FUNDAMENTALS

Friday, June 9, 2023 1:38 PM

https://github.com/pkellner/pluralsight-using-hooks-in-react18
JSX(javascript extension) looks like html but it is translated to js by a tool called babel since jsx is not undertadable by browsersd

Const Banner = ()=> <h1>ddd</h2>

To Markup(using babel)

Const banner = ()=> react.createElement("h1",null,"sss")

To Actual html(using React Dom)

<h1>sss</h1>

WE HAVE TO USE FUNCTION COMPONENT TYPE RATHER THAN CLASS COMPONENT TYPE

Tools required for development:

For creating readytogo dev env:
- Create react App (npx create-react-app projectname)
- Next Js(used in this course)(nox create-next-app projectname

ESLINT:
- Detect problems(click debug button in vs code and run and start debug button and in the launch.json file change the localhost port and add a breakpoint on any line in code)
- Code styling
- Commonly used
- Nextjs - build in react ruleset

Using Vs code extension for it we can see the reacts recommendations in code code directly

CSS MODULES:
- To have a css per component
- This is provided by webpack not by react

Import styles from "./banner.module.css"
styles.logo

<div style={{fontStyle: "italic"}} # not recommended

PROPS:

- Are read-only

<Banner text="ssss"/>

```
Const Banner = (props) =>{
Return (
<div>{props.text}</div>
);
};
Export default Banner;
```

CHILDREN PROP:

<Banner>something </Banner>

Const Banner = ({children})=>{
….
{children} - > it will contain whatever inside the tags are there

FRAGMENT:

Whenever in a code there is a need of the parent div to surround the content we can use a React.Fragment tag

<React.Fragment>…</React.Fragment> or <>…</>

KEY PROP:

In the list or table rows elements we can use key={h.id} to efficiently order the render the items(wherever we iterrate)

<tr key={h.id}>
…
</tr>

Or
<HourseRow key={h.id} property={h} />
Also we can use spread operator to send all parameters with {…h}

In the component parameter we can extract the value of the document instead of referencing the document variable in each line

Const Component = ({house}) =>{
Return (
{hourse.money}
);
};

To

Const Component = ({house:{address,price,money}}) =>{
Return (
{money}
{price}
);
};

We can also use javascript functions to format texts and numbers etc..

HOOKS:

- Only call hooks at the top level
    - In the same order
- Only call hooks in function components
    - Or custom hooks

Before return statement in component function we can add **const [houses, setHouses] = useState(houseArray);**
access , change

const AddElement = ()=>{
setHouses([…houses,{id:ss,money:ss,ss:Ss},]);

```
        };
    return (
  …
<button onClick={AddElement}>
Add
</button>
        );
```

RENDERING:

Rendering is equal to running the component function but updating browser means done by reconciliation by react

In the chrome extension react developer tools we can see the re rendering components

MEMO:

(caching)

```
Const HourseRowMem =  React.memo(HouseRow);

Export { HouseRowMem };
```

In another component we can use this mem export instead of houseRow component

- Use when it is faster
- When we can measure that it is faster(using profiler in react developer tools)
- Pure function component
- Renders often
- With the same prop values
- Use when JSX isnt trivial(of little importance; not worth considering)

EFFECT(SIDE EFFECTS):
=----------
(use in a unpredictable operations)

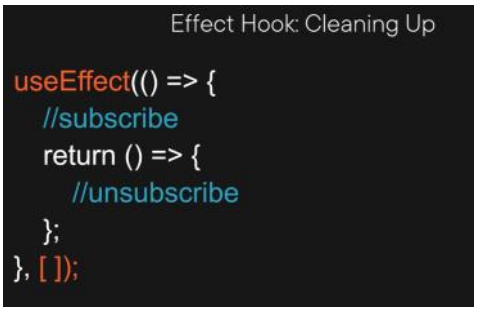
- API interaction
- Use browser API's
- USING TIMING FUNCTIONS

Runs after the pure function and browser has been rendered

```
Const HouseList = ()=>{
Const [hous,sethou] = usetate([]);
useEffect(()=>{
Const fetchhouses = async ()=>{
 const response = await fetch("/api/houses");
 # /pages/api/houses/[id].js directory
Const houses = await response.json();
setHouses(houses);
};
Fetchhouses();
});
…..
}
```

If we use effects like this the function renders and check the hooks and executes the effect and cycle goes on . For this we can set a dependency on which the effect should run

```
useEffect(()=>{
…
},[]); # ifwe give this it runs only once and if given any variable when that changes this will execute
```



MEMO HOOK:

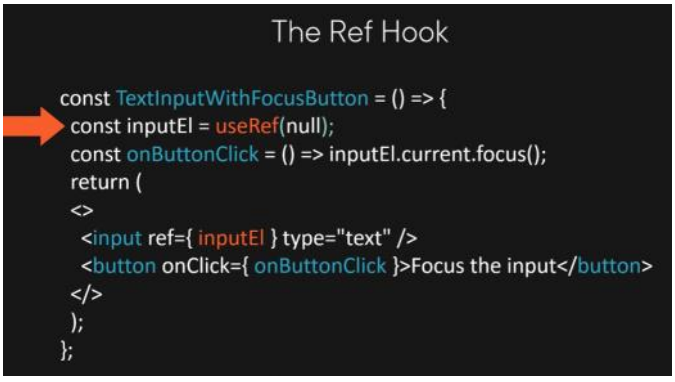
When doing some time consuming calculation function it will affect the performance of the application and rendering will be done frequently

Const result = useMemo(()=>{
Return timeconsumingCalculation(houses);
},[houses]);

Here it only fires when houses changes

REF HOOK:

Used to persist values that survive re-renders without causing a re-render



We can use conditional rendering like this

{house.ishigh && <tr>ssssss</tr>}

We can send the set state functions to other components like this
Const [house,setHouses] = useState();
<House setone = { setHouses } />

In there we can do something like

<tr onClick={()=> setHouses(oldhouse => return [...oldhouse,house])}>
…
</tr>

Also we can have another function doing the set function and we can pass that function to other components to save som ebugs in our code

useCallback is used when we don't need to recreate the variable everytime when multiple components uses it and goes to inifnite loop if not given (eg: if we pass a url into a api get finction then unless url changes this will not be recreated

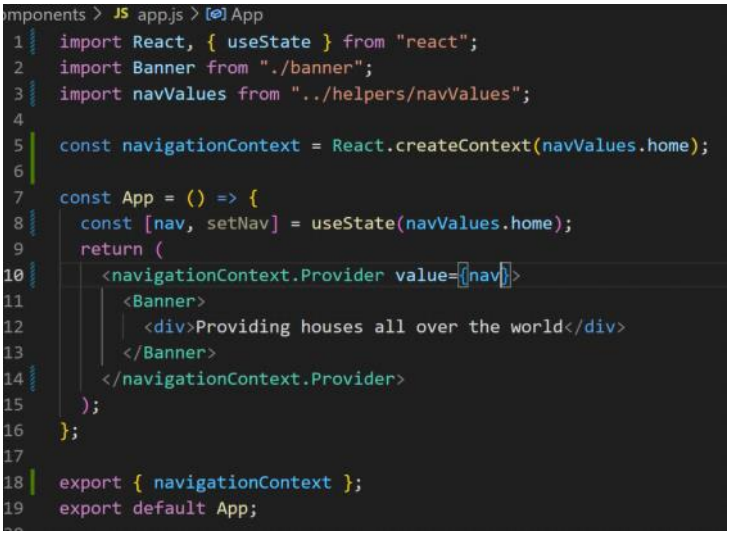
For creating custom hook we need to create a separate js file and wrap all the code from usestate line till useeffect line an d at last return houses from that function

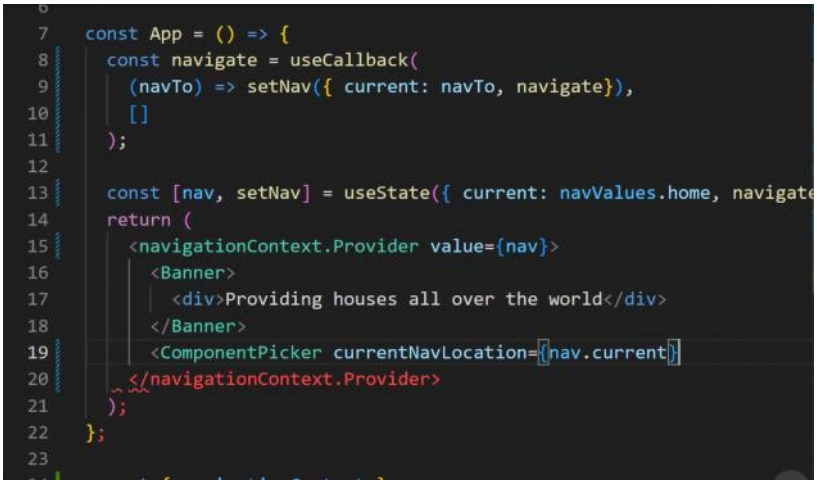
In the custom hook we can return the variable and set function and we can se them in components for code readability and reusability and isolating components state

CONTEXTS:

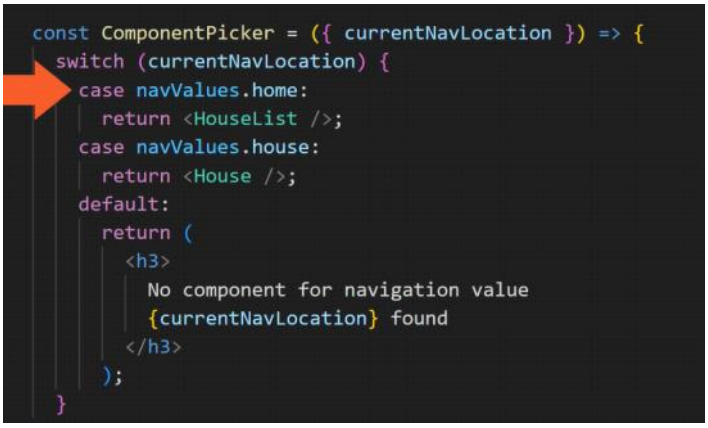
- used to be centralizing the properties and make the component use it without need of props

I DID NOT UNDERSTAND WHEN SEEING THE VIDEO SO FOR UNDERSTANDING I GOT SCREENSHJOT OF JS FILES



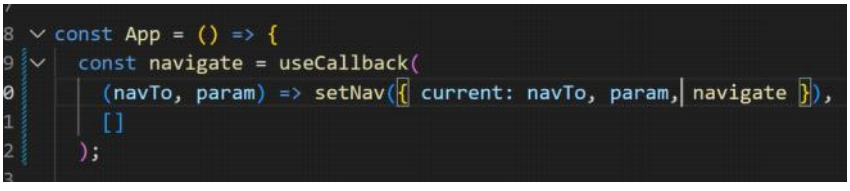


```
const App = () => {
  const navigate = useCallback(
    (navTo) => setNav({ current: navTo, navigate}),
    []
  );

  const [nav, setNav] = useState({ current: navValues.home, navigate
  return (
    <navigationContext.Provider value={nav}>
      <Banner>
        <div>Providing houses all over the world</div>
      </Banner>
      <ComponentPicker currentNavLocation={nav.current}
    </navigationContext.Provider>
  );
};

```

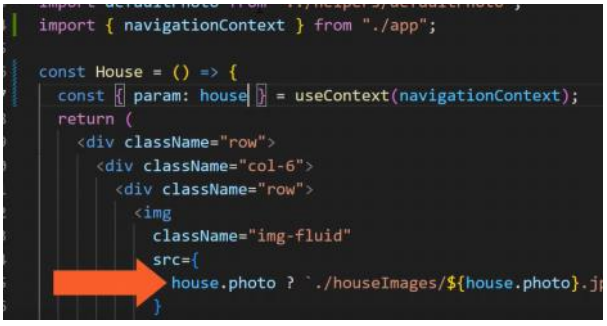


```
const ComponentPicker = ({ currentNavLocation }) => {
  switch (currentNavLocation) {
    case navValues.home:
      return <HouseList />;
    case navValues.house:
      return <House />;
    default:
      return (
        <h3>
          No component for navigation value
          {currentNavLocation} found
        </h3>
      );
  }
```

```
omponents > houseRow.js > HouseRow
import { useContext } from "react";
import currencyFormatter from "../helpers/currencyFormatter";
import navValues from "../helpers/navValues";
import { navigationContext } from "./app";

const HouseRow = ({ house }) => {
  const { navigate } = useContext(navigationContext);
  return (
    <tr onClick={() => navigate(navValues.house, house)}>
      <td>{house.address}</td>
      <td>{house.country}</td>
      {house.price && (
        <td className={`${house.price >= 500000 ? "text-primary
          {currencyFormatter.format(house.price)}
        </td>
      )}
    </tr>
  );
```



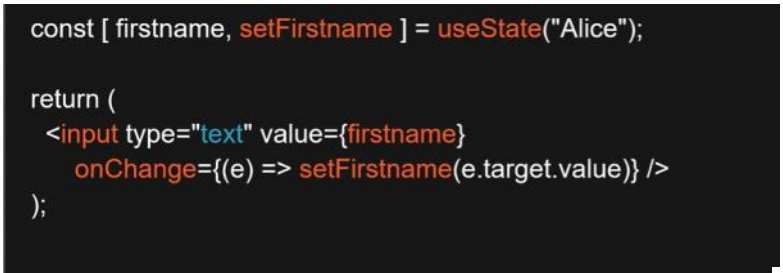
```
const App = () => {
  const navigate = useCallback(
    (navTo, param) => setNav({ current: navTo, param, navigate }),
    []
  );
```



```
import { navigationContext } from "./app";

const House = () => {
  const { param: house } = useContext(navigationContext);
  return (
    <div className="row">
      <div className="col-6">
        <div className="row">
          <img
            className="img-fluid"
            src={
              house.photo ? `./houseImages/${house.photo}.jp
            }
```

```jsx
const Banner = ({ children }) => {
  const { navigate } = useContext(navigationContext);
  return (
    <header className="row mb-4">
      <div className="col-5">
        <img
          src="./GloboLogo.png"
          alt="logo"
          className={logo}
          onClick={() => navigate(navValues.home)}
        />
      </div>
      <div className="col-7 mt-5" style={subtitleStyle}>
        {children}
      </div>
    </header>
  );
};
```

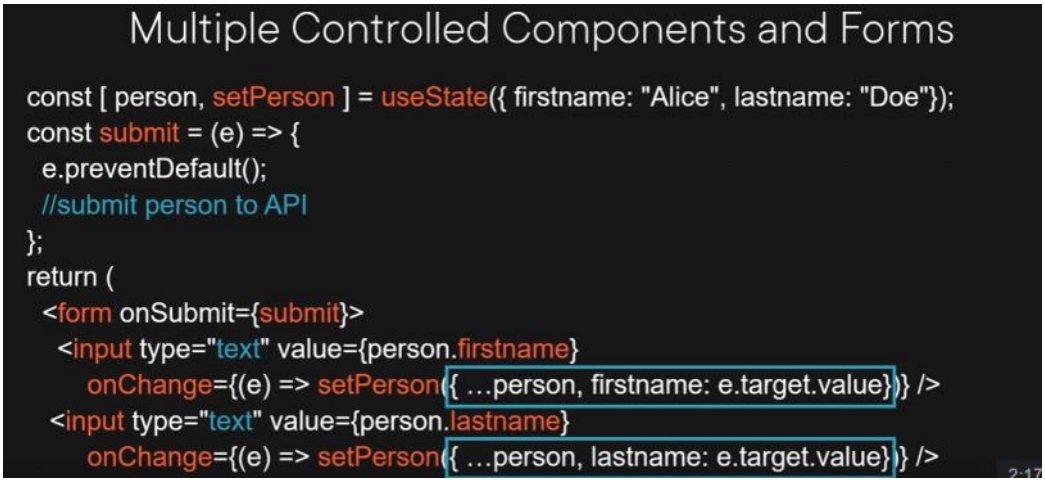
TO ROUTE BASED ON URL'S WE HAVE TO USE React Routerr for creatre-react-app and Nextjs Router if create-next-app



```jsx
const [ firstname, setFirstname ] = useState("Alice");

return (
  <input type="text" value={firstname}
     onChange={(e) => setFirstname(e.target.value)} />
);
```

Controlled Components and Forms

```jsx
const [ firstname, setFirstname ] = useState("Alice");
const submit = (e) => {
  e.preventDefault();
  //submit firstname to API
};

return (
  <form onSubmit={submit}>
    <input type="text" value={firstname}
      onChange={(e) => setFirstname(e.target.value)} />
  </form>
);
```



Multiple Controlled Components and Forms

```jsx
const [ person, setPerson ] = useState({ firstname: "Alice", lastname: "Doe"});
const submit = (e) => {
  e.preventDefault();
  //submit person to API
};
return (
  <form onSubmit={submit}>
    <input type="text" value={person.firstname}
      onChange={(e) => setPerson({ ...person, firstname: e.target.value})} />
    <input type="text" value={person.lastname}
      onChange={(e) => setPerson({ ...person, lastname: e.target.value})} />
```

A Common onChange Handler

```
const [ person, setPerson ] = useState({ firstname: "Alice", lastname: "Doe"});
const change = ((e) => setPerson({ …person, [e.target.name]: e.target.value });

return (
  <form onSubmit={submit}>
    <input type="text" name="firstname" value={person.firstname}
      onChange={change}/>
   <input type="text" name="lastname" value={person.lastname}
       onChange={change} />
  </form>
);)
```

select

```
HTML:
<select>
    <option value="option1">1</option>
    <option selected value="option2">2</option>
</select>

React component:
<select value={state} onChange={change} >
    <option value="option1">1</option>
    <option value="option2">2</option>
```

An UnControlled Component

```
const Form = () => {
 const inputEl = useRef(null);
 const submit = (e) => {
  e.preventDefault();
  const inputValue = inputEl.current.value;
  //process inputValue
  };
 return (
  <form onSubmit={submit}>
   <input ref={ inputEl } type="text" />
   <input type="submit" value="Submit" />
  </form>
 );
```

For simplicity we can use external libraries like formik.org to maximize productibity on forms

File Structure

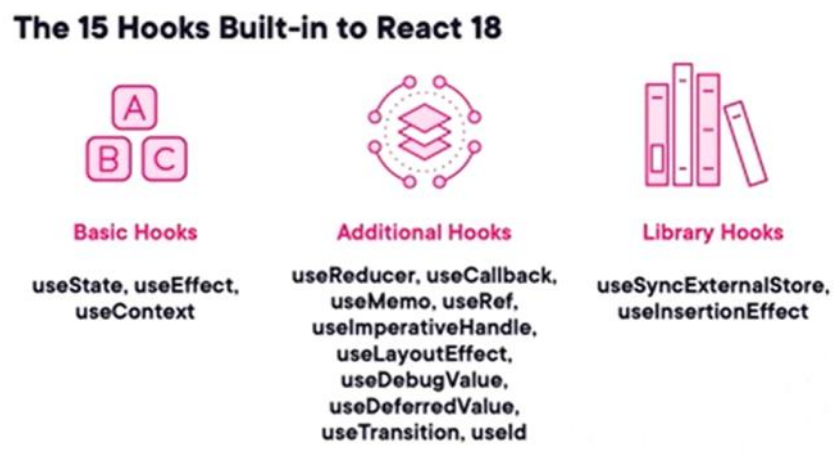
```
app.js
houses
  index.js      → import HouseList from "./houses"
  house.js
  useHouses.js
  useBids.js
  ...
customers
  index.js      → import CustomerList from "./customers"
  useCustomers.js
  ...
common
  currencyFormatter.js
  loadingStatus.js
  navValues.js
  useGetRequest.js
  ...
```

SUGGESTION:

- At the start don’t use state and effect and stuffs
- Start witj props and static pages and if that works then go with states and advanced terms
- Limit the states to limited components that use it

HOOKS

Saturday, June 10, 2023 11:38 AM



SAPLING is a vs code extension that gives a component tree view

useReducer:

---

const [state,dispatch] = useReducer(reducer,initialArg,init);

Reducer -> function takes 1 paramters(state,action)
Init -> function that returns the starting state

Dispatch -> the function that gets called whose parameters are passed into the reducer as the second parameter

/pages/demo-no-reducer.js

```
import { useState } from "react";

export default function demo() {
  const [cnt, setCnt] =
    useState(10);

  return (
    <button onClick={
          () => setCnt(cnt + 1)}>
      {cnt}
    </button>
  )
}
```

/pages/demo-with-reducer.js

```
import { useReducer } from "react";

function reducer(state, action) {
  switch (action.type) {
    case "increment":
      return state +
             action.incrementValue;
    default:
      return action;
  }
}

export default function demo() {
  const [state, dispatch] =
    useReducer(reducer, 10);
    return <button
      onClick={() =>
        dispatch({
          type: "increment",
          incrementValue: 1,
        })}>{state}</button>
}
```

The action is the object passed to dispatch function

Convert useState to useReducer

/pages/demo-reducer.js

```
import { useState, useReducer } from "react";
export default function demo() {
  const [cnt, setCnt] =
    useReducer((_, action) => action, 10);

  return <button onClick={
    () => setCnt(cnt + 1)}>{cnt}</button>
}
```

```
const [state, dispatch] =
  useReducer(reducer, initialArg,
```

Fullscreen (f)

useDeferredValue , useTransition:

---

useDeferredValue and useTransition Differences

useDeferredValue

For use when your state is managed outside of your apps direct control

useTransition

For use when you have direct access to React component state

```
export default function App() {

  const [search, setSearch] = useState("");

  return (
    <>
      <input value={search} onChange={ e => setSearch(e.target.value) } />
      <SlowResults query={search} />
    </>
  );
}
```

Typing into the input field is going to feel slow

```
export default function App() {

  const [search, setSearch] = useState("");
  const deferredSearch = useDeferredValue(search);

  return (
    <>
      <input value={search} onChange={ e => setSearch(e.target.value) } />
      <SlowResults query={deferredSearch} />
    </>
  );
}
```

setSearch has a high priority for updating than deferredSearch

deferredSearch will not update until all the setSearch's are complete

deferredSearch updates and that causes SlowResults to rerender

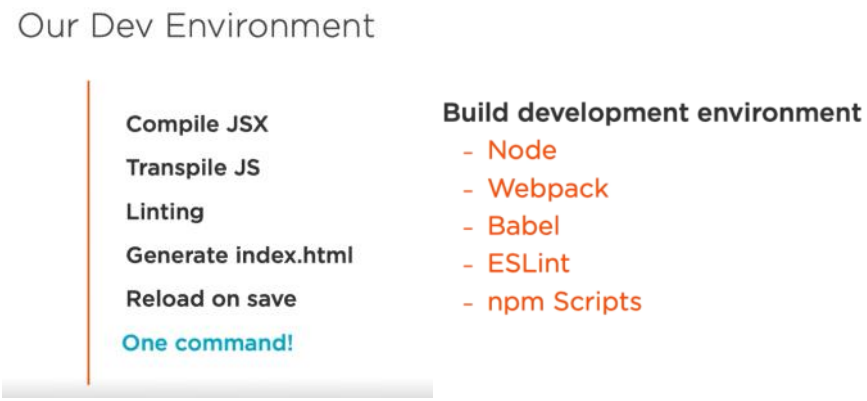
```
export default function App() {

  const [search, setSearch] = useState("");
  const [isPending, startTransition] = useTransition();
  const [currentSearch, setCurrentSearch] = useState("");

  return (
    <>
      <input value={currentSearch}
             onChange={e => {
               setCurrentSearch(e.target.value);
               startTransition(() => setSearch(e.target.value));
             }}
      />
        {isPending ? "refreshing..." : null}
        <SlowResults query={search} />
    </>
  );
}
```

Here when useTransition is set to low priorty it just displays refreshing and after update is complete the high priority value shows up in the box and low priority value load s up

REDUX

Saturday, June 10, 2023 11:40 AM

Use redux-starter-kit to reduce boilerplate



Prettier (vs code extension)is a code formatter we can use to format our code
Preferences -> settings ->search format -> format on save checkbox

FOLDER STRUICTURE STARTING:

Src/favicon.ico
.gitignore to ignorenodemodules,.github
Package.json to have dependent packages

Src/index.html,index.js (here index.html has html that has a container with an id called app , index.js would import React libraries like eact,render and renders the components into that id)

WEBPACK:

We have to create a webpack.config.dev.js in root the ofproject

webpack.config.dev.js

```
const webpack = require('webpack');
const path = require('path');
const HtmlWebpackPlugin = require('html-webpack-plugin');

process.env.NODE_ENV = 'development';

module.exports = {
    mode: 'development',
    target: 'web',
    devTool: 'cheap-module-source-map',
    entry: './src/index',
    output: {
        path: path.resolve(__dirname, "build"),
        publicPath: '/',
        filename: 'bundle.js'
    },
    devServer: {
        stats: 'minimal',
        overlay: true,
        historyApiFallback: true,
        disableHostCheck: true,
        headers: { "Access-Control-Allow-Origin": "*"},
        https: false
    },
    plugins: [
        new HtmlWebpackPlugin({
            template: "src/index.html",
            favicon: "src/favicon.ico"
        })
    ],
    module: {
      rules: [
        {
          test: /\.(js|jsx)$/,
          exclude: /node_modules/,
          use: ["babel-loader"]
        },
        {
          test: /(\.css)$/,
          use: ["style-loader", "css-loader"]
        }
      ]
    }
};
```

BABEL:

Transpiles into JS

In package.json include babel relaed packages to dependencies

Add a section :

"babel":{
"presets":[
"babel-preset-react-app"
]
}

Add npm scripts

```
"scripts": {
  "start": "webpack serve --config webpack.config.dev.js --port 3000"
},
"dependencies": {
  "bootstrap": "5.0.2",
```

ESLINT:

To alert us when making mistakes when coding

We need to add a eslint section in package.json like this

```
"eslintConfig": {
  "extends": [
    "eslint:recommended",
    "plugin:react/recommended",
    "plugin:import/errors",
    "plugin:import/warnings"
  ],
  "parser": "babel-eslint",
  "parserOptions": {
    "ecmaVersion": 2018,
    "sourceType": "module",
    "ecmaFeatures": {
      "jsx": true
    }
  },
  "env": {
    "browser": true,
    "node": true,
    "es6": true,
    "jest": true
  },
  "rules": {
    "no-debugger": "off",
    "no-console": "off",
    "no-unused-vars": "warn",
    "react/prop-types": "warn"
  },
  "settings": {
    "react": {
      "version": "detect"
```

createClass Component

```
var HelloWorld = React.createClass({
  render: function () {
    return (
      <h1>Hello World</h1>
    );
  }
});
```

JS Class Component

```
class HelloWorld extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return (
      <h1>Hello World</h1>
    );
  }
}
```

Function Component

```
function HelloWorld(props) {
  return (
    <h1>Hello World</h1>
  );
}
```

Arrow Function

```
const HelloWorld = (props) => <h1>Hello World</h1>
```

Container	Presentation
Little to no markup	Nearly all markup
Pass data and actions down	Receive data and actions via props
Knows about Redux	Doesn't know about Redux
Often stateful	Often no state

REACT ROUTER:

In the components:

<Link to="about" className="classes">
Component/text to click to navigate
</Link>

In the index.js:

Import { BrowserRouter } from "react-router-dom";

Render(
<BrowserRouter>
<App/>
</BrowserRouter>,
Document.getelementbyid("app")
);

In App.js:

Import { Route,Switch } from "react-router-dom";

…
Return(

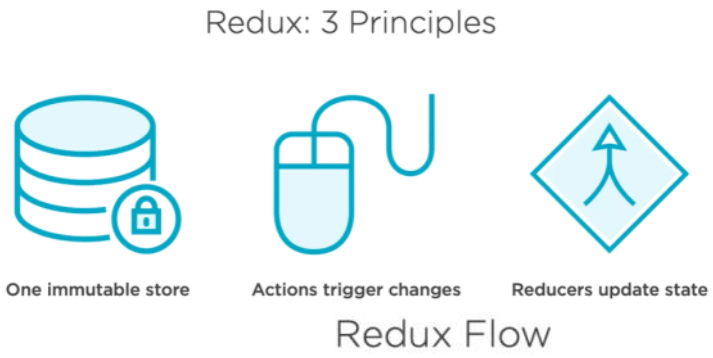
<Header/> # always render
<Switch>
<Route exact path="/" component={HomePageComponent}/>
<Route path="/about" component={AboutPageComponent}/>
<Route component={PageNotFound}/>
</Switch>

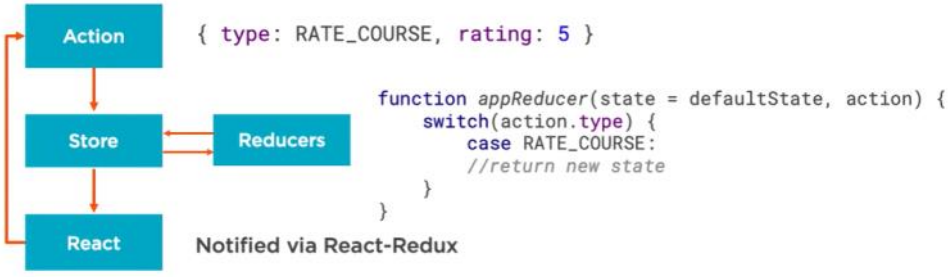
);
…

In Header/NavBar.js:

Import {NavLink} from "react-router-dom";
…
Const activeStyle = {color:"##22332"};
Return(
<nav>
<NavLink to="/" activeStyle={activeStyle} exact>Home</NavLink>
<NavLink to="/about" activeStyle={activeStyle} >About</NavLink>
</nav>
);

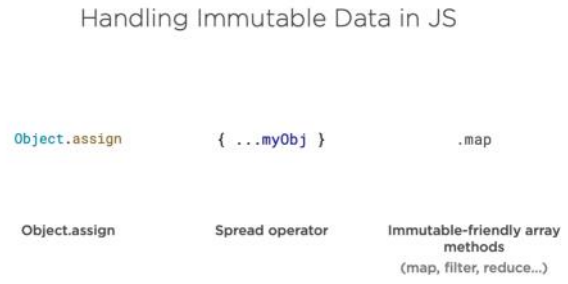
REDUX:





In react redux store we can do following:

- Store.dispatch(action)
- Store.subscribe(listener)
- Store.geState()
- replaceREducer(nextReducer)



Handle Data Changes via Immer

```
import produce from "immer"
const user = {
  name: "Cory",
  address: {
    state: "California"
  }
};

const userCopy = produce(user, draftState => {
  draftState.address.state = "New York"
})
```

Handling Arrays

Avoid	Prefer
push	map
pop	filter
reverse	reduce
	concat
	spread

Must clone array first

The immutability is because if we change state directly it will impact performanc esinc eit should search properties that needs to change but here since we update the whole state it is performanc eoptimized

Reducer function are like useReducer functions

Forbidden in Reducers

- Mutate arguments
- Perform side effects
- Call non-pure functions

Actions
- Represent user intent
- Must have a type

Store
- dispatch, subscribe, getState...

Immutability
- Just return a new copy

Reducers
- Must be pure
- Multiple per app
- Slice of state

Provider -> attaches app to store
Connect -> creates container components

React-Redux Provider

```
<Provider store={this.props.store}>
  <App />
</Provider>
```

Wrapping your App in provider makes the Redux store accessible to every component in your app

Connect

Wraps our component so it's connected to the Redux store.

```
export default connect(mapStateToProps, mapDispatchToProps)(AuthorPage);

function mapStateToProps(state, ownProps) {
  return { authors: state.authors };
}
```

mapStateToProps -> what state should I expose as props

So whatever we return inside this function can be accessed inside component with this.props.documentKey to access redux store data

Reselect

- **Memoize for performance**

```
const getAllCoursesSelector = state => state.courses;

export const getCoursesSorted = createSelector(
  getAllCoursesSelector,
  courses => {
    return [...courses].sort((a, b) =>
      a.title.localeCompare(b.title, "en", { sensitivity: "base" })
    );
});
```

mapDispatchToProps -> what actions do I want on props

```
function mapDispatchToProps(dispatch) {
  return {
    actions: bindActionCreators(actions, dispatch)
  };
}
```

Option 1:
If we don't specify mapDIspatchtoProps we can use this.props.dispatch(loadCourses()) inside the component difrectly

Option 2: Wrap Manually

```
function mapDispatchToProps(dispatch) {
  return {
    loadCourses: () => {
      dispatch(loadCourses());
    },
    createCourse: (course) => {
      dispatch(createCourse(course));
    },
    updateCourse: (course) => {
      dispatch(updateCourse(course));
    }
  };
}

// In component...
this.props.loadCourses()
```

Option 3: bindActionCreators

```
function mapDispatchToProps(dispatch) {
  return {
    actions: bindActionCreators(actions, dispatch)
  };
}
// In component:
this.props.actions.loadCourses();
```

Option 4: mapDispatchToProps as Object

```
const mapDispatchToProps = {
  loadCourses
};
```

Wrapped in dispatch automatically

```
// In component:
this.props.loadCourses();
```

4 Ways to Handle mapDispatchToProps

```
this.props.dispatch(loadCourses());
```

Ignore it

```
function mapDispatchToProps(dispatch) {
  return {
    loadCourses: () => dispatch(loadCourses();
  };
}
```

Manually wrap in dispatch

```
function mapDispatchToProps(dispatch) {
  return {
    actions: bindActionCreators(actions, dispatch)
  };
}
```

Use bindActionCreators

```
const mapDispatchToProps = {
  incrementCounter
};
```

Return object

Initial Redux Setup

1. Create action
2. Create reducer
3. Create root reducer
4. Configure store
5. Instantiate store
6. Connect component
7. Pass props via connect
8. Dispatch action

Add feature

1. Create action
2. Enhance reducer
3. Connect component
4. Dispatch action

Src/redux/actions/
courseActions.js

Export function createCourse(course){
Return {type:"ss",course:course};
}

/src/redux/reducers
courseReducer.js

Export default function courseReducer(state=[],action){
Switch(action.type){
Case "ss":
Return [...state,{...action.course}];
Default:
Return state;
}
}

Our Store

```
const courses = [
  { id: 1, title: "Course 1" },
  { id: 2, title: "Course 2" }
]
```

By ID

```
const courses = {
  1: { id: 1, title: "Course 1" },
  2: { id: 2, title: "Course 2" }
}
```

courses[2]

/src/redux/reducers
Index.js

Import {combineReducers } from "redux";
Import courses from './courseReducer";

Const rootReducer = combineReducers({
Courses: courses
});

Export default rootReducer;

/src/redux
configureStore.js

Import { createStore,applyMiddleware,compose } from "redux";
Import rootReducer from "./reducers';(since it is index.js it will take that file)
Import reduxImmutableStateInvariant from "redux-immutable-state-invariant";

Export default function configureStore(initialState){
Const composeEnhancers = window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__ || compose; // add support for redux dev tools
Return createStore(rootReducer,initalState,composeEnhancers(applyMiddleware(reduxImmutableStateInvariant())));
}

In the index.js file in root:

And on the components to connec to redux:

Import {connect} from "react-redux";
…
Export default connect(mapStateToProps,mapDispatchToProps)(ComponentName);

```
function mapStateToProps(state) {
  return {
    courses: state.courses
  };
}

export default connect(mapStateToProps)(CoursesPage);
```

```
handleSubmit = event => {
  event.preventDefault();
  this.props.dispatch(courseActions.createCourse(this.state.course))
};
```

Here we are omitting msapdispatchtoprops and using dispatch function directly inside the component and courseActions is the one we created at start

```
        />

        <input type="submit" value="Save" />
        {this.props.courses.map(course => (
          <div key={course.title}>{course.title}</div>
        ))}
      </form>
    );
  }
}

CoursesPage.propTypes = {
  courses: PropTypes.array.isRequired,
  dispatch: PropTypes.func.isRequired
};

function mapStateToProps(state) {
```

We can set debugger keyword in code to see breakpoints hit in browser

```
function mapDispatchToProps(dispatch) {
  return {
    createCourse: course => dispatch(courseActions.createCourse(course))
  };
}
```

Since we have wrapped in this itself inside the component we can replace this.props.dispatch(courseActions.createCourse(this.state.course)) to this.props.createCourse(this.state.course)

```
CoursesPage.propTypes = {
  courses: PropTypes.array.isRequired,
  createCourse: PropTypes.func.isRequired
};
```

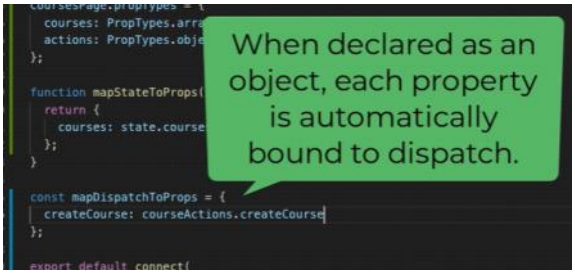
This we use to ignore the errors when implementing these dispatch ones
Import PropTypes from "prop-types";

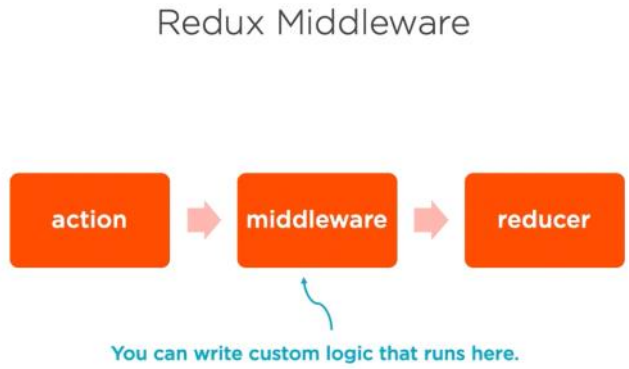
bindActionCreator:

Import{bindActionCreators} from "redux";

```
function mapDispatchToProps(dispatch) {
  return {
    createCourse: bindActionCreators(courseActions, dispatch)
  };
}
```

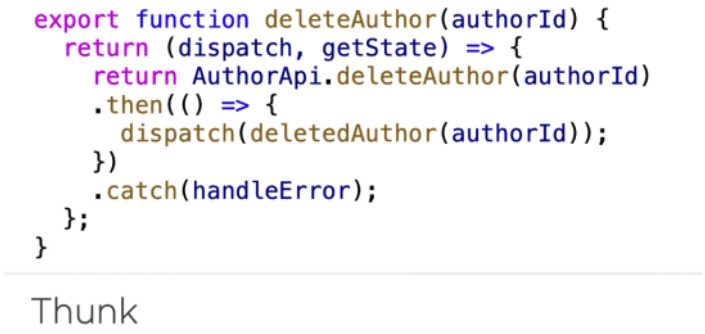
Automatically call dispatch:





Middleware can used to :

- Handling async api calls
- Logging
- Crash reporting
- Routing



In the applyMiddleware in the store configuration js pass thunk as an argument

Import thunk from "redux-thunk";

courseActions.js

```
import * as types from "./actionTypes";
import * as courseApi from "../../api/courseApi";

export function createCourse(course) {
  return { type: types.CREATE_COURSE, course };
}

export function loadCourseSuccess(courses) {
  return { type: types.LOAD_COURSES_SUCCESS, courses };
}

export function loadCourses() {
  return function(dispatch) {
    return courseApi
      .getCourses()
      .then(courses => {
        dispatch(loadCourseSuccess(courses));
      })
      .catch(error => {
        throw error;
      });
  };
}
```

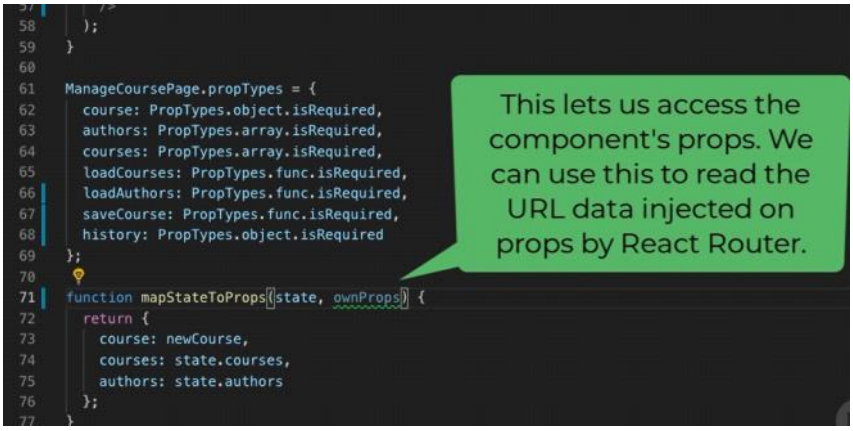
courseReducer.js

```
import * as types from "../actions/actionTypes";

export default function courseReducer(state = [], action) {
  switch (action.type) {
    case types.CREATE_COURSE:
      return [...state, { ...action.course }];
    case types.LOAD_COURSES_SUCCESS:
      return action.courses;
    default:
      return state;
  }
}
```

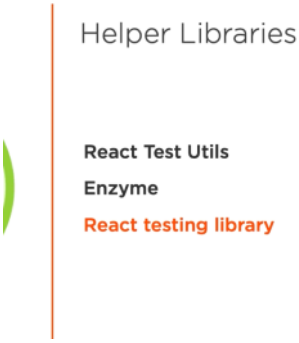
In <Route path="/course/:slug" component={Component}/>
We can use slug as a prop to render according to that

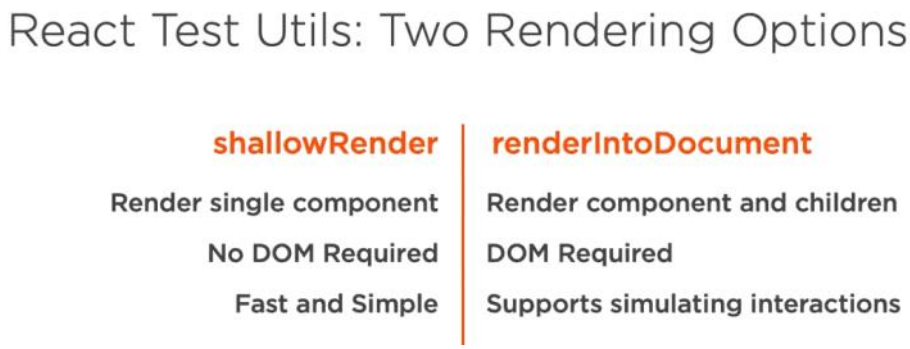
We can use >Redirect to="/course"/> to redirect based on values
Or in any function if it returns promise then use .then(()=>{
History.push("/courses");
});

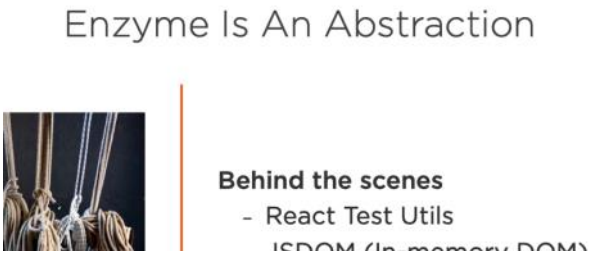
Hsgtory is passed as a prop by react router



We can use reacts UI components like Toast instead of creating a separate component by ourself



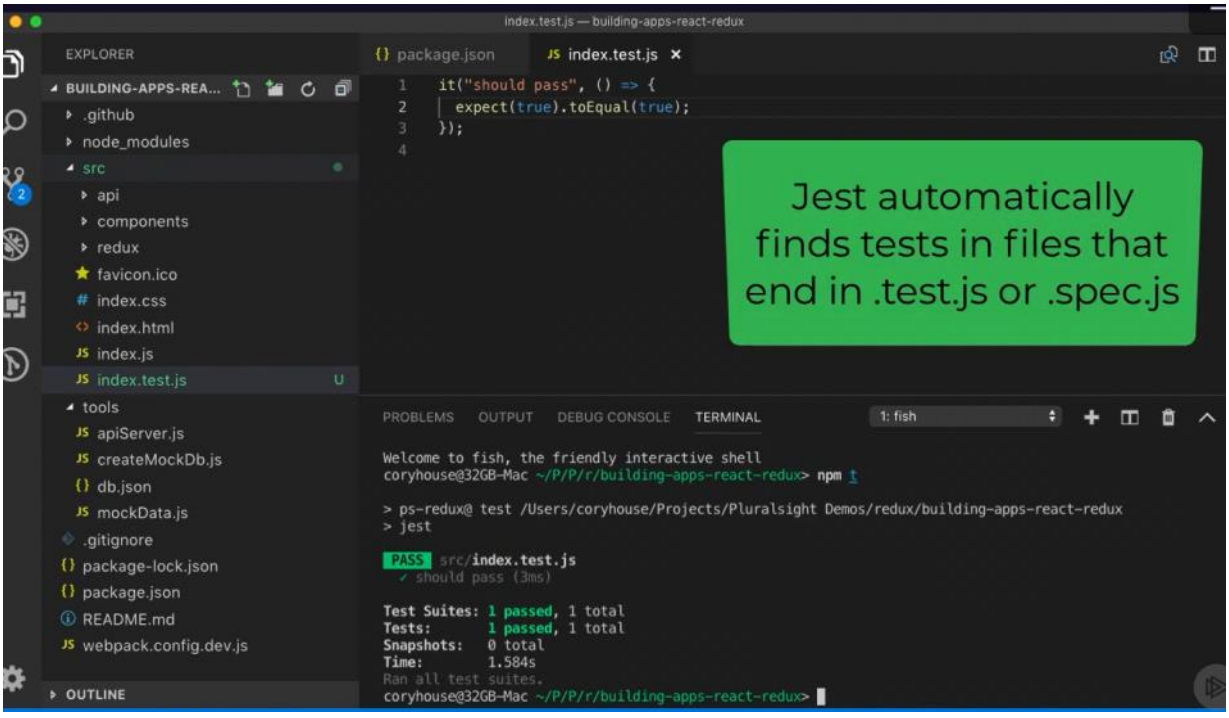




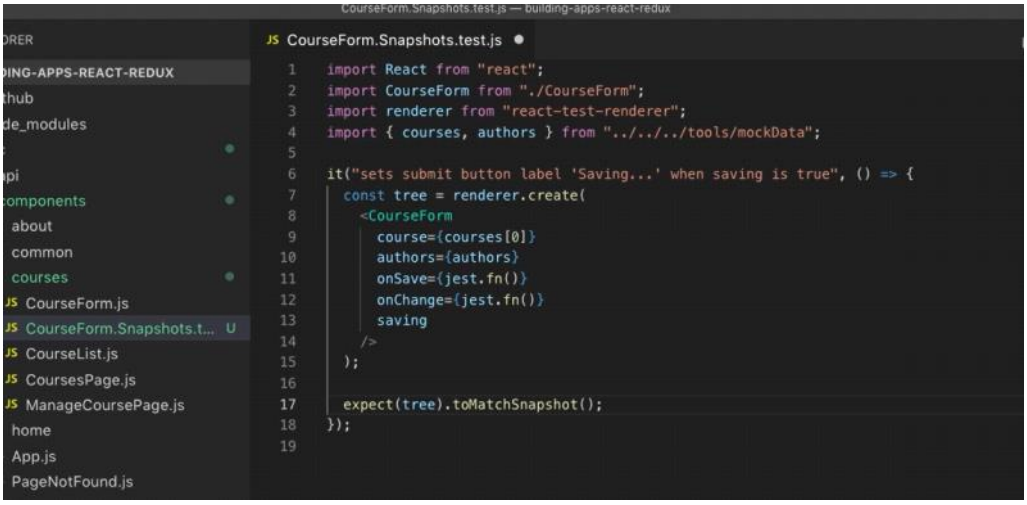
Enzyme Is An Abstraction

Behind the scenes

- React Test Utils
- JSDOM (In-memory DOM)
- Cheerio (Fast jQuery style selectors)

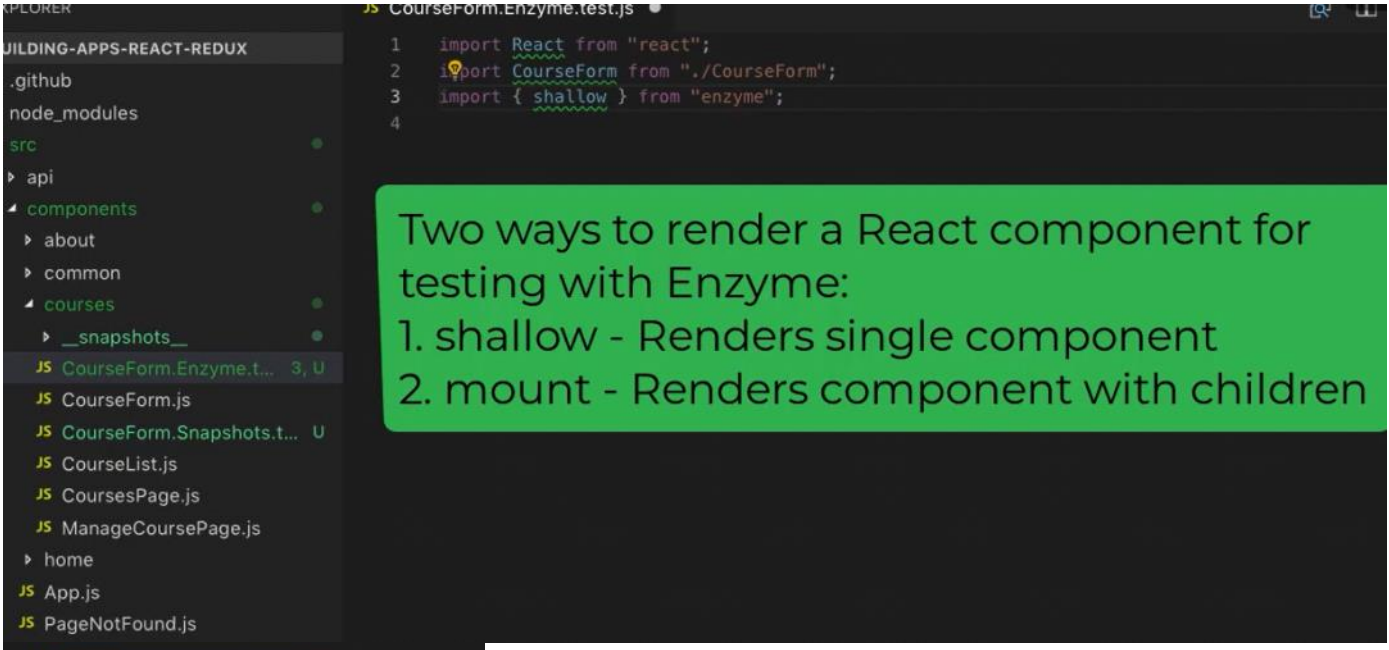


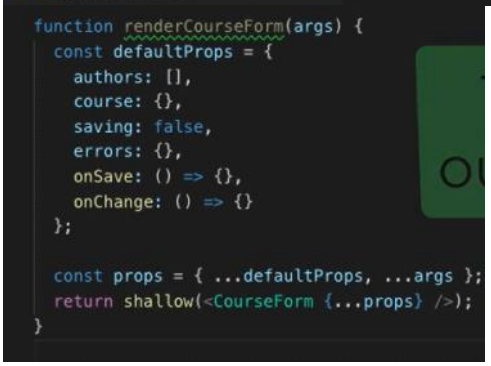
For using jest we need to import it in package.json

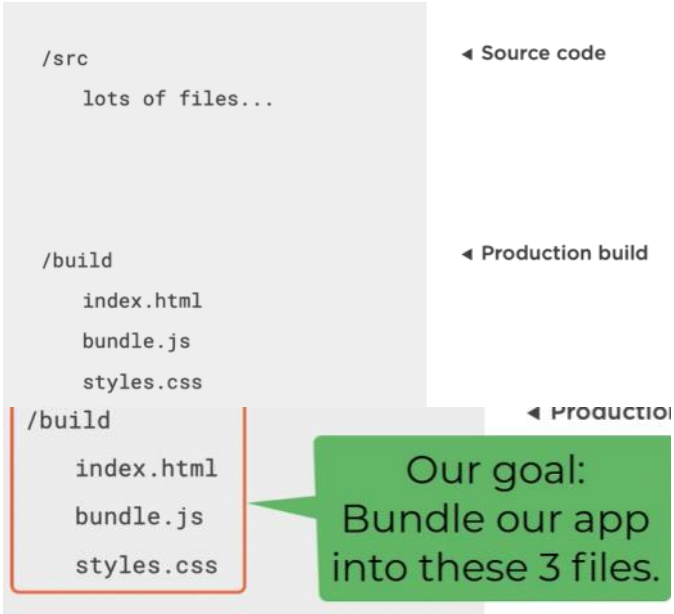


It writes a snapshots inside a separate folder with renderdd html

ALSO VS CODE EXTENSION snapshot-tools when hovered over toMatchSnapshot function gives a inline rendered code there itself







PRODUCTION BUILD:

- Modify our source code if anything we have given that is for development and testing

NODE JS BACKEND

Wednesday, July 5, 2023 4:47 PM

V8 is Google's open source high-performance JavaScript and WebAssembly engine, written in C++. It is used in Chrome and in Node.js, among others.

Asynchronous APIs(no threads)
C++ addons available in nodejs
Debugger and other utilities
Npm
Module dependency manager

(by default js will do asynchronous but we can use async await to wait for something to finish)

In javascript if we get a promise we can handle the success outcome and fail outcome using .then().catch() suntax

(util.promisify)
(require('fs').promises)

Node version manager(nvm) will make us work on multiple versions of node

https://github.com/jscomplete/ngs

REPL(Read,Eval,Print,Loop) is the shell we get when executing command node or python,python3 etc to test things before going further

.save file.js (inside repl session to save contents in a file)
.load file.js (loads from file)

Doubletab to see available functions for a Array. Or any other
Single tab to complete the word

To run node file:
Node file.js

Sample.js

```
Const http = require('http');

Const server = http.createServer((req,res)=>{
Res.end("hello world");
});

Server.listen(4242,()=>{
Console.log('server is running');
});
```

An MJS file is a source code file containing an ES Module (ECMAScript Module) for use with a Node so if we give extension as .mjs the require will not work and instead **import http from 'http';** will work.

Require etc.. Are CommonJS scripts(.cjs) and import statement etc.. Are ECMAScript module system (.mjs)

setTimeout: (waits for specified amount of time and executes the fucntion)
----
setTimeout(
()=>{
Console.log('ss');
};
,
4*1000);

With arguments:
setTimeout(func,2*1000,'Argumnetvalue');
Const func = who =>{
Console.log(who+'rocks');
};

setInterval(func,3000); # this will run every 3 secodns unless we kill it

We can clear the interval through code using :
Const timerid = setTimeout(func,0);
clearTimeout(timerid) #clearInterval(intervalid);

Let counter=0
Const intervalid = setInterval(()=>{
Console.log('helloworld');
Count+=1;
If(counter ===5){
clearInterval(intervalid);
}
},1000);

In Node CLI there will options starting with--harmony* which depicts that it is a not a tested one and it is not a final one

In node -h command it will list all the commands and environment variables that we can set to use

Eg: NODE_DEBUG="http" node file.js
The above will debug more output for http module respoinse,request and stuffs
We can give comma sepe

Inside js file we can access env vars like
process.env.VAR1
node -p "process.argv" hello 42
['nodepath','scriptname if executing','firstparam','secondparam']

process.stdout.write('message'); # to write to stdout same as console.log

Stdout and stdin and everythign are streams where we can use that to get the stdin data

process.stdin.on('readable',()=>{
Const chunk = process.stdin.read();
If(chunk !== null){

Process.stdout.write(chunk);
}
});
Or
Process.stdin.pipe(process.stdout)

Process.exit() will exit the process

Process.on('exit',()=>{}); # can get the exit call

MODERN JAVASCRIPT:

Block scope(in for loop for example after the for loop has ended we can access the I variable outside as well) for this issue we can replace var with let when declaring the variable and also we can use nested block scope with the combination of let,const(can we mutated other than array) we can have a scoped way inside a block

{
{
Let var=calculation;
}
}

Objects:

Const obj = {
P1:12,
P2:230,
F1(){},
F2:()=>{},
[mystery]:42,
PI:PI (also as PI)
};

obj.mystery (undefined)
obj.answer (42)

Rest operator:

Const [first,…restItems] = [10,20,30,40];
First = 10
restItems = [20,30,40];

Dynamic strings:(template strings)

Const html = `<div>${Math.random()}</div>`

12-async-await.js

```
const https = require('https');

function fetch (url) {
  return new Promise((resolve, reject) => {
    https.get(url, (res) => {
      let data = '';
      res.on('data', (rd) => data = data + rd);
      res.on('end', () => resolve(data));
      res.on('error', reject);
    });
  });
}

fetch('https://www.javascript.com/')
  .then(data => {
    console.log(data.length);
  });

  (async function read() {
    const data = await fetch('https://www.javascript.com/');

    console.log(data.length);
  })();

```

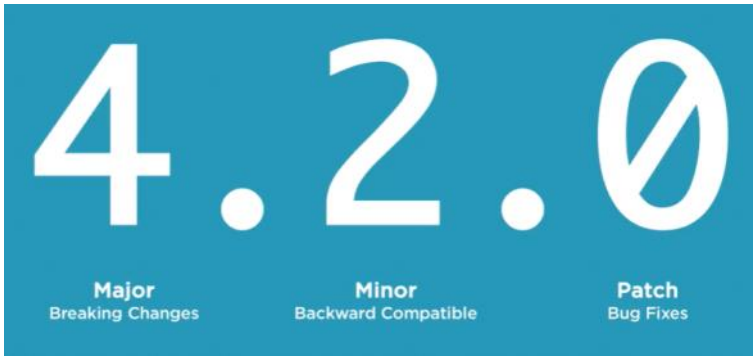
The node will use node_modules folder to search for any module
When we install a package the node will place the module files and its depenc=dencies file in node_modules folder and add an entry in the dependencies argument in package.json file

The package-lock.json will document the versions used in the project and if node_modules is deleted and run npm install command it will install the exact same version previously used even if new version is released

Npm I -D(development) nodemon
(this will add a new entry with devDependencies as the argument instead of dependencies)

(Npm I (then it will not install packages inside devDependencies argument)

Npm init (forcreating package.json file) -y(to default)



~ -> just patch versions
Eg: ~1.2.3 means 1.2.x/ from 1.2.3 till 1.2.x

^ -> minor versions
Eg: ^1.2.3 means 1.x.x/ from 1.2.3 till 1.x.x

If you give only the version that means exact version
Npm I -g create-react-app (global instead of local it will store it in some global directory)

Create and publish packages

CREATE:

(module.exports = (){} will be used if we are not destructuring the module in code)
mkdir packagename
touch index.js
Npm init
Index.js:
(
Console.log('testing….');
Module.exports = function print (msg){
console.log(`ss${msg}`);
};
)
Mkdir testpackagecreated
Touch index.js
Index.js:
(
Const print = require('../packagename');(if not given relative path it will search in node_modules directory)
Print('Hello NPM!);
)

PUBLISH:

- Create a account on npmjs website
- Npm login (username,password)
- Package.json should be there inside the directory
- Cd packagedirectoryname
- Npm publish

The scripts argument in package.json can have start,test command but while running we can use either npm run start or npm start alone

For eg: we wanna run jest for testing without giving it globally we can give it in scripts because it will not run jest command without npm scripts

If you wanna run then we can use **npx jest**

Npx eslint --init (to install local binary)

Pre and post command scripts:

"scripts":{
"test":"some command",
"posttest":"somecommand",
}

This will run posttest after test command is given

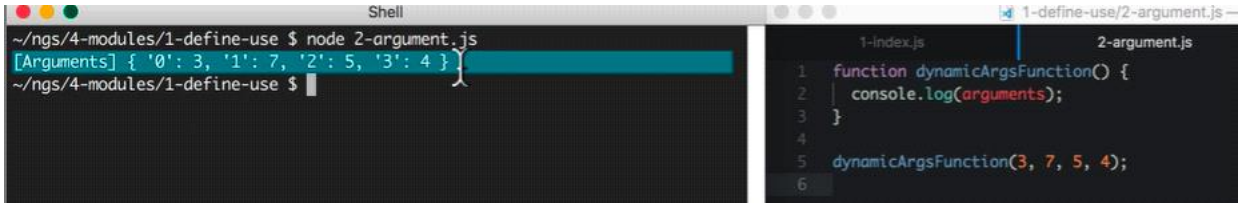
Npm ls (give dependency graph)

Npm update(to update the packages(update however specified in the package.json(caret/tilde))

Npm show loadash versions(to see available versions)

Npm outdated (will list packages that can be updated and can be installed versions it willl print out)

Npm I lodash@version (or @latest) (install specific version)



If we give only console.log(Arguments) but not inside a function since node wraps everythign inside a function it will output something like exports,module,require,__filename,__dirname as arguments

Eg:
Function (exports,module,require,__filename,__dirname){
..
Our all code
..
// return module.exports;
}

We can do the exports using both module.exports,exports

Console.dir(global,{depth:0}); (gives us the similar to dir() in python for global)

Global.answer = 42;(access in any node module)
(don't use these unless needed)

The above is synchronous action(one by one)
To make it asynchronous ise readFile(__filename,function cb(err,data){});

STREAMS,CLUSTERING ARE ADVANCED CONCPETS IN NODE
Npm install pm2 -g (this will do production processs manager)

EVENT EMITTERS:

```
Const EventEmitter = require('events');

Const myEmitter = new EventEmitter();

myEmitter.emit('TEST_EVENT');

myEmitter.on('TEST_EVENT',()=>{
Console.log('TEST event was fired');
});
```

myEmitter.emit('TEST_EVENT'); (should be given after declaring .on() function to work or use setImmediate(()=>{myEmitter.emit('TEST_EVENT');}); to run this after others are done)

Nodemon is used to make the changes to see as we type in the code instead of exiting and coming up

METEOR is an open source platform for web , mobile and desktop

Using express we can do server.render('filename without extension') to render html files
Also we can give <%- jscode -%> to give dynamic html files

WORKING WITH OS:

- **os**
- **fs**
- **child_process(spawn .bat,.cmd on windows)(.spawn() fuction in child_process module)**

DEBUGGING:

Node --inspect-brk file.js

Go to chrome://inspect in browser and go to node process listed there

NODEJS AND EXPRESS

Thursday, July 6, 2023 6:42 PM

Nvm install node
Nvm install 15(node 15 version)
Nvm install --lts

Setting up express:

```
const express  = require('express')

const app = express();

app.get('/',(req,res)=>{
 console.log('get /');
res.send('Hello from my app');
});

app.listen(3000,()=>{console.log('');})
```

_____

Chalk module is used to colorize texts on console/stdout
(console.log(require('chalk').green('2000')) - npm install chalk

Debug module lets us debug code (npm install debug)
Const debug = require('debug')('app') #need to give DEBUG=app in package.json scripts
App.listen(30000,()=>{
Debug('listening');
});
DEBUG=* node file.js

Npm install morgan (print web traffic)
Const morgan = require('morgan')
App.use(morgan('combined');

Render html page:

Const express = require('express')
Const path = require('path')
Const app = express();
App.use(express.static(path.join(__dirname,'/public'))); # will server in "/" if filename is index.html

In npm scripts anything other than start,test should have run in between in it when executing

Nodemon:

In package.json we can give nodemonConfig to give some parameters
"nodemonConfig":{
"restartable":"rs", # if we press rs then nodemon will restart
"delay":2500,
"env":{
"PORT":4000 # can use this in js file as *process.env.PORT*
}
}

TEMPLATING:
----
npm install ejs
app.set('views','./src/views');
app.set('view engine','ejs');
App.get('/',(req,res)=>{
Res.render('index',{title:'Welcome to glomantics',data:[10,20,30]});
});

In the html index.ejs give <%=title%> in place where we should produce dynamic content
Give <% data.map((i)=>{
%>
<%=i%>
<%
})
%>

<%- include('header') %>

<h1><%=title%></h1>
Profile

transfer
<%- include('footer') %>
(above to incude another ejs file into another ejs file
<%- include('summary', { account: accounts.checking }) %>
  <%- include('summary', { account: accounts.savings }) %>
  <%- include('summary', { account: accounts.credit }) %>
(above with variables)
ROUTER:

```
index.ejs   app.js
app.js > get() callback
const chalk = require('chalk');
const debug = require('debug')('app');
const morgan = require('morgan');
const path = require('path');

const PORT = process.env.PORT || 3000;
const app = express();
const sessionsRouter = express.Router();

app.use(morgan('tiny'));
app.use(express.static(path.join(__dirname, '/public/')));

app.set('views', './src/views');
app.set('view engine', 'ejs');

sessionsRouter.route('/')
  .get((req,res)=>{
    res.send('hello sessions');
  })

sessionsRouter.route('/1')
  .get((req,res)=>{
    res.send('hello single sessions');
  })

app.use('/sessions', sessionRouter);
```

In the above one sessionsRouter should come in 27th line
(this works for /sessions/* everything after /sessions)

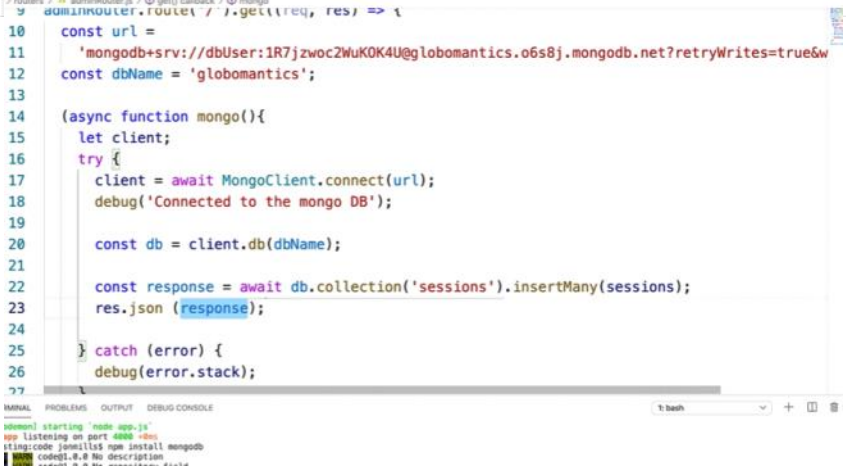
```
sessionsRouter.route('/:id').get((req, res) => {
  const id = req.params.id;
  res.send('hello single session ' + id);
});
```

To get dynamic parameter

```
<div class="container">

  <% sessions.map((session, index)=>{%>
      <div class="col-xs-12 col-sm-4 col-md-4 adj_text">

        <h3><%=session.title%></h3>
        <p><%=session.description.slice(0,300)%></p>

          <a href="/sessions/<%=index%>" class="btn-oval">Learn More</a>

      </div>
  <% }) %>

</div>
</section>
```

```
const { MongoClient } = require('mongodb');
const sessions = require('../data/sessions.json');

const adminRouter = express.Router();

adminRouter.route('/').get((req, res) => {
  const url =
    'mongodb+srv://dbUser:1R7jzwoc2WuKOK4U@globomantics.o6s8j.mongodb.net?retryWrites=tru
  const dbName = 'globomantics';

  (async function mongo(){
    let client;
    try {
      client = await MongoClient.connect(url);
      debug('Connected to the mongo DB');
```

```
const { MongoClient } = require('mongodb');
const sessions = require('../data/sessions.json');

const adminRouter = express.Router();

adminRouter.route('/').get((req, res) => {
  const url =
    'mongodb+srv://dbUser:1R7jzwoc2WuKOK4U@globomantics.o6s8j.mongodb.net?retryWrites=tru
  const dbName = 'globomantics';

  (async function mongo(){
    let client;
    try {
      client = await MongoClient.connect(url);
      debug('Connected to the mongo DB');

      const db = client.db(dbName);
```



Here we are doing (async function mongo(){…await something()}())
In the adminrouter we can add the /:id route to get the id and findOne() in mongodb and give that session value to ejs

PASSPORT(LOGIN,AUTH):

```
<form name="signup" action="/auth/signUp" method="post">
  username: <input name="username" id="username"/><br/>
  password: <input name="password" id="password"/><br/>
<input type="submit" class="btn btn-primary btn-lg">SIGN UP TODAY</in>

</form>
```

Implement a authrouter same as before and .post() for getting the data with req.body

```
app.use(express.json());
app.use(express.urlencoded({extended:false}));
```

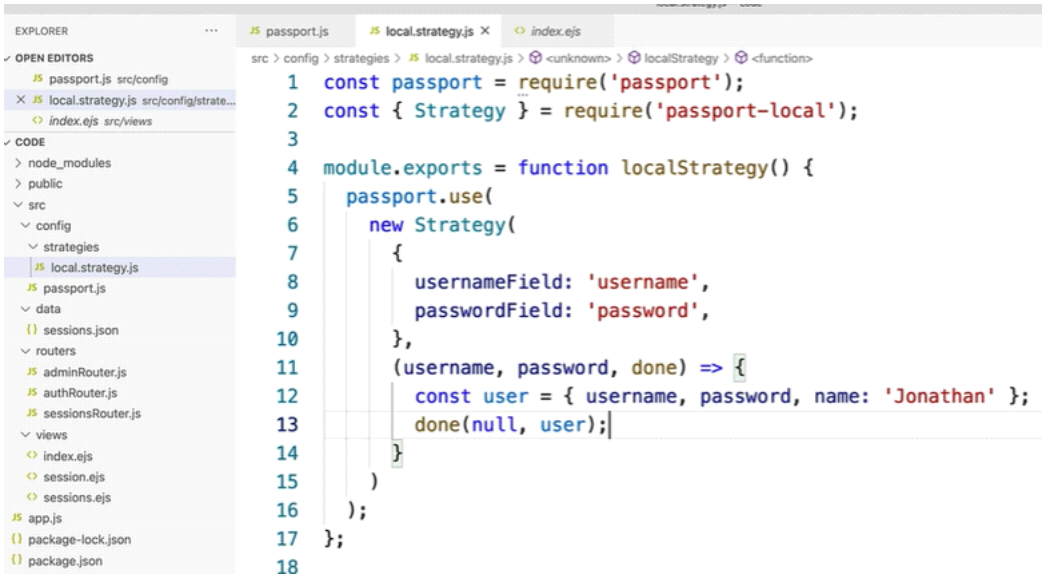
Setup passport:

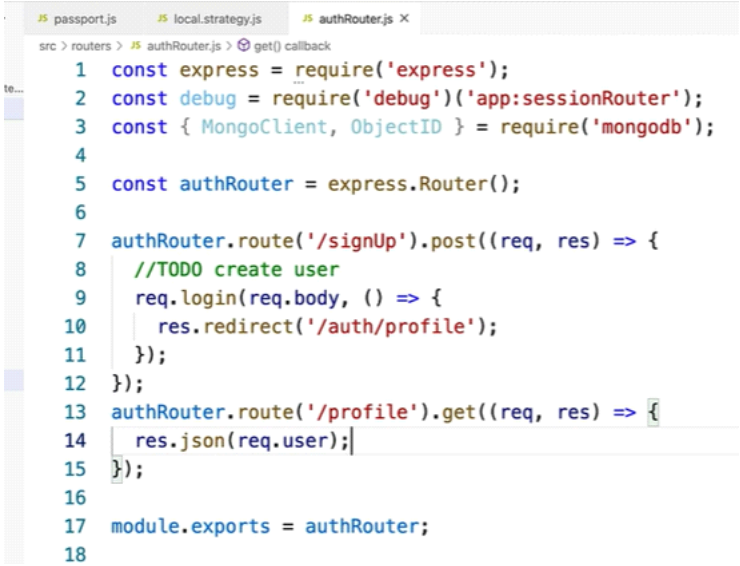
```
const passport = require('passport');
const cookieparser = require('cookie-parser');
const session = require('express-session');

app.use(cookieparser());
app.use(session({secret:'secret'}));

require('./src/config/passport.js')(app)
```

_____

/src/config/passport.js

```
passport.js ●
src > config > passport.js > ...
1  const passport = require('passport');
2  require('./strategies/local.strategy')();
3
4  module.exports = function passportConfig(app) {
5    app.use(passport.initialize());
6    app.use(passport.session());
7
8    passport.serializeUser((user, done) => {
9      done(null, user);
10   });
11
12   passport.deserializeUser((user, done) => {
13     done(null, user);
14   });
15 };
16
```





Inside 8th line above put this to create a user

```
const {username, password} = req.body;
const url =
  'mongodb+srv://dbUser:1R7jzwoc2WuKOK4U@globomantics.o6s8j
const dbName = 'globomantics';

(async function addUser(){
  let client;
  try {
    client = await MongoClient.connect(url);

    const db = client.db(dbName);
    const user = {username, password};
    const results = db.collection('users').insertOne(user);
    debug(results);

  } catch (error) {
```

In the req.login now we can give results.ops[0]

```
authRouter
  .route('/signIn')
  .get((req, res) => {
    res.render('signin');
  })
  .post(
    passport.authenticate('local', {
      successRedirect: '/auth/profile',
      failureMessage: '/',
    })
  );
```

To create our middleware we can use the below one to give next() after we validate something

```
const sessionsRouter = express.Router();
sessionsRouter.use((req, res, next) => {
  if (req.user) {
    next();
  } else {
    res.redirect('/auth/signIn');
  }
});
```

In the passport localstrategy code we have not done any db connect we can add below like code to validate

```
let client;
try {
  client = await MongoClient.connect(url);
  debug('Connected to the mongo DB');

  const db = client.db(dbName);

  const user = await db.collection('users').findOne({ username });

  if (user && user.password === password) {
    done(null, user);
  } else {
    done(null, false);
  }
} catch (error) {
  done(error, false);
}
client.close();
```

MONGODB AND NOSQL

Friday, July 7, 2023 7:24 PM

NOSQL - NOT ONLY SQL

C Consistency

A Availability

P Partition Tolerance

Types of NoSQL Databases

SQL Terms vs MongoDB Terms

SQL Terms	MongoDB Terms
Database	Database
Table	Collection
Row	JSON / BSON Document
Column	Field
Index	Index

JSON vs BSON

	JSON (JavaScript Object Notation)	BSON (Binary JSON)
Encoding	UTF-8 String	Binary
Data Support	String, Boolean, Number, Array	String, Boolean, Number (Integer, Float, Long, Decimal128...), Array, Date, Raw Binary
Readability	Human and Machine	Machine Only

CRUD – Create, Read, Update, Delete

C	Create or Insert	db.collection.insertOne()
R	Read or Find	db.collection.find()
U	Update	db.collection.updateOne()
D	Delete	db.collection.deleteOne()

db.collection.insertOne()

db.collection.insertMany()

Things to Remember for Insert Behavior

All write operations in MongoDB are atomic on the level of a single document

If the collection does not currently exist, insert operations will create the collection

If an inserted document omits the _id field, the MongoDB driver automatically generates an ObjectId for the _id field

mongodb.txt | SampleData.json | commandprompt.txt

```
/*Connect to MongodB Atlast
Use your servername and username/password
*/
mongo "mongodb+srv://sqlauthority-ckafd.mongodb.net/test"  --username newuser --password newpassword

/*Create Database*/
USE sqlauthority

/* Show dbs */
show dbs

/* Show current Db */
db

/* create collection */
db.createCollection("newusers")

/* show collection */
db.createCollection("show collections")

/* insert single document */
db.newusers.insertOne(
{
    "DisplayName": "Pinal Dave",
    "UserName": "pinaldave",
    "Job": {
        "Title": "DBA",
        "Area": "Database Performance Tuning",
        "isManager": "false"
    },
    "Programming Languages": ["T-SQL", "JavaScript", "HTML"]
}
)
```

Db.newusers.find({}).pretty() # pretty for formatting output

```
db.newusers.insertMany(
[
{
    "DisplayName": "Pinal Dave",
    "UserName": "pinaldave",
    "Job": {
        "Title": "DBA",
        "Area": "Database Performance Tuning",
        "isManager": "false"
    },
    "Programming Languages": ["T-SQL", "JavaScript", "HTML"]
},
{
    "DisplayName": "Jason Brown",
    "UserName": "jasonbrown",
    "Job": {
        "Title": "DBA",
        "Area": "Database Performance Tuning",
        "isManager": "true"
    },
    "Programming Languages": ["T-SQL", "JavaScript", "HTML"]
},
```

Query and Query Operators

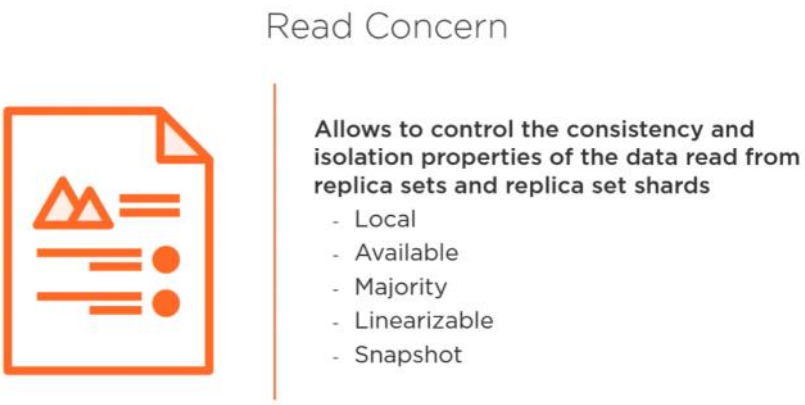
Find method supports many different query operators to filter data

- Comparison
- Logical
- Element
- Evaluation
- Geospatial
- Array
- Bitwise

Query Projection

Specifies the fields to return in the documents that match the query filter

- 1 or true : include the field
- 0 or false : exclude the field



{runtime: 11} to filter documents with runtime=11
{runtime: {$gt: 11}} to filter docs greater than 11
$lte - less than or equalto

{cast: "Billy"} - here cast is a array in document (filter where Billy was one in the array list

{"awards.wins":3} "awards":{"wins":3} -

{$and: [{runtime: {$gt: 40}}, {"awards.wins":3}]} - to specify multiple filters

In the OPTIONS>PROJECT we can give {title:1,runtime:1,"Awards.wins":1} to display only those fields
_id:0 will remove id field in result

Db.movies.find({runtime:11}).pretty().limit(3)
Db.movies.find({runtime:11} , {title:1,runtime:1,"Awards.wins":1}).pretty().limit(3)
Db.movies.find({runtime:11} ,
{title:1,runtime:1,"Awards.wins":1}).pretty().limit(3).sort({title:1}).count()
Db.movies.find({runtime:11} , {title:1,runtime:1,"Awards.wins":1}).pretty().limit(3).sort({title:-1})

Find({filter} , {projection})

Db.movies.find({runtime:11} ,
{title:1,runtime:1,"Awards.wins":1}).pretty().limit(3).sort({title:-1}).readConcern("majority")

Db.movies.find({runtime:11} ,
{title:1,runtime:1,"Awards.wins":1}).pretty().limit(3).sort({title:-1}).readConcern("majority").maxtim
eMS(10000) # to specify timeout

Write Concern

Level of acknowledgement requested from MongoDB for write operations

w:1 – Ack from primary

w:0 – No ack

w:(n) – Primary + (n-1) secondary

w: majority

wtimeout: Time limit to prevent write operations from blocking indefinitely

Things to Remember for Update Behavior

Atomic on the level of a single document

_id field cannot be replaced with different value

$set creates field if not already existing

upsert : true

- Update on match of filter
- Insert no match of filter

```
db.movies.updateOne(#updateMany(
{title: {$eq: "Old Crocodile"}),
{
 $set: {"title":"The new crocodile"}
},
{upsert:true, w:"majority", wtimeout:1000} # or false in upsert
)
```

```
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.replaceOne(
...     {runtime: {$eq: 1122}},
...     { runtime: 1122, "NoTitle": "ReplaceOneExample", "NewYear": 2020, "awards.losts": 5 }
... )
```

Things to Remember for Delete Behavior

All write operations in MongoDB are atomic on the level of a single document

Delete does not drop indexes

db.collection.deleteOne()

db.collection.deleteMany()

db.collection.remove()

In remove() function if we pass true then it will delete only one document in the colleciton

```
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.find( {runtime: 25} ).count()
11
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.deleteOne( {runtime: 25} )
{ "acknowledged" : true, "deletedCount" : 1 }
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.find( {runtime: 25} ).count()
10
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.deleteOne( {runtime: 25} )
{ "acknowledged" : true, "deletedCount" : 1 }
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.deleteOne( {runtime: 25} )
{ "acknowledged" : true, "deletedCount" : 1 }
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.find( {runtime: 25} ).count()
8
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.deleteMany( {runtime: 25} )
{ "acknowledged" : true, "deletedCount" : 8 }
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.find( {runtime: 25} ).count()
0
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.find( {runtime: 35} ).count()
9
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY> db.movies.remove( {runtime: 35}, true )
WriteResult({ "nRemoved" : 1 })
MongoDB Enterprise SQLAuthority-shard-0:PRIMARY>
```

.db.movies.remove({}) - to delete everything

MySQL

```
CREATE INDEX
       idx_user_name_age
ON user(name, age DESC)
```

MongoDB

```
db.people.createIndex(
{ name: 1, age: -1 }
)
```

MySQL

```
DROP TABLE user
```

MongoDB

```
db.user.drop()
```

MONGOOSE

Friday, July 7, 2023 8:05 PM

Mongoose is the officially supported ODM for Node.js. It has a thriving open source community and includes advanced schema-based features such as async validation, casting, object life-cycle management, pseudo-joins, and rich query builder support.

Object document mapper(ODM)

MODELS/SCHEMA FOR DOCUMENTS:

Simple Schema Example

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var customerSchema = new Schema({
        name:       String,
        address:    String,
        city:       String,
        state:      String,
        country:    String,
        zipCode:    Number,
        createdOn:  Date,
        isActive:   Boolean
});

var simpleSchema = new Schema({ fieldName: SchemaType });
```

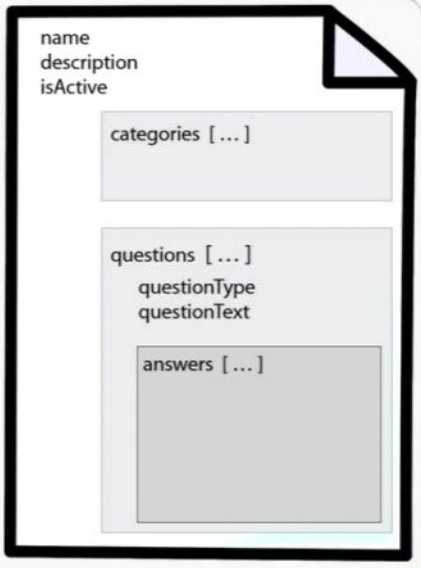
Allowed Data Types

Mongoose Schema Types	JavaScript Data Types
String	String
Number	Number
Date	Object
Buffer	Object
Boolean	Boolean
Mixed	Object
ObjectId	Object
Array	Array (Object)

More Complex Schema Example

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// child address schema...
var addressSchema = new Schema({
    type: String,
    street: String,
    city: String,
    state: String,
    country: String,
    postalCode: Number
});

// parent customer schema...
var customerSchema = new Schema({
    name: {
        first: String,
        last: String
    },
    address: [ addressSchema ],
    createdOn: { type: Date, default: Date.now },
    isActive: { type: Boolean, default: true},
});
```

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// Pre-define sub-documents...
var subCategory = {
        name: String,
        description: String,
        isActive: Boolean };

var subAnswers = {
        answerText: String,
        isCorrect: Boolean,
        displayOrder: Number };

var subQuestions = {
        questionType: String,
        questionText: String,
        answers: [ subAnswers ] };

// Define main document schema...
var quizSchema = new Schema({
    name: String,
    description: String,
    categories: [ subCategory ],
    questions: [ subQuestions ]
});
```




```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

// Pre-define sub-documents...
var subCategory = {
        name: String,
        description: String,
        isActive: Boolean };

var subAnswers = {
        answerText: String,
        isCorrect: Boolean,
        displayOrder: Number };

var subQuestions = {
        type: { type: String },
        text: String,
        answers: [ subAnswers ] };

// Define main document schema...
var quizSchema = new Schema({
    name: String,
    description: String,
    categories: [ subCategory ],
    questions: [ subQuestions ]
});
```

```
{
  "_id" : ObjectId("5422cfdb36663a0424d7a66b"),
  "name" : "Test Quiz",
  "description" : "Test Quiz long description...",
  "questions" : [
    {
      "type" : "Multiple-Choice",
      "text" : "What is your favorite color?",
      "_id" : ObjectId("5422cfdb36663a0424d7a670"),
      "answers" : [
        {
          "answerText" : "Red",
          "isCorrect" : false,
          "displayOrder" : 1,
          "_id" : ObjectId("5422cfdb36663a0424d7a673")
        },
        // abbreviated for brevity sake ...
    ],
    "categories" : [
        // abbreviated for brevity sake ...
    ]
}
```

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var standupSchema = new Schema({
    memberName: String,
    project: String,
    workYesterday: String,
    workToday: String,
    impediment: String,
    createdOn: { type: Date, default: Date.now }
});

// disabled _id
var noIdSchema = new Schema({ name: String}, {_id: false});

// Example of using Schema.add ...
var exampleSchema = new Schema;

exampleSchema.add({
    project: String,
    workYesterday: String,
    workToday: String,
    impediment: String,
    createdOn: { type: Date, default: Date.now }
});
```

Our First Model

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var customerSchema = new Schema({
        name:      String,
        address:   String,
        city:      String,
        state:     String,
        country:   String,
        zipCode:   Number,
        createdOn: Date,
        isActive:  Boolean
});

// Build a model from the customer schema…
var Customer = mongoose.model('Customer', customerSchema);

// Add a custom property to the schema…
customerSchema.add({ discountCode: String });

var DiscountedCust = mongoose.model('DiscountCust', customerSchema);
```

```
var personSchema = new Schema({
    firstName: String,
    lastName: String
});

var Person = mongoose.model('Person', personSchema);

var bob = new Person({
    firstName: 'Bob',
    lastName: 'Doe'
});
```

SAVING THE DOCUMENT USING THE MODEL,SCHEMA CREATED:

---

bob.save(callback); # to save the document

```
var categories = [];
// Fill in the categories array
var cat1 = { name: 'Test1', description: 'Test 1 category', isActive: true };
var cat2 = { name: 'Test2', description: 'Test 2 category', isActive: true };

categories.push(cat1, cat2);

var questions = [];
// Add in some questions and answers...
var q1 = { type: 'Multiple-Choice', text: 'What is your favorite color?',
     answers: [
          { answerText: 'Red', isCorrect: false, displayOrder: 1 },
          { answerText: 'White', isCorrect: false, displayOrder: 2 },
          { answerText: 'Blue', isCorrect: true, displayOrder: 3 }
     ] };

var q2 = { type: 'Multiple-Choice', text: 'What is your favorite animal?',
     answers: [
          { answerText: 'Dog', isCorrect: true, displayOrder: 1 },
          { answerText: 'Cat', isCorrect: false, displayOrder: 2 },
          { answerText: 'Squirrel', isCorrect: false, displayOrder: 3 }
     ] };

questions.push(q1, q2);

// Create the parent quiz document - supply the categories and questions now...
var quiz2 = new Quiz({
    name: 'Example Quiz',
    description: 'Example Quiz long description...',
    categories: categories,
    questions: questions });
```

```
var express = require('express');
var router = express.Router();
var standupCtrl = require('../controllers/standup.server.controller');

/* GET home page. */
router.get('/', function(req, res) {
  res.render('index', { title: 'Express' });
});

/* GET New Note page. */
router.get('/newnote', function(req, res) {
    return standupCtrl.getNote(req, res);
});

/* POST New Note page. */
router.post('/newnote', function(req, res) {
    return standupCtrl.create(req, res);
});

module.exports = router;
```

```
var Standup = require('../models/standup.serve

exports.create = function(req, res) {
    var entry = new Standup({
        memberName: req.body.memberName,
        project: req.body.project,
        workYesterday: req.body.workYesterday,
        workToday: req.body.workToday,
        impediment: req.body.impediment
    });

    entry.save();

    // redirect to home page...
    res.redirect(301, '/');
};
```

FUCNTIONS/METHODS AVAILABLE TO DO OPERATIONS:

Model.find(conditions, [fields], [options], [callback])

```
var Standup = require('../models/standup.server.model.js');

// No callback... Deferred execution
var query = Standup.find();

// With callback... Executes query immediately
Standup.find(function (err, results) {
    // handle the error... Or results here
});

// With callback and query conditions
Standup.find({ memberName: 'David' }, function (err, results) {
    // handle the error... Or results here
});

// Limit the returned fields...
Standup.find({ memberName: 'Mary' }, 'memberName impediment',
  function (err, results) {
        // handle the error... Or results here
});
```

memberName impediment are the fields to be listed in above last find

```
// No callback... No conditions...
var query = Standup.findOne();
query.exec(function (err, results) {
    // handle the error... Or results here
});

// With conditions...
var query = Standup.findOne({ memberName: 'Mark'});
```

Model.findById(id, [fields], [options], [callback])

```
var id = '541c6fefefdf9c84172162a6';
var Standup = require('../models/standup.server.model.js');

// By Id... No conditions...
var query = Standup.findById(id);
query.exec(function (err, doc) {
    // handle the error... Or results here
});

// Same as above... Chained method calls...
Standup.findById(id).exec(function (err, results) { ... });

// By Id... Return every field BUT impediment...
var query = Standup.findById(id, '-impediment');
```

$gt	greater than
$gte	greater than or equal t
$in	exists in
$lt	less than
$lte	less than or equal to
$ne	not equal to
$nin	does not exist

Model.where(path, [val])

```
Customer.find({discount: {$gte: 10, $lt: 20}}, function(err, results) {
    if (err) throw err;
    console.log(results);
});

Customer.where('discount').gte(10).lt(20).exec(function(err, results) {
    if (err) throw err;
    console.log(results);
});

// Chain where methods together...
Customer.where('discount').gte(10).lt(20)
        .where('zipCode', '12345')
        .exec(function(err, results) {
            if (err) throw err;
            console.log(results);
        });
```

Updating and Removing Documents

Update

Remove

findByIdAndUpdate

findByIdAndRemove

```
Standup.findById(id).exec(function (err, doc) {
    // handle any errors
    if (err) return errorHandler(err);

    // update the found document
    doc.impediment('None');
    doc.save(function (err) {
        if (err) return errorHandler(err);
        console.log('Document updated');
    });
});
```

Model.update(conditions, update, [options], [callback])

```
var Standup = require('../models/standup.server.model.js');

var condition = { memberName: 'Mary' };
var update = { impediment: 'None - Mary no longer works here!' };

Standup.update(condition, update, function(err, numberAffected, rawResponse)
    // Handle error or raw results here...
});

// finding a document - then updating it...
Standup.findOne({ memberName: 'Mary' }, function(err, doc) {
    // Handle errors here... Validate document results... Etc.
    doc.impediment = 'None - Mary won the lottery and is on an island now';
    doc.save(function (err) {
        // Handle errors
    });
});
```

Model.update(conditions, update, [options], [callback])

Option	Description	Default Value
safe	Safe mode – default to value set in schema	True
upsert	Create the document if it does not match	False
multi	Determines if multiple documents should be updated or not	False
strict	Override the strict option for this update	
overwrite	Disables the update-only mode to allow for overwrite of document	False

```
// Example: update multiple documents that match condition
var condition = { firstName: 'Bob' };
var update = { firstName: 'Robert' };

Customer.update(condition, update, { multi: true },
function(err, numberAffected, raw) {
   // Handle error, returned # affected, and raw response from MongoDB...
});
```

Model.remove(conditions, [callback])

```
var Standup = require('../models/standup.server.model.js');

var condition = { memberName: 'Mary' };

Standup.remove(condition, function(err) {
    // Handle error here...
});

// Remove any document created on or after Halloween day
var gteDate = new Date(2014, 10, 31);
Standup.remove({ createdOn: { $gte: gteDate }}, function (err) {
    // Handle error here...
});

// Execute query w/o a callback function - does not wait on response
var query = Standup.remove({ createdOn: { $gte: gteDate }});
query.exec();
```

Model.findByIdAndUpdate(id, update, [options], [callback])

Option	Description	Default Value
new	Set to true to return the modified document rather than the original.	True
upsert	Create the document if it does not match	False
select	Specify the document fields to return	

Model.findByIdAndRemove(id, [options], [callback])

Option	Description	Default Value
select	Specify the document fields to return	

VALIDATORS(BUILTIN,MODDLEWARE,CUSTOM):

Schema Type	Built-in Validators		
String	required	enum	match
Number	required	min	max
Date	required		
Buffer	required		
Boolean	required		
Mixed	required		
ObjectId	required		
Array	required		

```
// Required Validator Example
var customerSchema = new Schema({
        name:       { type: String, required: true },
        address:    String,
        city:       String,
        state:      String,
        country:    String,
        zipCode:    Number,
        createdOn:  Date,
        isActive:   Boolean
});

// After the schema is defined - via the path API
customerSchema.path('city').required(true, 'Oops! Supply a city.');

// Signature = required(required, [message])
```

Strings – Match and Enum

```
// String - Match Validator Example
var reMatch = /[a-zA-Z]/;
var customerSchema = new Schema({
    name: { type: String,
            required: true,
            match: reMatch },

        // abbreviated...
});

// String - Enum Validator Example
var impediments = ['none', 'minor', 'blocking', 'severe'];

var standupSchema = new Schema({
    // abbreviated...
    impediment: { type: String,
                  required: true,
                  enum: impediments }
});
```

Numbers

```
// Customers must receive at least a 5% discount
var customerSchema = new Schema({
        name:       String,
        // ...
        discount: { type: Number, min: 5 }
});

// Customers not allowed more than 60% discount
var customerSchema = new Schema({
        name:       String,
        // ...
        discount: { type: Number, max: 60 }
});

// Customers allowed a discount between 5% and 60% only
var customerSchema = new Schema({
        name:       String,
        // ...
        discount: { type: Number, min: 5, max: 60 }
});
```

CUSTOM VALIDATOR:

```
// Custom validation - method signature = validate(obj, [errorMsg])
var sizeValidator = [
    function (val) {
        return (val.length > 0 && val.length <= 50)
    },
    // Custom error text...
    'String must be between 1 and 50 characters long' ];

var personSchema = new Schema({
    firstName: { type: String, required: true, validate: sizeValidator },
    lastName:  { type: String, required: true, validate: sizeValidator },
    status:    { type: String, required: true, default: 'Alive' }
});

// Build a model from the person schema
var Person = new mongoose.model('Person', personSchema);

// New document instance of a Person model
var newPerson = new Person( { firstName: 'John', lastName: 'Doe' } );

// Save the document... And validate
newPerson.save(function (err) {
  if (err) return handleError(err);
  // saved the person document!
});
```

```
var msg = module.exports = exports = {};

msg.general = {};
msg.general.default = "Validator failed for path '{PATH}' with value
'{VALUE}'";
msg.general.required = "Path '{PATH}' is required.";

msg.Number = {};
msg.Number.min = "Path '{PATH}' ({VALUE}) is less than minimum allowed
value ({MIN}).";
msg.Number.max = "Path '{PATH}' ({VALUE}) is more than maximum allowed
value ({MAX}).";

msg.String = {};
msg.String.enum = "'{VALUE}' is not a valid enum value for path
'{PATH}'.";
msg.String.match = "Path 'PATH}' is invalid ({VALUE}).";
```

{PATH} {VALUE} {TYPE} {MIN} {MAX}

```
var mongoose = require('mongoose');
mongoose.Error.message.Number.min = "{PATH} is too low! Must be at least
{MIN}.";
```

```
var entry = new Standup({
    memberName: req.body.memberName,
    project: req.body.project,
    workYesterday: req.body.workYesterday,
    workToday: req.body.workToday,
    impediment: req.body.impediment
});

// validate now...
entry.schema.path('memberName').validate(function (value) {
    return value != 'None';
}, 'You must select a team member name.');

entry.save();

// redirect to home page...
res.redirect(301, '/');
```