matrix

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

React

מתחילים עוד כמה דקות

Gal Amouyal gal.amouyal@ibm.com https://github.com/galamo

matrix

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

React

JavaScript library for building user interfaces

Agenda

- SPA
- JavaScript & async operations
- JavaScript single threaded
- Event loop
- Functional programing
- JavaScript for react
  - Objects Copy
  - Async await
  - Higher order functions
- Imperative vs declarative
- ECMA Latest features
- Development tools - vite

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

Agenda

- TypeScript
  - Interfaces
  - Types
  - Generics
  - Type declaration (intersection & Union)

Agenda

- React Concept
- TSX
- Folder recommendation structure
- Component architecture
- The virtual DOM
- Diffing Algorithm

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

Agenda

- Basic components
- State & props
- Stateless VS statefull components
- Function components
- Managing component state
- High Order Components
  - Children
  - Render props

Agenda

- React Hooks
  - useState
  - useEffect
  - useMemo
  - useCallback
  - useDefferedValue
  - useTransition
  - useRef (useRef vs useState)

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

Agenda

- Lazy loading
- Managing global state with useContext & useReducer
- Using Axios interceptors
- Using redux toolkit vs Managing state with useContext & useReducer

Agenda

- When should we use MF’s
- Major features distinction
- Advantages
- React solution
- Webpack Or Vite configuration
- Example of MF’s with react
  - Build time integration - libraries
  - Run time integration – MF’s

Eco System

- Routing – React router DOM
- React redux
- Material Design / Bootstrap

DevGeekWeek 2024
INSPIRED BY INNOVATION

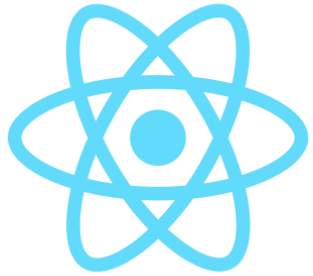
JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

References

- React Official Docs - https://react.dev/

React VS Angular

React



Google	Facebook 2011
JavaScript Framework	JavaScript Library
Full MVC	View
Regular DOM	Virtual DOM
Bi-Directional	Uni-directional
TypeScript, HTML	JSX, JS, Typescript

About React

- React is a JavaScript library - one of the most popular ones, with over 140,000 stars on GitHub
- Open-source project
- React is used to build user interfaces (UI) on the front end.
- View layer
- Declarative
- Efficient
- Component based

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

React’s past and future

- Fiber engine
- Hooks
- New lifecycle
- Concurrent
- Suspense
- New Dev tools

https://reactjs.org/blog

Before we start…


LETS PLAY A GAME...
memegenerator.net

Do we know JavaScript ?

What is the Call Stack ? Is it part of V8 ?

- Part of V8 Engine (in case chrome & nodejs)
- Model that implemented by different browsers
- Use to keep track on function invocations

Busy Call Stack

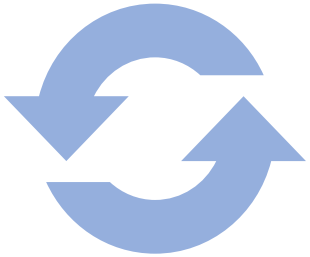
```
const f1 = () => { f2(); };

const f2 = () => { f3(); };

const f3 = () => { f4(); };

const f4 = () => { f4(); };
```

What is the Event Loop? Is it part of V8 ?



Event Loop

- Handles external events and convert them into callback invocation

Event Loop

- A loop that picks events from the event queue and pushes their callbacks to call stack

Event Loop - example

- Example

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

JavaScript for react

- ECMA Latest features
- Const
- Let
- Template strings
- Arrow functions
- Destructuring Objects & Arrays
- Object literals
- Async/Await
- Es6 modules

Functional Programming

- JavaScript functions are first class citizens
- Last ECMA new features – Arrow functions, Promises etc..
- Functions can represent data
- Higher order function
- Part of declarative programing

Imperative vs Declarative

- What should happen?

Imperative

Declarative

How it Happen?

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

Let's do a little experiment

This code …

```
const string = "What is this code doing?";
let result = "";

for (let i = 0; i < string.length; i++) {
  if (string[i] === " ") {
    result  += "-";
  } else {
    result  += string[i];
  }
}
```

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

Ready…

This code …

```
const string = "Restaurants in Tel aviv";
const urlFriendly = string.replace(/ /g, "-");
```

And the winner is ?

Replace the string to **URL friendly**

Round 2…

Ready ?

This code …

```
const users = [”Adi", ”Noa", ”Eitan"];
const newUsers = [];

for (let index = 0; index < users.length; index++) {
  if (users[index].length > 3) {
    newUsers.push(users[index]);
  }
}
```


Easy…

...

```
const users = [”Adi", ”Noa", ”Eitan"];
const newUsers = users.filter(user => user.length > 3);
```

Filter!!!

Imperative vs Declarative - Welcome

```
const target = document.getElementById("target");
const wrapper = document.createElement("div");
const headline = document.createElement("h1");

wrapper.id = "welcome";
headline.innerText = "Hello World";

wrapper.appendChild(headline);
target.appendChild(wrapper);
```

React way

```
const { render } = ReactDOM;

const Welcome = () => (
  <div id="welcome">
    <h1>Hello World</h1>
  </div>
);

render(<Welcome />, document.getElementById("target"));
```

The render function uses the instructions declared in the component to build the DOM, abstracting away the details of how the DOM is to be rendered

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

More functional Concepts

- Immutability
- Pure functions
- Data transformations - High order function
- recursion

Immutability

- Unchangeable
- Data is immutable
- Return a Copy

mutation - example

```
let color_book = {
  title: "book",
  color: "#00FF00",
  rating: 0
};

function rateColor(color, rating) {
  color.rating = rating;
  return color;
}

console.log(rateColor(color_book, 5).rating); // 5
console.log(color_book.rating); // 5
```

Immutability - exmaple

```
const rateColor = function(color, rating) {
  return Object.assign({}, color, { rating: rating });
};

console.log(rateColor(color_book, 5).rating); // 5
console.log(color_book.rating); // 4
```

Immutability - example

```
const rateColor = (color, rating) => ({
  ...color,
  rating
});
```

Pure function

- Returns a value that is computed based on its arguments
- At least one argument
- Always return a value or another function
- No side effects
- Arguments as immutable data
- Testable

Pure function - example

```
const frederick = {
  name: "Frederick Douglass",
  canRead: false,
  canWrite: false
};

const selfEducate = person => ({
  ...person,
  canRead: true,
  canWrite: true
});

console.log(selfEducate(frederick));
console.log(frederick);
```

DOM Side effects

```
function Header(text) {
  let h1 = document.createElement("h1");
  h1.innerText = text;
  document.body.appendChild(h1);
}

Header("Header() caused side effects");
```

Pure function \ component – react way

```
const header = text => <h1> {text} </h1>;
```

Higher order functions

- Manipulate other functions
- Takes functions as arguments
- Return functions

Higher order functions

```
const invokeIf = (condition, fnTrue, fnFalse) =>
  condition ? fnTrue() : fnFalse();

const showWelcome = () => console.log("Welcome!!!");

const showUnauthorized = () => console.log("Unauthorized!!!");

invokeIf(true, showWelcome, showUnauthorized); // "Welcome!!!"
invokeIf(false, showWelcome, showUnauthorized); // "Unauthorized
!!!"
```

Data Transformation

- Reduce
- Map
- Filter
- Find
- Findindex

Data Transformation

```
const editName = (oldName, name, arr) => {
  return arr.map(item => {
    return item.name === oldName ? { ...item, name } : item
;
  });
};
```

Composition

- Small pure functions, focus each function in a specific task
- Combine functions to larger function
- Chainable

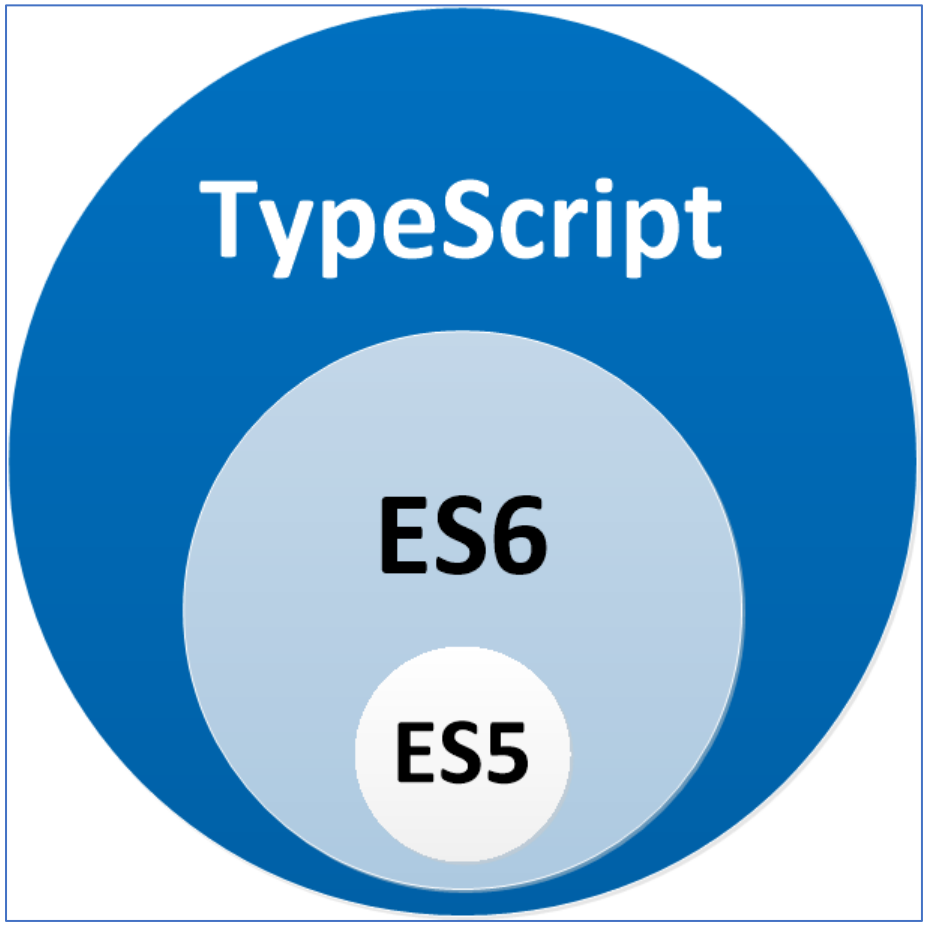
Composition

```
const civilianHours = clockTime => ({
   ...clockTime,
   hours: clockTime.hours > 12 ? clockTime.hours - 12 : clockTime.hours
 });

 const appendAMPM = clockTime => ({
   ...clockTime,
   ampm: clockTime.hours >= 12 ? "PM" : "AM"
 });

const both = date => appendAMPM(civilianHours(date));
```

Typescript



Typescript

- Microsoft
- JavaScript that scales
- Typescript compiler
- Interfaces
- Strongly types
- Classes modifiers
- More…

React core concepts

- React Element
- JSX
- Components and props
- Virtual DOM
- Render
- State
- Lifecycle

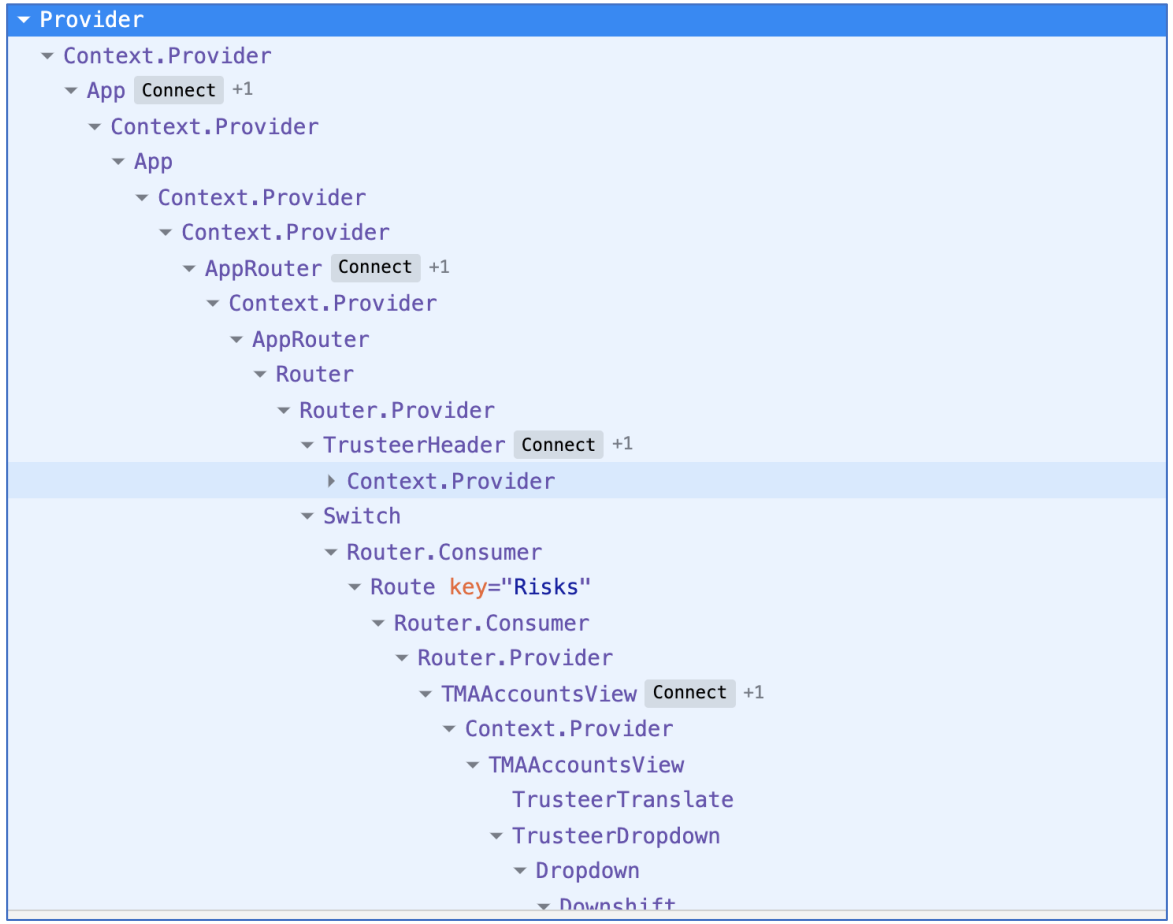
DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

Thinking in react

- Break The UI Into A Component Hierarchy
- Data Model
- Identify The Minimal Representation Of UI State
- Identify Where Your State Should Live
- Add Inverse Data Flow

React Component Hierarchy

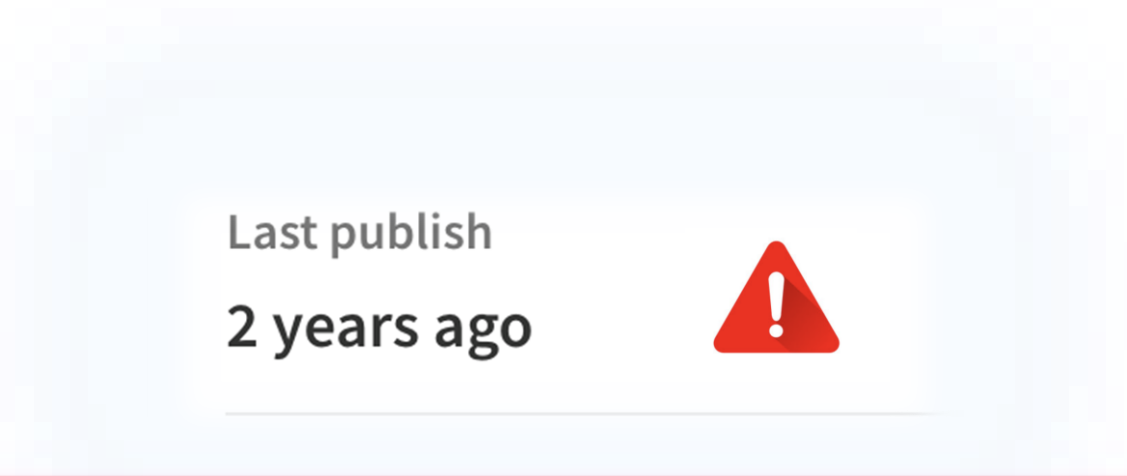


How react works

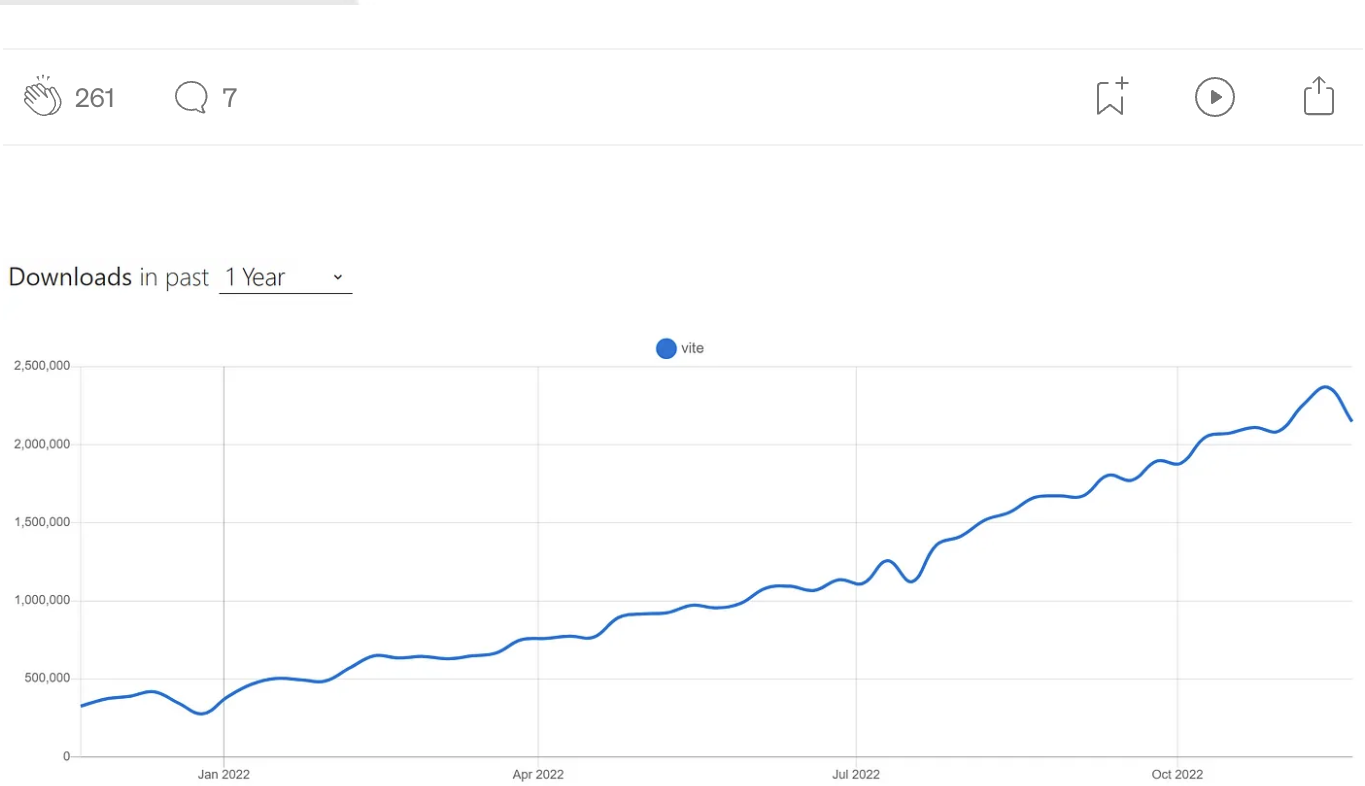
- Can use 2 minimal libs
- React - Generate views
- React DOM - render the UI in the browser
- React designed to update the DOM for us

React Create App

- Create react app is a comfortable environment for learning React and is the best way to start building a new single-page application in React.



Vite



https://npmtrends.com/vite

Vite

- Native ES modules
- Rollup

React Create App

```
npm install -g create-react-app
```

Vite new project

https://vitejs.dev/guide/

React CLI

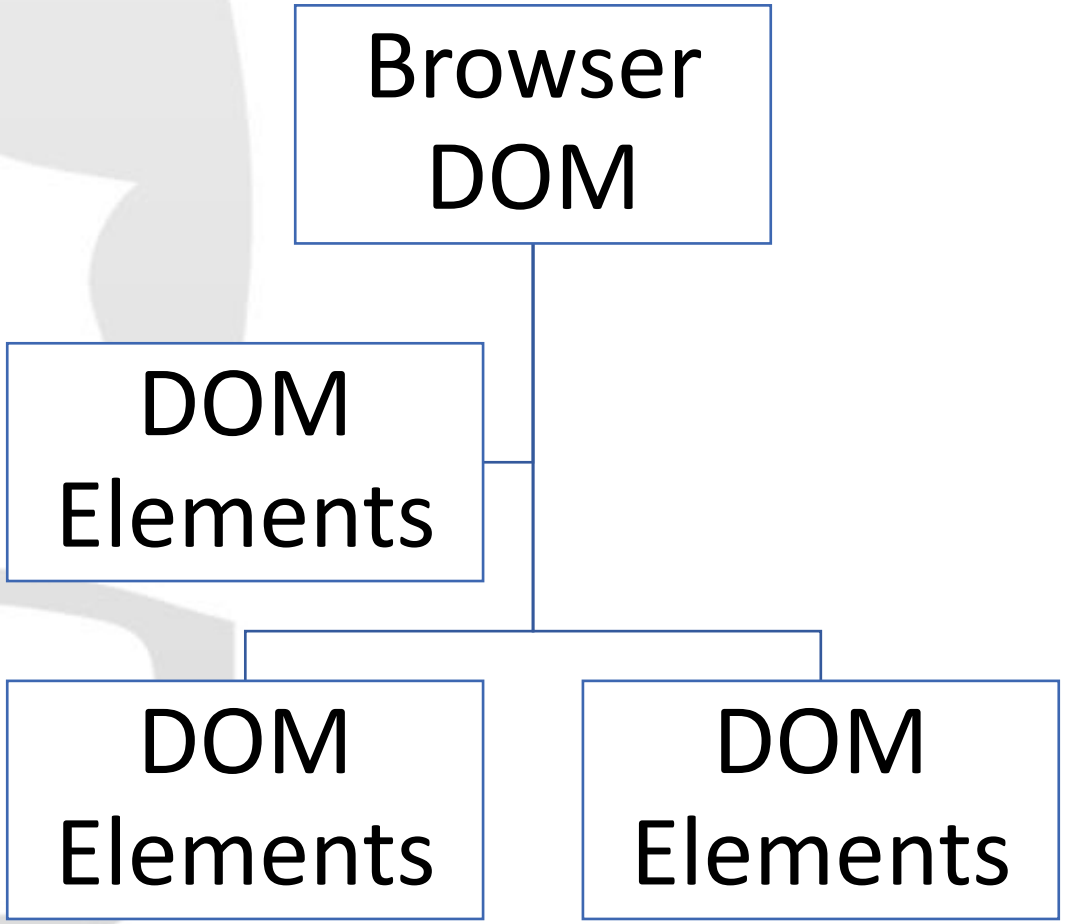
create-react-app →

What we get?	
React	The React lib!
Webpack	Links together JS files
Babel	Turns ES2015/6/7 and JSX into ES5 code
Jest	Automated test runner

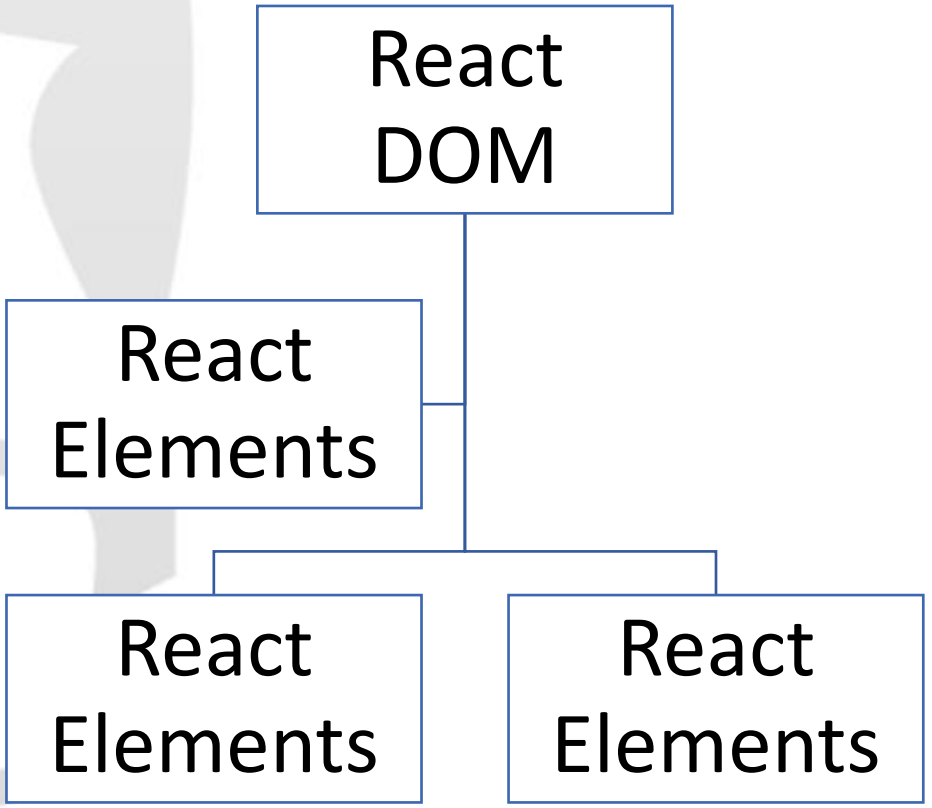
Project Structure

- Ui Components
- Main Components – Pages
- Utils
- Configurations
- State management – redux

DOM Element



React Element



React Element

- The most atomic Unit in react
- React elements are the instructions for how the browser DOM should be created.
- HTML Tags are become DOM elements when the browser loads the HTML
- React.CreateElement – working with the DOM API
- Instructions for what we want React to build
- Rendering and reconciling the elements

React Element

```
<h1 id=”h1-app">Application Header</h1>
```


```
{
  $$typeof: Symbol(React.element),
  "type": "h1",
  "key": null,
  "ref": null,
  "props": {id: ”h1-app", children: "Application Heade "},
  "_owner": null,
  "_store": {}
 }
```

React Children

- React renders child elements using props.children
- Element tree

```
<ul>
  <li>2 lb salmon</li>
  <li>5 sprigs fresh rosemary</li>
  <li>2 tablespoons olive oil</li>
  <li>2 small lemons</li>
  <li>1 teaspoon kosher salt</li>
  <li>4 cloves of chopped garlic</li>
</ul>
```

React Children

```
React.createElement(
  "ul",
  null,
  React.createElement("li", null, "2 lb salmon"),
  React.createElement("li", null, "5 sprigs fresh rosemary"),
  React.createElement("li", null, "2 tablespoons olive oil"),
  React.createElement("li", null, "2 small lemons"),
  React.createElement("li", null, "1 teaspoon kosher salt"),
  React.createElement("li", null, "4 cloves of chopped garlic")
);
```

ClassName in react

- HTML Elements use class attribute
- Class is reserved word
- React use className
- Style object is also available

ClassName in react

```
React.createElement(
 "ul",
 { className: ” ingredients " },
  ingredients.map(item => React.createElement("li", null,  item))
);
```

```
const items = [
  "2 lb salmon",
  "5 sprigs fresh rosemary",
  "2 tablespoons olive oil",
  "2 small lemons",
  "1 teaspoon kosher salt",
  "4 cloves of chopped garlic"
];
```

Problem

Writing all our application with Create Element

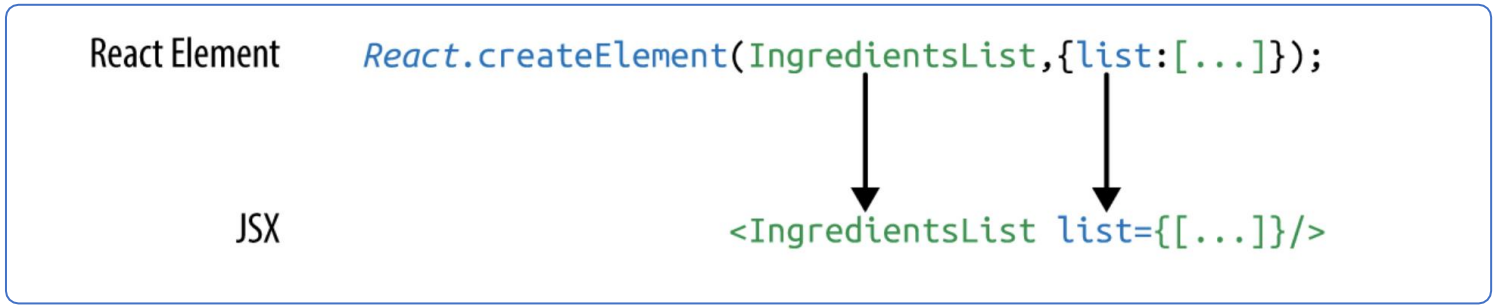
is time consuming

Solution

- JSX
- on the beach!

JSX

- concise syntax
- Easy to create DOM trees with attributes
- More readable like HTML & XML
- Using JS expressions ..{}



Mapping Arrays with JSX


```
<ul>
  {props.ingredients.map((ingredient, i) => (
    <li key={i}>{ingredient}</li>
  ))}
</ul>
```

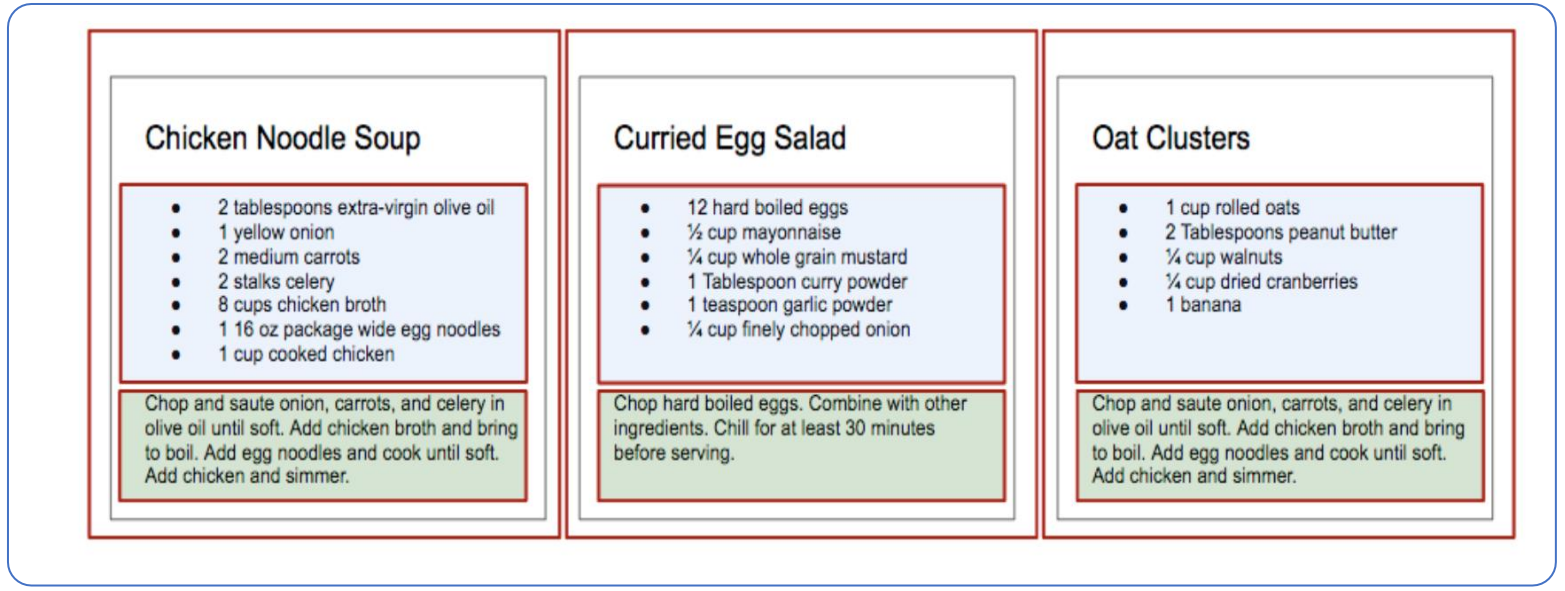
Babel

Convert our fancy source code into something

that the browser can interpret

Components

- User interface is made up of parts
- When we put them all together we are getting user interface
- Independent
- reusable pieces



Components

- Class
- Function

Function Component

```tsx
import React from "react";
interface Props {
  title: string;
  color: string;
}
function Header(props: Props) {
  const { title, color } = props;
  return (
    <h1 className="jumbotron" style={{ color }}>
      {" "}
      {title}{" "}
    </h1>
  );
}
```

Class Component

```
export class Header extends React.Component<Props, any> {
 constructor(props: Props) {
  super(props);
 }

 render() {
  const { title, color } = this.props;
  return (
   <h1 className="jumbotron" style={{ color }}>
    {title}
   </h1>
  );
 }
}
```

Props

- Object
- Props are stands for properties which passed to the components
- Props are read only
- Props can contain primitive and complex data

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

React Fragments

- Container
- Avoid DIV Hell

Stateless Component

Component is just a plain JavaScript function

which takes props as an argument and

returns a react element

Stateful Component

Component that contains State object

and reflect their state with the render process

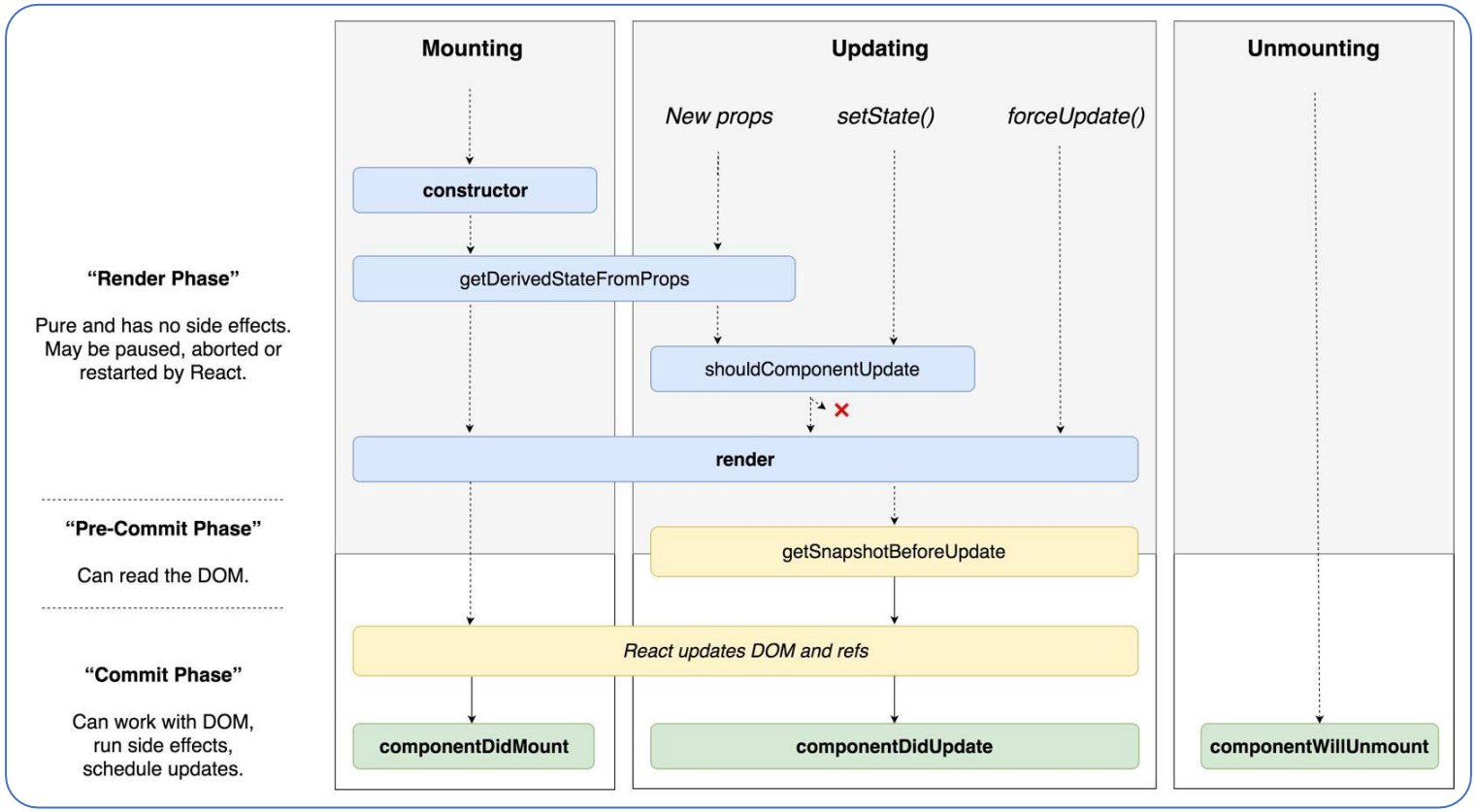
The State

- The *state* of a React application is driven by data that has the ability to change
- Reflect the content - UI

Using the state

- Don’t change the state directly
- Use setState mothod
- React may batch multiple setState() calls into a single update for performance
- State updates are merged

Components lifecycle methods



mounting

- These methods are called in the following order when an instance of a component is being created and inserted into the DOM:
- constructor()
- static getDerivedStateFromProps()
- render()
- componentDidMount()

getDerivedStateFromProps()

- Before render
- Return object that updates the state
- When state depends on props – AntiPattern
- Fetch data on props changed

render()

- React Element tree is created
- React updates his virtual DOM

ComponentDidMount()

- Invoked immediately after a component is mounted - inserted into the tree
- Call remote endpoint
- Subscriptions

Updating

- An update can be caused by changes to props or state. These methods are called in the following order when a component is being re-rendered:
- static getDerivedStateFromProps()
- shouldComponentUpdate()
- render()
- getSnapshotBeforeUpdate()
- componentDidUpdate()

shouldComponentUpdate()

- Invoked before rendering when new props or state are being received.
- Help us to decide if the state changes need to affect the component render
- Return Boolean value
- Default true

getSnapshotBeforeUpdate()

- Before the last render committed to the DOM
- Capture information from the DOM
- passed as a parameter to componentDidUpdate()

componentDidUpdate()

- Invoked immediately after updating occurs
- Not after initial render
- SetState ? Be carful from infinite loop

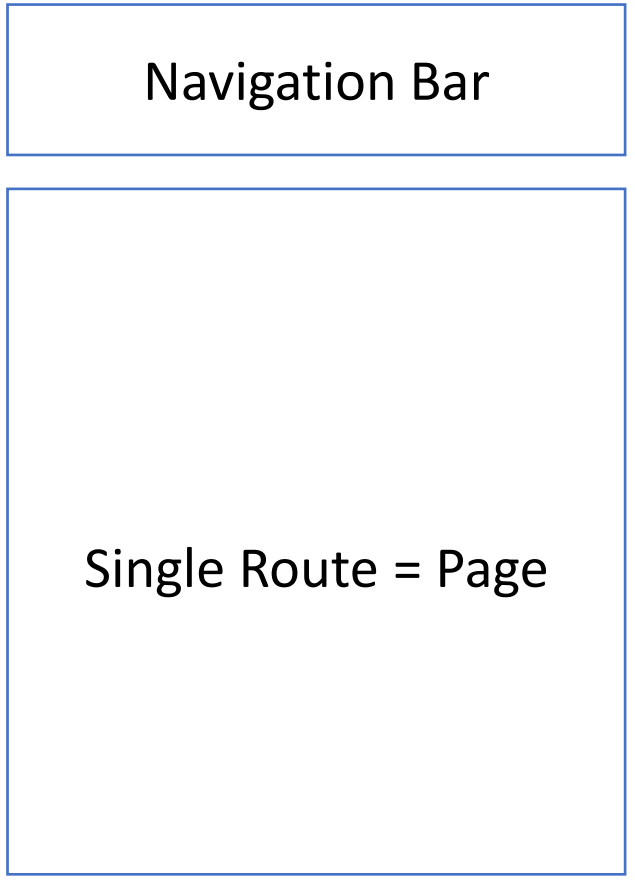
Unmount

- This method is called when a component is being removed from the DOM:
- ComponentWillUnmount()

Routing

```
<Link to="/home">Home</Link>


<BrowserRouter>
<Switch>
<Route path="/signIn" component={SignIn} />
<Route path="/signup" component={SignUp} />
<Route path="/home" component={Home} />
<Route path="**" component={() => <h1> Not Found! </h1>} />
</Switch>
</BrowserRouter>
```



Routing

npm I react-router-dom

Virtual DOM

- Real DOM – Document Object Model
- Tree Data structure – Fast
- UI re-rendering or repainting is heavy
- Virtual DOM – virtual representation, saved in memory
- Every time that our state changed the virtual DOM is updated

Virtual DOM

- State changed => New virtual tree created
- The tree is compared or “diffed” with the previous
- Calculates the best possible way to apply the changes

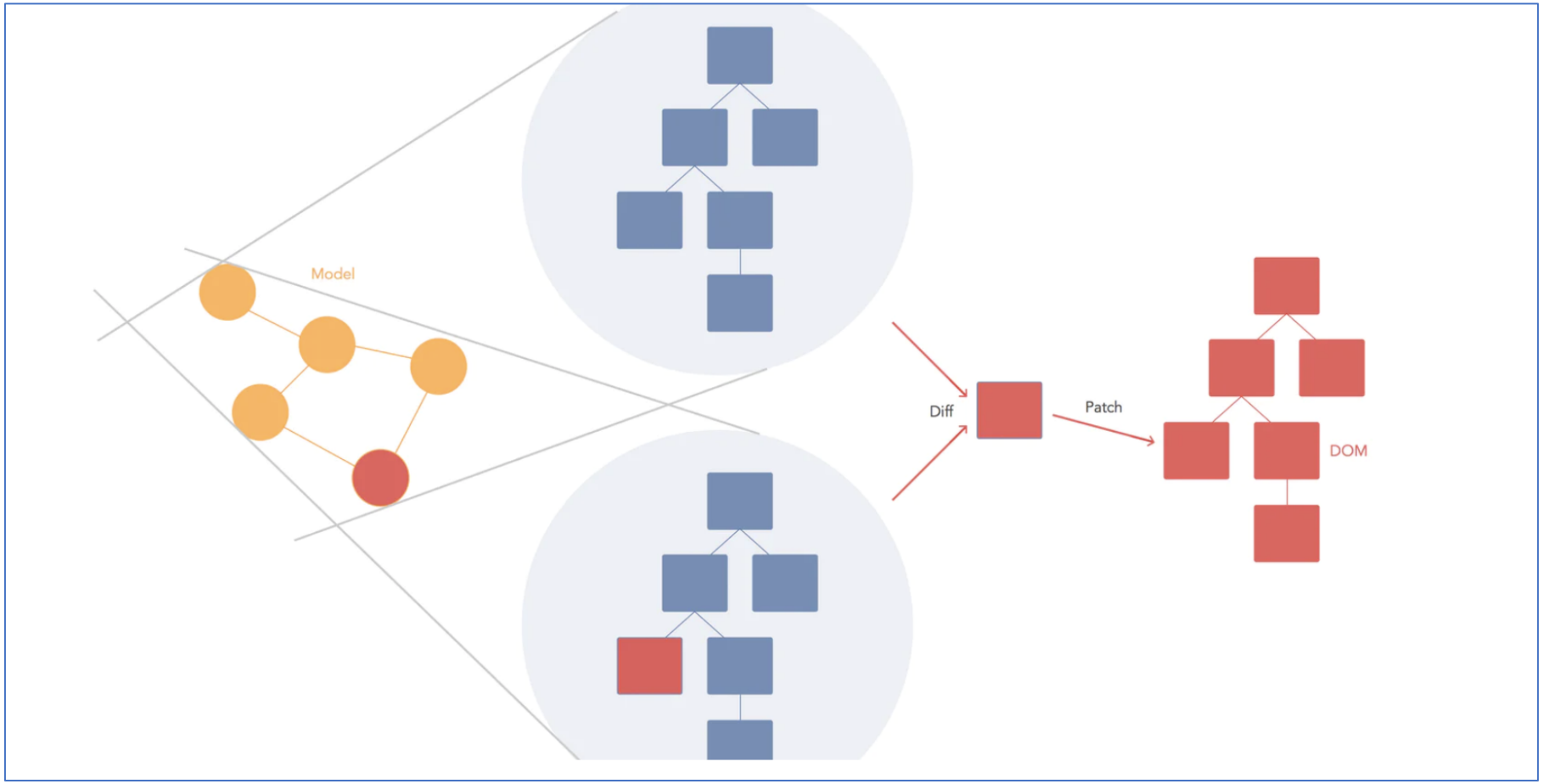
How does React use Virtual DOM

- React use observable pattern
- Listen to state changes
- New version of virtual DOM diffing with the old
- Once react know what object are changed
- It updated only the changed object on the real DOM

Virtual DOM – batch update

- React follow a batch update mechanism to update the real DOM
- No manipulating the DOM every single change

Virtual DOM



Reconciliation

Two elements of different types will produce different trees.

The developer can hint at which child elements may be stable across different renders with a key prop.

Reconciliation – The Diffing Algorithm

- Root element type – different ?
- DOM element same type, check the attribute
- Children's compare
- Using the Key

Reconciliation – Children's

- React iterates both lists and generate mutation over the diff

```
<ul>
  <li>first</li>
  <li>second</li>
</ul>

<ul>
  <li>first</li>
  <li>second</li>
  <li>third</li>
</ul>
```

Reconciliation – Children's

- React will Mutate every child

```
<ul>
  <li>Duke</li>
  <li>Villanova</li>
</ul>

<ul>
  <li>Connecticut</li>
  <li>Duke</li>
  <li>Villanova</li>
</ul>
```

```
<ul>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>

<ul>
  <li key="2014">Connecticut</li>
  <li key="2015">Duke</li>
  <li key="2016">Villanova</li>
</ul>
```

HOC

- A higher-order component is a function that takes a component and returns a new component.
- Reuse Logic

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

HOC

- Take a Component as argument
- Return Component

What sucks ?

- It’s hard to reuse stateful logic between components
- Complex components become hard to understand
- Classes confuse both people and machines

Hooks

- React Hooks are functions that let us hook into the React state and lifecycle features from function components

Rules of Hooks

Only call Hooks **at the top level**. Don’t call Hooks inside loops, conditions, or nested functions.

Only call Hooks **from React function components**.

useState

- Managing state inside function component

useEffect

- For side effects – event handlers, async request, etc…

Authentication HOC

- Sync - check if token exist
- Async – Validate the token
- Return the relevant component

useEffect

- For side effects – event handlers, async request, etc…

useRef

- Access the DOM directly
- Mutating ref

Controller components vs uncontrolled

feature	uncontrolled	controlled
one-time value retrieval (e.g. on submit)	✅	✅
validating on submit	✅	✅
instant field validation	❌	✅
conditionally disabling submit button	❌	✅
enforcing input format	❌	✅
several inputs for one piece of data	❌	✅
dynamic inputs	❌	✅

useCallback & useMemo

- The React useCallback Hook returns a memoized callback function.
- The React useMemo Hook returns a memoized value.

Code splitting

- your app grows, your bundle will grow too
- “splitting” your bundle
- Webpack

Suspense

- Expiramental
- Suspense is not a data fetching library. It’s a mechanism for data fetching libraries
- We should make it as easy as possible to build apps that start fast and stay fast
- Suspense is a feature for managing asynchronous operations in a React app. It lets your components communicate to React that they’re waiting for some data.

Suspense

- Parallel data and view trees
- Fetch in event handlers
- Load data incrementally
- Treat code like data

React.memo


When to use
React.memo()
Heuristics whether a React component should be wrapped in React.memo()
01
Pure functional component
Your <Component> is functional and given the same props, always renders the same output.
02
Renders often
Your <Component> renders often.
03
Re-renders with the same props
Your <Component> is usually provided with the same props during re-rendering.
04
Medium to big size
Your <Component> contains a decent amount of UI elements to reason props equality check.

React.memo

Problem – Global State Management

- Inputs
- Forms
- Routing
- Date pickers
- Menus
- Tabs
- Filters

Problem – Global State Management

- Data

Redux

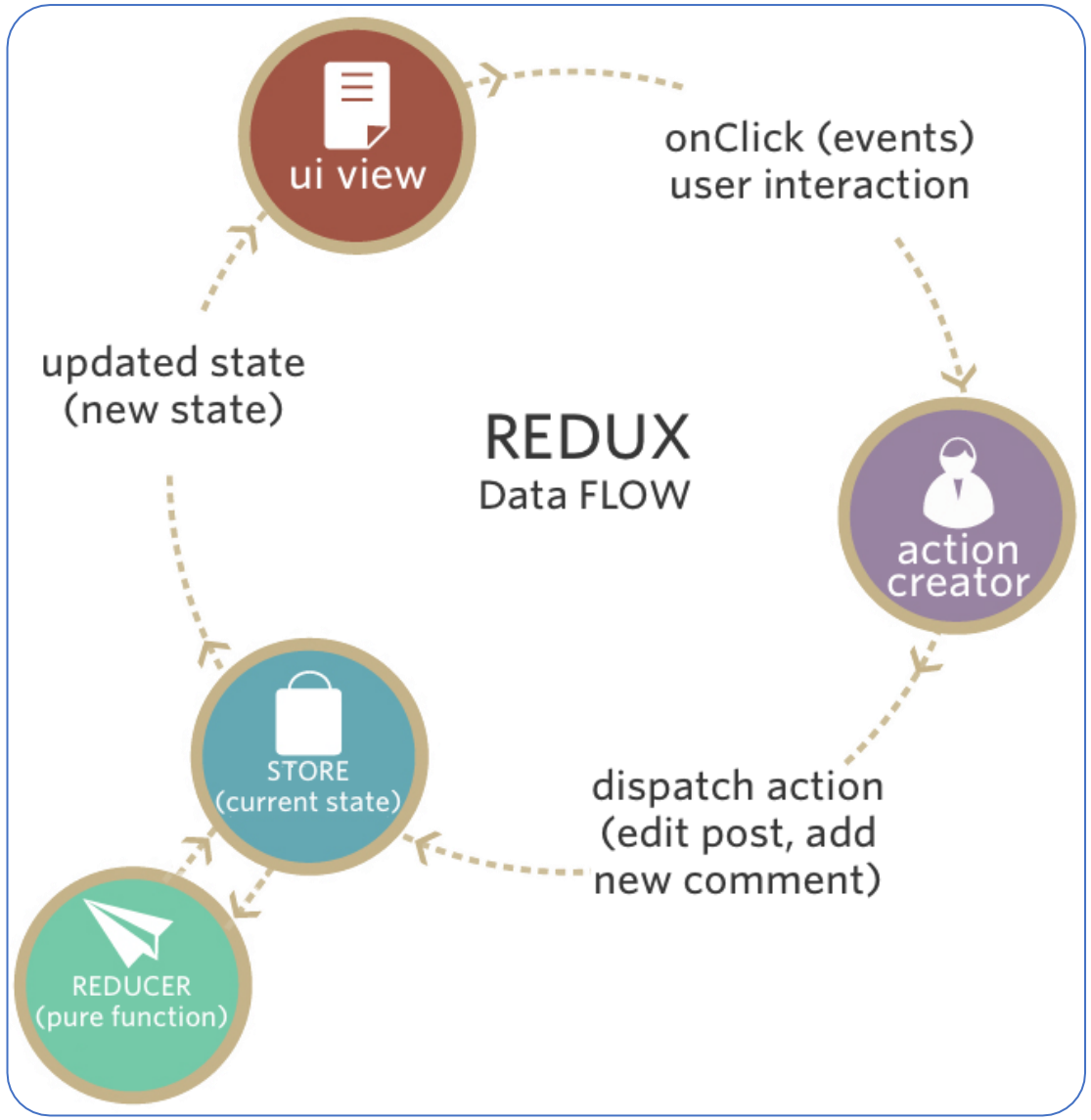
- The application has a central / root state.
- A state change triggers View updates.
- Only special functions can change the state.
- A user interaction triggers these special, state changing functions.
- Only one change takes place at a time.

Redux

- Store
- GetState
- Dispatch
- Subscribe
- Action
- Reducers

Redux toolkit

- Configure store
- Create reducer
- Create slice
- Create async thunk


ui view
onClick (events)
user interaction
updated state
(new state)
REDUX
Data FLOW
action
creator
STORE
(current state)
dispatch action
(edit post, add
new comment)
REDUCER
(pure function)

Redux

More Options

- MobX
- useContext

React Testing

THE FOUR TYPES OF TESTS

End to End

A helper robot that behaves like a user to click around the app and verify that it functions correctly.

Sometimes called “functional testing” or e2e.

Integration

Verify that several units work together in harmony.

Unit

Verify that individual, isolated parts work as expected.

Static

Catch typos and type errors as you write the code.

Testing tradeoffs

- Cost
- Speed
- Complexity

Distinctions

I don't really care about the distinctions.

Testing options

- Rendering component trees
- Running a complete app

Testing tradeoffs

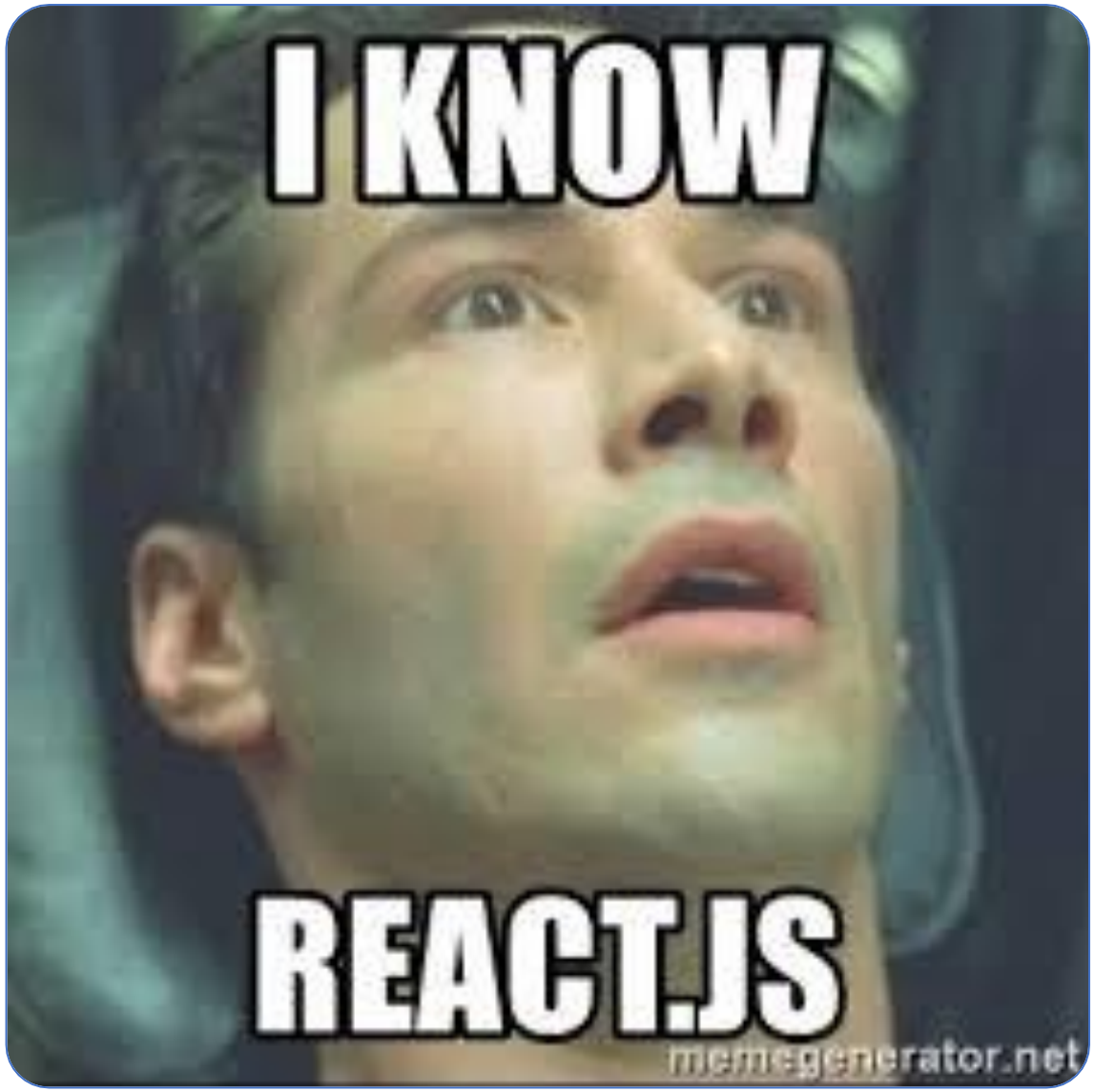
- Iteration speed vs Realistic environment
- How much to mock

Tools

- Jest (using jsdom)
- React Testing library

Style Components

- Using tagged template literals and arrow functions in ES6+ and CSS, styled-components is a React-specific CSS-in-JS styling solution that creates a platform for developers to write actual CSS code to style React components.



THANKS!

galamouyal88@gmail.com

DevGeekWeek 2024
INSPIRED BY INNOVATION

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

תודה :)

נתראה בשנה הבאה

matrix

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company