

import { Component } from 'react';

import User from './User';

import classes from './Users.module.css';

const DUMMY\_USERS = [

  { id: 'u1', name: 'Max' },

  { id: 'u2', name: 'Manuel' },

  { id: 'u3', name: 'Julie' },

];

class Users extends Component {

  constructor() {

    super();

    this.state = {

      showUsers: true,

      more: 'Test',

    };

  }

  toggleUsersHandler() {

    // this.state.showUsers = false; // NOT!

    this.setState((curState) => {

      return { showUsers: !curState.showUsers };

    });

  }

  render() {

    const usersList = (

      <ul>

        {DUMMY\_USERS.map((user) => (

          <User key={user.id} name={user.name} />

        ))}

      </ul>

    );

    return (

      <div className={classes.users}>

        <button onClick={this.toggleUsersHandler.bind(this)}>

          {this.state.showUsers ? 'Hide' : 'Show'} Users

        </button>

        {this.state.showUsers && usersList}

      </div>

    );

  }

}

// const Users = () => {

//   const [showUsers, setShowUsers] = useState(true);

//   const toggleUsersHandler = () => {

//     setShowUsers((curState) => !curState);

//   };

//   const usersList = (

//     <ul>

//       {DUMMY\_USERS.map((user) => (

//         <User key={user.id} name={user.name} />

//       ))}

//     </ul>

//   );

//   return (

//     <div className={classes.users}>

//       <button onClick={toggleUsersHandler}>

//         {showUsers ? 'Hide' : 'Show'} Users

//       </button>

//       {showUsers && usersList}

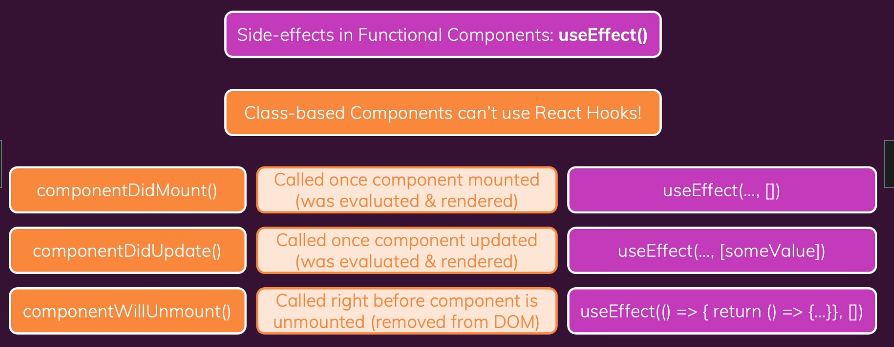
//     </div>

//   );

// };

export default Users;

167. The Component Lifecycle



168. Lifecycle Methods In Action (Change UseEffect when use class)

Step 1 Clone source ‘lifeCycle Methods In Action’

Step 2 conver UserFinder to new code

Old Code

import { Fragment, useState, useEffect } from 'react';

import Users from './Users';

import classes from './UserFinder.module.css';

const DUMMY\_USERS = [

  { id: 'u1', name: 'Max' },

  { id: 'u2', name: 'Manuel' },

  { id: 'u3', name: 'Julie' },

];

const UserFinder = () => {

  const [filteredUsers, setFilteredUsers] = useState(DUMMY\_USERS);

  const [searchTerm, setSearchTerm] = useState('');

  useEffect(() => {

    setFilteredUsers(

      DUMMY\_USERS.filter((user) => user.name.includes(searchTerm))

    );

  }, [searchTerm]);

  const searchChangeHandler = (event) => {

    setSearchTerm(event.target.value);

  };

  return (

    <Fragment>

      <div className={classes.finder}>

        <input type='search' onChange={searchChangeHandler} />

      </div>

      <Users users={filteredUsers} />

    </Fragment>

  );

};

export default UserFinder;

New Code

import { Fragment, useState, useEffect, Component } from 'react';

import Users from './Users';

import classes from './UserFinder.module.css';

const DUMMY\_USERS = [

  { id: 'u1', name: 'Max' },

  { id: 'u2', name: 'Manuel' },

  { id: 'u3', name: 'Julie' },

];

class UserFinder extends Component {

  constructor() {

    super();

    this.state = {

      filteredUsers: [],

      searchTerm: '',

    };

  }

  componentDidMount() {

    // Send http request...

    this.setState({ filteredUsers: DUMMY\_USERS });

  }

  componentDidUpdate(prevProps, prevState) {

    if (prevState.searchTerm !== this.state.searchTerm) {

      this.setState({

        filteredUsers: DUMMY\_USERS.filter((user) =>

          user.name.includes(this.state.searchTerm)

        ),

      });

    }

  }

  searchChangeHandler(event) {

    this.setState({ searchTerm: event.target.value });

  }

  render() {

    return (

      <Fragment>

        <div className={classes.finder}>

          <input type='search' onChange={this.searchChangeHandler.bind(this)} />

        </div>

        <Users users={this.state.filteredUsers} />

      </Fragment>

    );

  }

}

// const UserFinder = () => {

//   const [filteredUsers, setFilteredUsers] = useState(DUMMY\_USERS);

//   const [searchTerm, setSearchTerm] = useState('');

//   useEffect(() => {

//     setFilteredUsers(

//       DUMMY\_USERS.filter((user) => user.name.includes(searchTerm))

//     );

//   }, [searchTerm]);

//   const searchChangeHandler = (event) => {

//     setSearchTerm(event.target.value);

//   };

//   return (

//     <Fragment>

//       <div className={classes.finder}>

//         <input type='search' onChange={searchChangeHandler} />

//       </div>

//       <Users users={filteredUsers} />

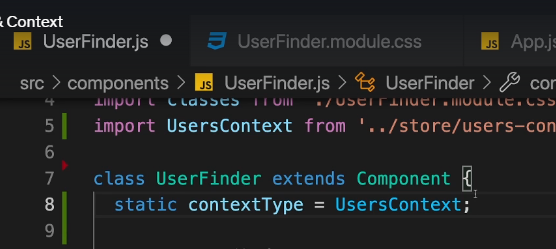
//     </Fragment>

//   );

// };

export default UserFinder;

169. Class-based Components & Context (Change UseContext when use class)



=> Use static when use Context in class (Because context only allow init one time when work with component)

Code in UserFinder.js

import { Fragment, useState, useEffect, Component } from 'react';

import Users from './Users';

import classes from './UserFinder.module.css';

import UsersContext from '../store/users-context';

class UserFinder extends Component {

  static contextType = UsersContext;

  constructor() {

    super();

    this.state = {

      filteredUsers: [],

      searchTerm: '',

    };

  }

  componentDidMount() {

    // Send http request...

    this.setState({ filteredUsers: this.context.users });

  }

  componentDidUpdate(prevProps, prevState) {

    if (prevState.searchTerm !== this.state.searchTerm) {

      this.setState({

        filteredUsers: this.context.users.filter((user) =>

          user.name.includes(this.state.searchTerm)

        ),

      });

    }

  }

  searchChangeHandler(event) {

    this.setState({ searchTerm: event.target.value });

  }

  render() {

    return (

      <Fragment>

        <div className={classes.finder}>

          <input type='search' onChange={this.searchChangeHandler.bind(this)} />

        </div>

        <Users users={this.state.filteredUsers} />

      </Fragment>

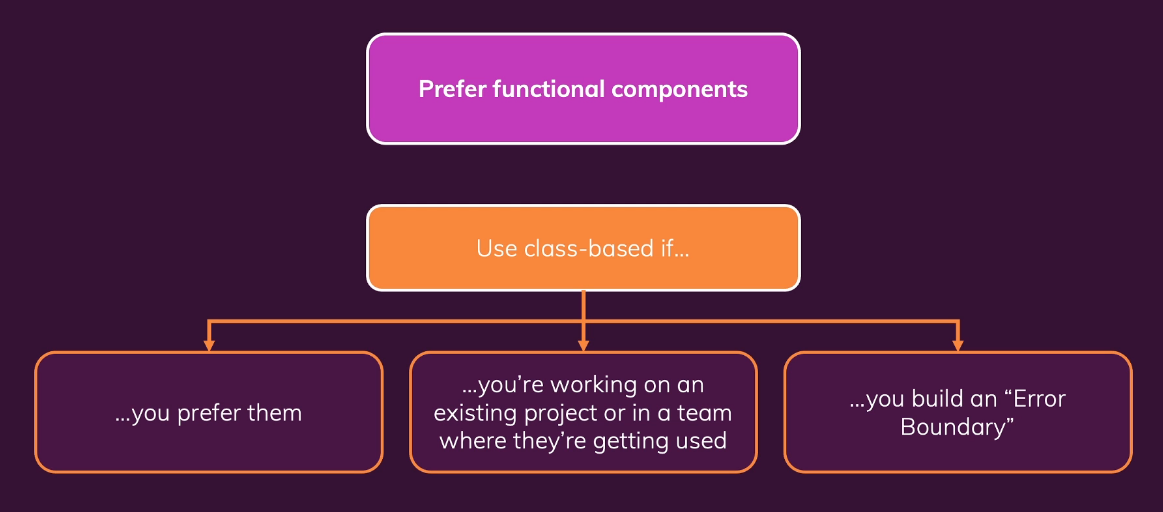
    );

  }

}

export default UserFinder;

170. Class-based vs Functional Components: A Summary



171. Introducing Error Boundaries

Step 1 Create new ErrorBoundary.js in folder components

import { Component } from 'react';

class ErrorBoundary extends Component {

  constructor() {

    super();

    this.state = { hasError: false };

  }

  componentDidCatch(error) {

    console.log(error);

    this.setState({ hasError: true });

  }

  render() {

    if (this.state.hasError) {

      return <p>Something went wrong!</p>;

    }

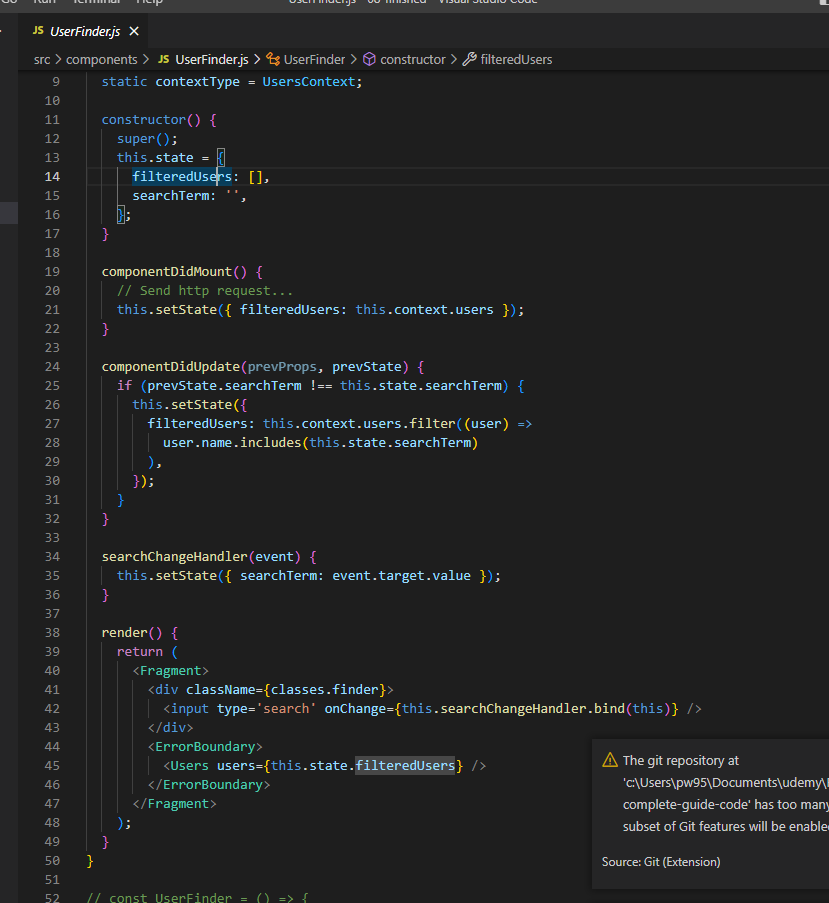
    return this.props.children;

  }

}

export default ErrorBoundary;

Step 2: then wrap to tag may get error like this



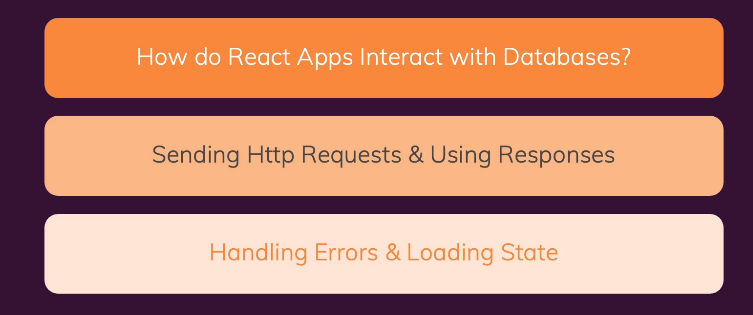
172. Module Resources

You may want to **compare your code to mine** (e.g. to find + fix errors).

For that, you find **multiple code snapshots** for this module here in this Github repository: <https://github.com/academind/react-complete-guide-code/tree/13-class-based-cmp>

Section 14 Sending Http Requests

173. Source to start



<https://github.com/academind/react-complete-guide-code/tree/14-sending-http-requests>

175. using the Star Wars API

**Using the Star Wars API**

**MUST READ:**

In the next lecture, you will be introduced to our demo backend that will be used in this course section: The Star Wars API.

I will use this page: <https://swapi.dev/>

Loading this page (and hence accessing this backend) **might fail** - if that is the case for you, you can **use this alternative**: <https://swapi.py4e.com/>

176. Our Starting App & Backend

Test Access data

<https://swapi.dev/api/films>

177. Sending a GET Request

Step 1 Change app.js like this

import React, { useState } from 'react';

import MoviesList from './components/MoviesList';

import './App.css';

function App() {

  const [movies, setMovies] = useState([]);

  function fetchMoviesHandler() {

    fetch('https://swapi.dev/api/films/')

      .then((response) => {

        return response.json();

      })

      .then((data) => {

        const transformedMovies = data.results.map((movieData) => {

          return {

            id: movieData.episode\_id,

            title: movieData.title,

            openingText: movieData.opening\_crawl,

            releaseDate: movieData.release\_date,

          };

        });

        setMovies(transformedMovies);

      });

  }

  return (

    <React.Fragment>

      <section>

        <button onClick={fetchMoviesHandler}>Fetch Movies</button>

      </section>

      <section>

        <MoviesList movies={movies} />

      </section>

    </React.Fragment>

  );

}

export default App;

Step 2: check result

178. Using async / await

Version 1 dùng callBack

+Trong một thời gian rất dài, chúng ta phải dựa vào callbacks để làm việc

+ Callback có rất nhiều nhược điểm. Khi ta có nhiều thao tác bất đồng bộ, các callback phải chờ nhau thực hiện, thời gian để hoàn thành sẽ bị kéo dài hơn. Ngoài ra, việc viết các callback lồng nhau cũng làm cho mã nguồn của ta rắc rối và khó bảo trì.

Version 2 dùng Promise

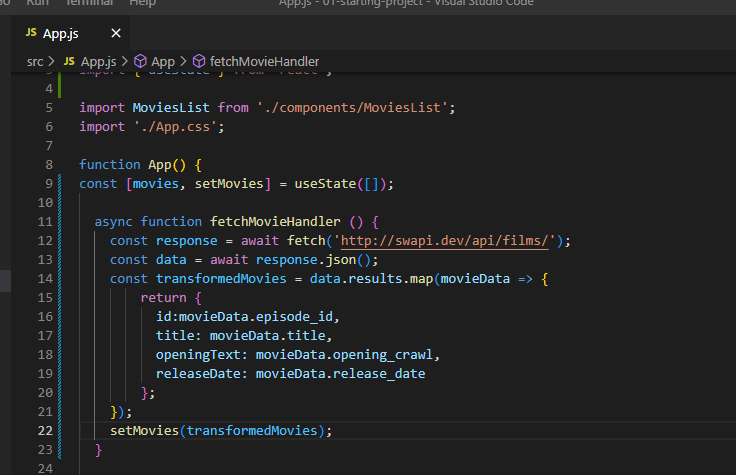
+Code mới của chúng ta gần giống với code cũ, kết quả là trông dễ theo dõi và bảo trì hơn. Tuy nhiên các vấn đề của callback vẫn chưa được giải quyết triệt để.

Version 3 Dùng Async / Await

 Async / Await đã được bổ xung để việc viết code bất đồng bộ trong JavaScript tốt hơn, code dễ nhìn hơn và dễ sử dụng hơn.

* **Async** - khai báo một hàm bất đồng bộ (async function someName(){...}).
  + Tự động biến đổi một hàm thông thường thành một Promise.
  + Khi gọi tới hàm async nó sẽ xử lý mọi thứ và được trả về kết quả trong hàm của nó.
  + Async cho phép sử dụng Await.
* **Await** - tạm dừng việc thực hiện các hàm async. (Var result = await someAsyncCall ()
  + Khi được đặt trước một Promise, nó sẽ đợi cho đến khi Promise kết thúc và trả về kết quả.
  + Await chỉ làm việc với Promises, nó không hoạt động với callbacks.
  + Await chỉ có thể được sử dụng bên trong các function async.

Step 1 set default of react (add async / await in app.js)



179. Handling Loading & Data States

Step 1 Step 1 add loading in App.js

import React from 'react';

import { useState } from 'react';

import MoviesList from './components/MoviesList';

import './App.css';

function App() {

  const [movies, setMovies] = useState([]);

  const [isLoading, setIsLoading] = useState(false);

  async function fetchMovieHandler () {

    setIsLoading(true);

    const response = await fetch('http://swapi.dev/api/films/');

    const data = await response.json();

    const transformedMovies = data.results.map(movieData => {

        return {

          id:movieData.episode\_id,

          title: movieData.title,

          openingText: movieData.opening\_crawl,

          releaseDate: movieData.release\_date

        };

    });

    setMovies(transformedMovies);

    setIsLoading(false);

  }

  return (

    <React.Fragment>

      <section>

        <button onClick={fetchMovieHandler}>Fetch Movies</button>

      </section>

      <section>

        {!isLoading && movies.length > 0 && <MoviesList movies={movies} />}

        {!isLoading && movies.length === 0 && <p>Found no movies</p>}

        {isLoading && <p>Loading...</p>}

      </section>

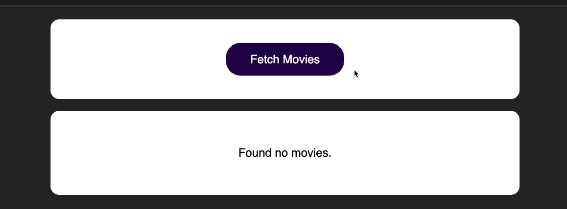
    </React.Fragment>

  );

}

export default App;

Step 2 check result



180. Handling Http Errors

Step 1 in app.js add error when backend return 404, 500

import React from "react";

import { useState } from "react";

import MoviesList from "./components/MoviesList";

import "./App.css";

function App() {

  const [movies, setMovies] = useState([]);

  const [isLoading, setIsLoading] = useState(false);

  const [error, setError] = useState();

  async function fetchMovieHandler() {

    setIsLoading(true);

    setError(null);

    try {

      // remove s of films to test error

      const response = await fetch("http://swapi.dev/api/film/");

      const data = await response.json();

      if (!response.ok) {

        throw new Error("Something went wrong");

      }

      const transformedMovies = data.results.map((movieData) => {

        return {

          id: movieData.episode\_id,

          title: movieData.title,

          openingText: movieData.opening\_crawl,

          releaseDate: movieData.release\_date,

        };

      });

      setMovies(transformedMovies);

    } catch {

      setError(error.message);

    }

    setIsLoading(false);

  }

  let content = <p>Found no movies.</p>;

  if (movies.length > 0) {

    content = <MoviesList movies={movies} />;

  }

  if (error) {

    content = <p>{error}</p>;

  }

  if (isLoading) {

    content = <p>Loading...</p>;

  }

  return (

    <React.Fragment>

      <section>

        <button onClick={fetchMovieHandler}>Fetch Movies</button>

      </section>

      <section>{content}</section>

    </React.Fragment>

  );

}

export default App;

181. Using useEffect() For Request

reload and auto call function (to get data from api)

Step 1 in app.js use UseEffect (to call immedialy function) and callBack to ensure (useEffect only run when need)

import React from "react";

import { useState, useEffect, useCallback } from "react";

import MoviesList from "./components/MoviesList";

import "./App.css";

function App() {

  const [movies, setMovies] = useState([]);

  const [isLoading, setIsLoading] = useState(false);

  const [error, setError] = useState();

  //reload and auto call function (to get data from api)

  useEffect(()=>{

    fetchMovieHandler();

  },[fetchMovieHandler]);

  //use call back ensure useEffect only run when need

  const fetchMovieHandler = useCallback(async () => {

    setIsLoading(true);

    setError(null);

    try {

      // remove s of films to test error

      const response = await fetch("http://swapi.dev/api/film/");

      const data = await response.json();

      if (!response.ok) {

        throw new Error("Something went wrong");

      }

      const transformedMovies = data.results.map((movieData) => {

        return {

          id: movieData.episode\_id,

          title: movieData.title,

          openingText: movieData.opening\_crawl,

          releaseDate: movieData.release\_date,

        };

      });

      setMovies(transformedMovies);

    } catch {

      setError(error.message);

    }

    setIsLoading(false);

  },[]);

  let content = <p>Found no movies.</p>;

  if (movies.length > 0) {

    content = <MoviesList movies={movies} />;

  }

  if (error) {

    content = <p>{error}</p>;

  }

  if (isLoading) {

    content = <p>Loading...</p>;

  }

  return (

    <React.Fragment>

      <section>

        <button onClick={fetchMovieHandler}>Fetch Movies</button>

      </section>

      <section>{content}</section>

    </React.Fragment>

  );

}

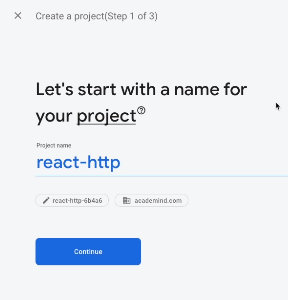
export default App;

Step 2 check your app is call when first time reload page

182. Preparing The Project For The Next Steps

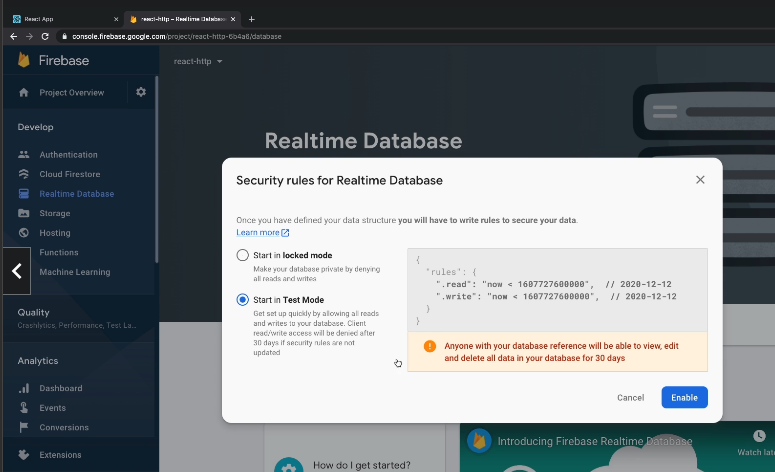
Go to <https://console.firebase.google.com/?pli=1> then create new project

Step 1



Step 2 and click next + finish

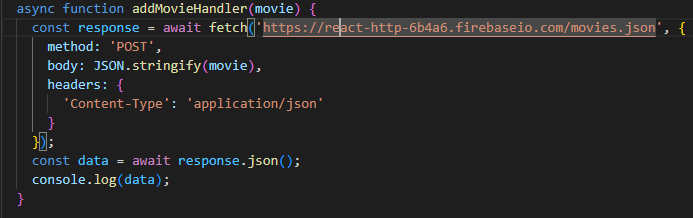
Step 3 click realtime Database then start test mode like this



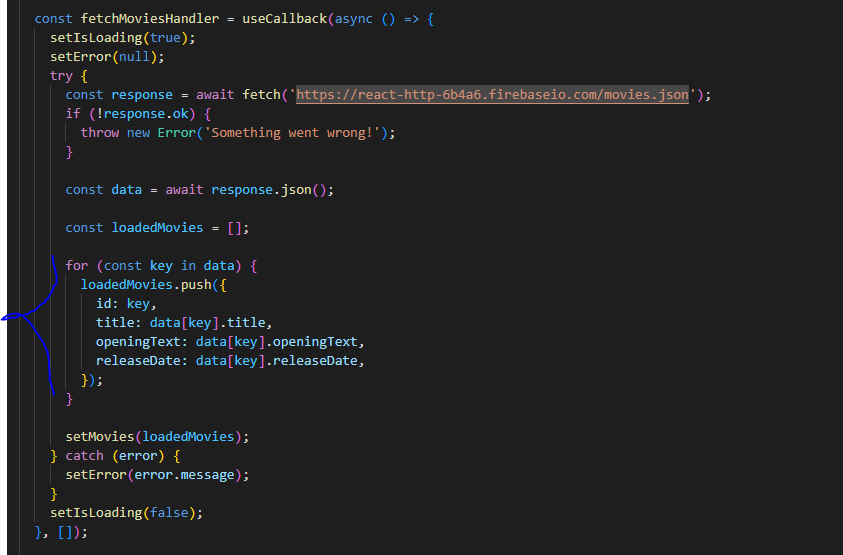
183. Sending a Post Request

Step 1 go to app.js and change like this

Point 1



Point 2



Code

import React, { useState, useEffect, useCallback } from 'react';

import MoviesList from './components/MoviesList';

import AddMovie from './components/AddMovie';

import './App.css';

function App() {

  const [movies, setMovies] = useState([]);

  const [isLoading, setIsLoading] = useState(false);

  const [error, setError] = useState(null);

  const fetchMoviesHandler = useCallback(async () => {

    setIsLoading(true);

    setError(null);

    try {

      const response = await fetch('https://react-http-6b4a6.firebaseio.com/movies.json');

      if (!response.ok) {

        throw new Error('Something went wrong!');

      }

      const data = await response.json();

      const loadedMovies = [];

      for (const key in data) {

        loadedMovies.push({

          id: key,

          title: data[key].title,

          openingText: data[key].openingText,

          releaseDate: data[key].releaseDate,

        });

      }

      setMovies(loadedMovies);

    } catch (error) {

      setError(error.message);

    }

    setIsLoading(false);

  }, []);

  useEffect(() => {

    fetchMoviesHandler();

  }, [fetchMoviesHandler]);

  async function addMovieHandler(movie) {

    const response = await fetch('https://react-http-6b4a6.firebaseio.com/movies.json', {

      method: 'POST',

      body: JSON.stringify(movie),

      headers: {

        'Content-Type': 'application/json'

      }

    });

    const data = await response.json();

    console.log(data);

  }

  let content = <p>Found no movies.</p>;

  if (movies.length > 0) {

    content = <MoviesList movies={movies} />;

  }

  if (error) {

    content = <p>{error}</p>;

  }

  if (isLoading) {

    content = <p>Loading...</p>;

  }

  return (

    <React.Fragment>

      <section>

        <AddMovie onAddMovie={addMovieHandler} />

      </section>

      <section>

        <button onClick={fetchMoviesHandler}>Fetch Movies</button>

      </section>

      <section>{content}</section>

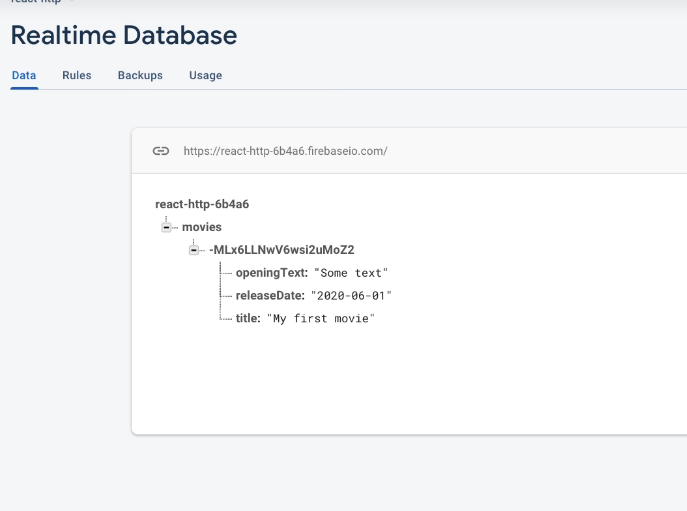
    </React.Fragment>

  );

}

export default App;

Step 2 check in firebase, your will see data you send to firebase



185. Module Resources

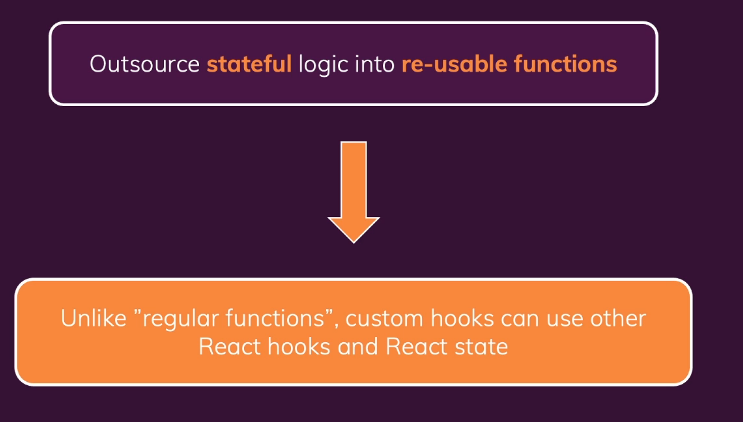
You may want to **compare your code to mine** (e.g. to find + fix errors).

For that, you find **multiple code snapshots** for this module here in this Github repository: <https://github.com/academind/react-complete-guide-code/tree/14-sending-http-requests>

Section 15: Building Custom React Hooks



187. What are ‘Custom Hooks’



188. Creating a Custom React Hook Function + 189. Using Custom Hooks

Source to start

<https://github.com/academind/react-complete-guide-code/tree/15-building-custom-react-hooks>

Step 1: clone source staring project

Step 2: create folder src/hooks and use-counter.js

import { useState, useEffect } from 'react';

const useCounter = () => {

  const [counter, setCounter] = useState(0);

  useEffect(() => {

    const interval = setInterval(() => {

      setCounter((prevCounter) => prevCounter + 1);

    }, 1000);

    return () => clearInterval(interval);

  }, []);

  return counter;

};

export default useCounter;

Step 3: in ForwardCounter.js change to this

import Card from './Card';

import useCounter from '../hooks/use-counter';

const ForwardCounter = () => {

  const counter = useCounter();

  return <Card>{counter}</Card>;

};

export default ForwardCounter;

190.Configuring Custom Hooks

Step 1 in use-counter.js

import { useState, useEffect } from 'react';

//default forwards value

const useCounter = (forwards = true) => {

  const [counter, setCounter] = useState(0);

  useEffect(() => {

    const interval = setInterval(() => {

      if (forwards) {

        setCounter((prevCounter) => prevCounter + 1);

      } else {

        setCounter((prevCounter) => prevCounter - 1);

      }

    }, 1000);

    return () => clearInterval(interval);

  }, [forwards]);

  return counter;

};

export default useCounter;

Step 2 in BackwardCounter.js

import Card from './Card';

import useCounter from '../hooks/use-counter';

const BackwardCounter = () => {

  const counter = useCounter(false);

  return <Card>{counter}</Card>;

};

export default BackwardCounter;

191. Onwards To a More Realistic

Clone source code 04-onwards-to-a-more-realistic

192. Building a Custom Http Hook

Step 1 create new folder hooks and use-http.js

import { useState, useCallback } from 'react';

const useHttp = () => {

  const [isLoading, setIsLoading] = useState(false);

  const [error, setError] = useState(null);

  const sendRequest = useCallback(async (requestConfig, applyData) => {

    setIsLoading(true);

    setError(null);

    try {

      const response = await fetch(requestConfig.url, {

        method: requestConfig.method ? requestConfig.method : 'GET',

        headers: requestConfig.headers ? requestConfig.headers : {},

        body: requestConfig.body ? JSON.stringify(requestConfig.body) : null,

      });

      if (!response.ok) {

        throw new Error('Request failed!');

      }

      const data = await response.json();

      applyData(data);

    } catch (err) {

      setError(err.message || 'Something went wrong!');

    }

    setIsLoading(false);

  }, []);

  return {

    isLoading,

    error,

    sendRequest,

  };

};

export default useHttp;

Step 2 Then in app.js

import React, { useEffect, useState } from 'react';

import Tasks from './components/Tasks/Tasks';

import NewTask from './components/NewTask/NewTask';

import useHttp from './hooks/use-http';

function App() {

  const [tasks, setTasks] = useState([]);

  const { isLoading, error, sendRequest: fetchTasks } = useHttp();

  useEffect(() => {

    const transformTasks = (tasksObj) => {

      const loadedTasks = [];

      for (const taskKey in tasksObj) {

        loadedTasks.push({ id: taskKey, text: tasksObj[taskKey].text });

      }

      setTasks(loadedTasks);

    };

    fetchTasks(

      { url: 'https://react-http-6b4a6.firebaseio.com/tasks.json' },

      transformTasks

    );

  }, [fetchTasks]);

  const taskAddHandler = (task) => {

    setTasks((prevTasks) => prevTasks.concat(task));

  };

  return (

    <React.Fragment>

      <NewTask onAddTask={taskAddHandler} />

      <Tasks

        items={tasks}

        loading={isLoading}

        error={error}

        onFetch={fetchTasks}

      />

    </React.Fragment>

  );

}

export default App;

195. Using The Custom Hook In More Components

Step 1 go to NewTask.js

import Section from '../UI/Section';

import TaskForm from './TaskForm';

import useHttp from '../../hooks/use-http';

const NewTask = (props) => {

  const { isLoading, error, sendRequest: sendTaskRequest } = useHttp();

  const createTask = (taskText, taskData) => {

    const generatedId = taskData.name; // firebase-specific => "name" contains generated id

    const createdTask = { id: generatedId, text: taskText };

    props.onAddTask(createdTask);

  };

  const enterTaskHandler = async (taskText) => {

    sendTaskRequest(

      {

        url: 'https://react-http-6b4a6.firebaseio.com/tasks.json',

        method: 'POST',

        headers: {

          'Content-Type': 'application/json',

        },

        body: { text: taskText },

      },

      createTask.bind(null, taskText)

    );

  };

  return (

    <Section>

      <TaskForm onEnterTask={enterTaskHandler} loading={isLoading} />

      {error && <p>{error}</p>}

    </Section>

  );

};

export default NewTask;

196. Module Resources

You may want to **compare your code to mine** (e.g. to find + fix errors).

For that, you find **multiple code snapshots** for this module here in this Github repository: <https://github.com/academind/react-complete-guide-code/tree/15-building-custom-react-hooks>