Introduction to

REACT JS

ES6

History of JavaScript

- 1997 – ECMAScript 1
- 1998 – ECMAScript 2
- 1999 – ECMAScript 3
- 2009 – ECMAScript 5
- 2011 – ECMAScript 5.1
- 2015 – ECMAScript 2015 (ES6)
- 2016 – ECMAScript 2016 (ES7)
- 2017 – ECMAScript 2017 (ES8)
- 2018 – ECMAScript 2018 (ES9)

What is ES6??

JavaScript has had many versions over the years and ES6 is one of the JavaScript versions.

ES6 was released in 2015 and ES6 refers to version 6 of the ECMAScript programming language. ECMAScript is the standardized name for JavaScript. It is a major enhancement to the JavaScript language, and adds many more features intended to make large-scale software development easier.

ES6 Features

Here are some of the features of ES6 which are heavily used in React Development:

- Variable creation using "let" and const"
- Template Strings(also known as Template Literals).
- Rest and Spread Operator
- Arrow Functions
- Array Functions: map(), reduce(), filter(), find() and findIndex()
- Destructuring
- Classes, Properties & Methods
- Promises

and much more.

Let and Const

In old JavaScript, we use “*var*” to create variables. In ES6 we use “*let*” and “*const*”.

Think of *let* as the new *var*. It is used to create a variable for which the value may change even after it’s declared. **It doesn’t supports hoisting.**

const is used to create a variable for which the value will never change after it’s declared. JavaScript will give you an error if you try to update the value of a const variable. It doesn’t supports hoisting.

For eg,

var mName = “John Lark”

let mAge = 28

const endpoint = “https://api.rest.com/v1/”

Arrow Functions

ES6 gives us a new syntax for defining functions using a fat arrow(=>). Arrow functions bring a lot of clarity & code reduction. It **does not support hoisting.**

Old Method:

```
function mFunc(arg1, arg2...) {
    // Function Body
}
```

Arrow Functions:

```
const mFunc = (arg1, arg2) => {
    // Function Body
}
```

Arrow Functions (Variations)

const mFunc = arg1 => { // Function Body } **// If there is a single argument then you can get rid of parenthesis.**

const mFunc = arg1 => { return arg1*2 }

const mFunc = arg1 => arg1*2 **// If there is a single line of code then you can get rid of curly braces and the return statement.**

Template Strings(Template Literals)

It is just a string which allows embedding expressions inside it. To create a template string you don't use single or double quotes, you use back tick. and the expression is wrapper inside ${}.

For example,

```
const mGreetings = `Hello ${firstName + lastName}`

const mFullName = `My full name is: ${getFullName()}`
```

Primitive and Reference Types

Primitive Types just hold a value whereas, reference types hold a pointer/reference to the stored location in the memory.

Primitive Types - Numbers, String and Boolean.

Reference Types - Arrays and Objects.

When you copy a variable in another like below example, for primitive types the value is copied but for reference types the pointer is copied.

const mVar1;

const mVar2 = mvar1;

Rest and Spread

Rest and Spread is represented by the same operator which is ‘...’ . This operator is called Rest and Spread based on how it is being used.

Spread is used to split array elements and object properties.

Rest is used to merge all the function arguments in an array. This helps to create a function which accepts variable argument.

Destructuring

Destructuring helps to extract array elements and object properties. These extracted values are stored in variables.

For eg,

var [a, b] = [‘Alpha’, ‘Beta’, ‘Gamma’]

var {topSpeed, name} = {name: ‘Koenigsegg Agera’, manufacturer: ‘Koenigsegg’, topSpeed: ‘457kmph’}

Array Functions

There are many array functions but here are some which are used very often during React development - map(), reduce() and filter().

map() - Used to iterate list items. Accepts a callback function which runs on all the items. It returns an array with same length as the initial array.

reduce() - Used to iterate list or object items. Accepts two parameter an accumulator and a callback function. But instead of returning the result to array, it passes it to the next item.

filter() - Used to filter out data from a list. It returns an array with the filtered data.

Array Functions: Syntax

```
const resultingArr = arr.map((item, pos) => {
    return resultingArr;
})


const resultingArr = arr.reduce((accumulator, item) => {
  return accumulator + item.property;
}, 0);


const resultingArr = arr.filter(item => {
  return true/false;
});
```

Callbacks

In JavaScript the code is executed line-by-line in a sequence so when we run a parallel operation or asynchronous operation like fetching data from backend, JavaScript doesn't waits for the response it simply executes the next line of code. So, we give the asynchronous operation a function to call when it is completed. This function is called a callback function.

For example,

```
$.get('some-url', callbackFunc);
```

Nested Callbacks

The get() method provided by jquery is an asynchronous function. It runs in parallel to the main program flow. It sends a call to the backend and whenever the response is received, it runs the callback function. And it works perfectly fine.

But now let's say, the call returns a list of IDs and you need to send another call to the backend to get details for a specific ID.

How would you do it??

You cannot write it after the get() method because you have no way of knowing when the response is received. So your only option is to put the details call inside the first callback function.

Callback Hell!!

I hope you are starting to notice that when you have a callback inside a callback like this, the code starts to get less readable and a more messy. This is known as **callback hell.**

Promises

A promise is used to handle the asynchronous result of an operation. What it does is, it defers the execution of a code block until an asynchronous request is completed. This way, other operations can keep running without interruption.

A Promise have three states:

Pending: This means the operation is going on.

Fulfilled: This means the operation was completed.

Rejected: The operation did not complete and an error can be thrown.

How to create Promises?

Syntax:

```
const mPromise = new Promise((resolve, reject) => {
    // Promise body
    //Call resolve() when the operation is complete.
    //Call reject() when the operation is failed.
})
```

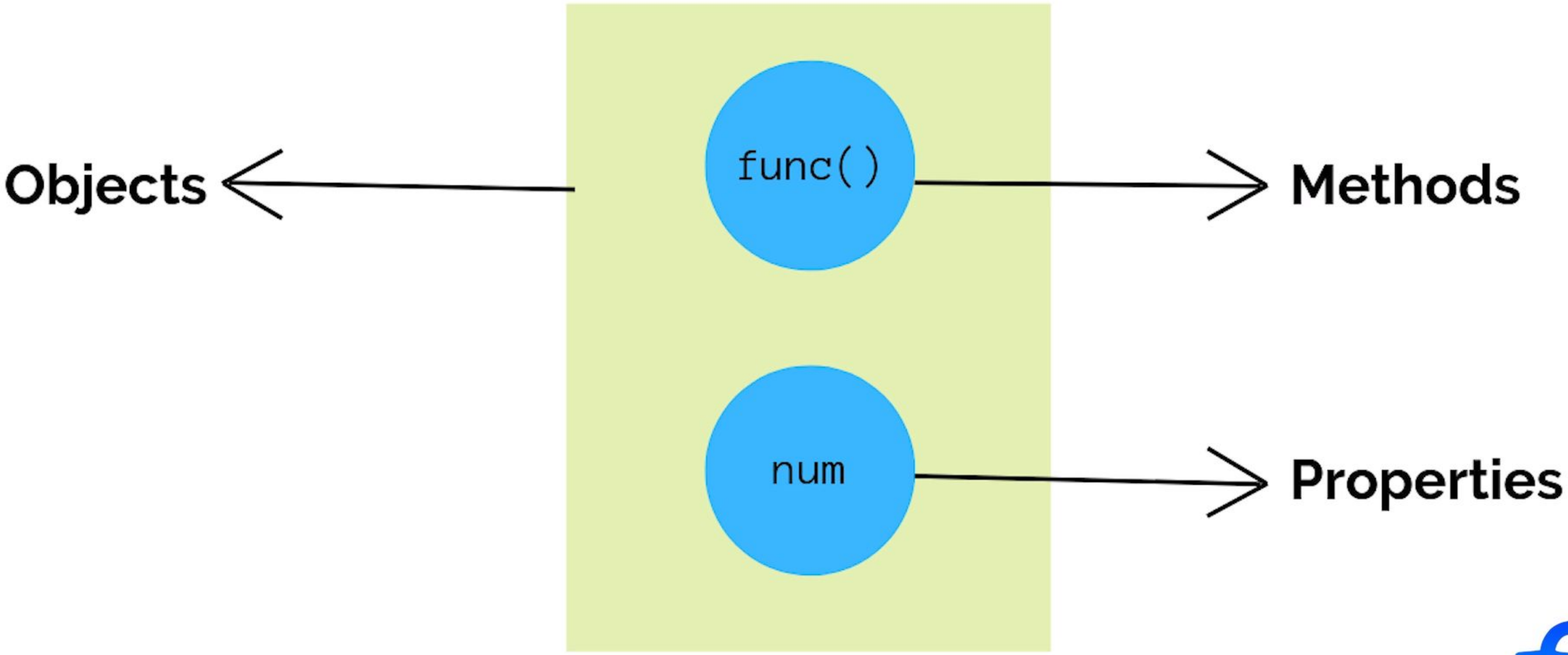
"then()" and "catch()" Methods

then() method is called when the resolve() is executed. It receives data passed in the resolve() method as arguments.

catch() method is called when the reject() method is executed. It receives the data passed in the reject() method as arguments.

OOPs

OBJECT ORIENTED PROGRAMMING



Object Oriented Programming

It is not a programming language or a library. It is a way of writing code. In Object oriented programming we combine the variables and related functions into a unit. This unit is called an Object. The variables are called as Object properties and the functions are called as object methods.

OOPs in JavaScript

Object-Oriented Programming in JS is about two things:

- Creating individual objects from a common object
- Inheritance

Classes, Properties and Methods

Classes are blueprints for creating objects. To create a new class we use the *“class”* keyword.

A class consists of properties and methods.

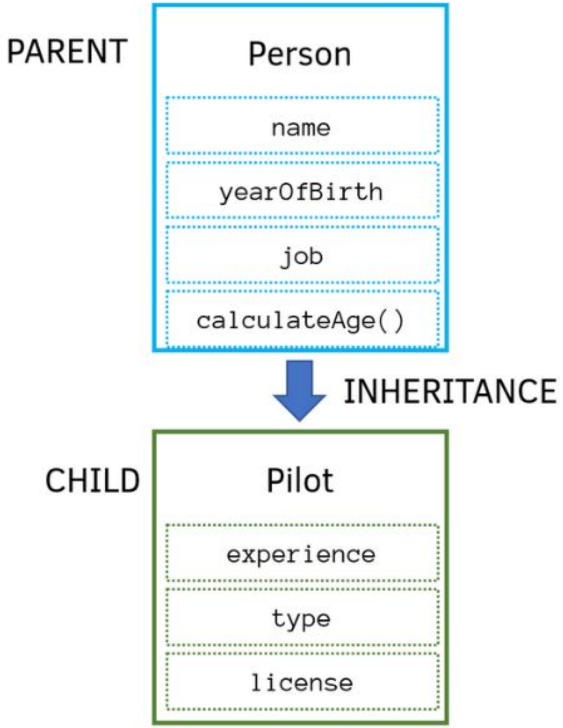
JavaScript classes also supports Inheritance. To inherit the properties and methods of another class, the *extends* keyword is used.

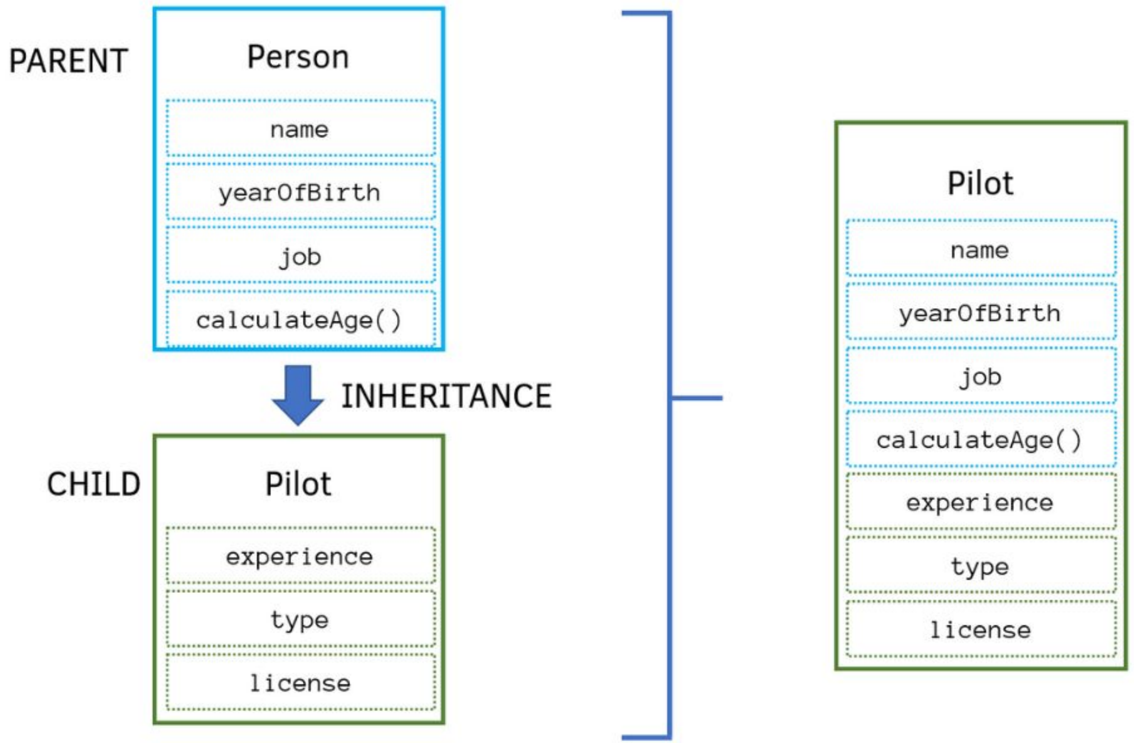
Classes, Properties and Methods

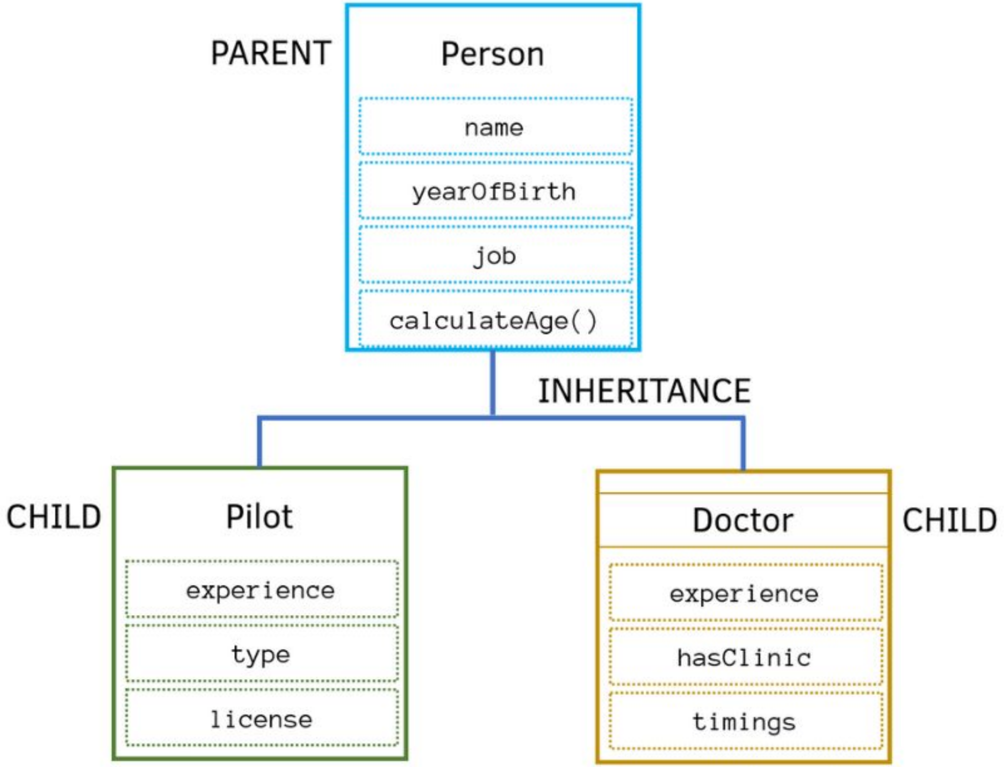
```
class Person {
  constructor(name, age, job) {
    this.name = name;
    this.age = age;
    this.job = job;
  }
  getDetails() { //Function Body }
}
```

Inheritance

Inheritance




PARENT
Person
name
yearOfBirth
job
calculateAge()
INHERITANCE
CHILD
Pilot
experience
type
license
Pilot
name
yearOfBirth
job
calculateAge()
experience
type
license


PARENT
Person
name
yearOfBirth
job
calculateAge()
INHERITANCE
CHILD
Pilot
experience
type
license
Doctor
experience
hasClinic
timings
CHILD

Constructor Functions

It a function which is used as an object factory. It can be used to create an instance of similar objects.

For example,

```
function Person(fName, lName, age) {
    this.firstName = fName;
    this.lastName = lName;
    this.age = age;
}

const jane = new Person(‘Jane’, ‘Doe’, 26);
```

INTRO TO REACTJS

What is ReactJs?

React is a JavaScript library for building SPAs with interactive user interfaces.

The entire React application can be modeled as a set of independent, isolated and reusable components. In other words you can say components are the building blocks of React App. These components are put together to design complex layouts.

React makes it painless to create interactive UIs. It has a concept called state for this purpose.

Some of the websites built with React - Facebook, Netflix, Instagram, Twitter, Airbnb, Nike, MakeMyTrip, Codepen, Snapchat, Prime Videos and many many more.

Single Page Applications

A single-page application is an app that renders inside a browser and does not require page reloading during use. For eg, Gmail, Facebook, Prime Videos, Trello and many more.

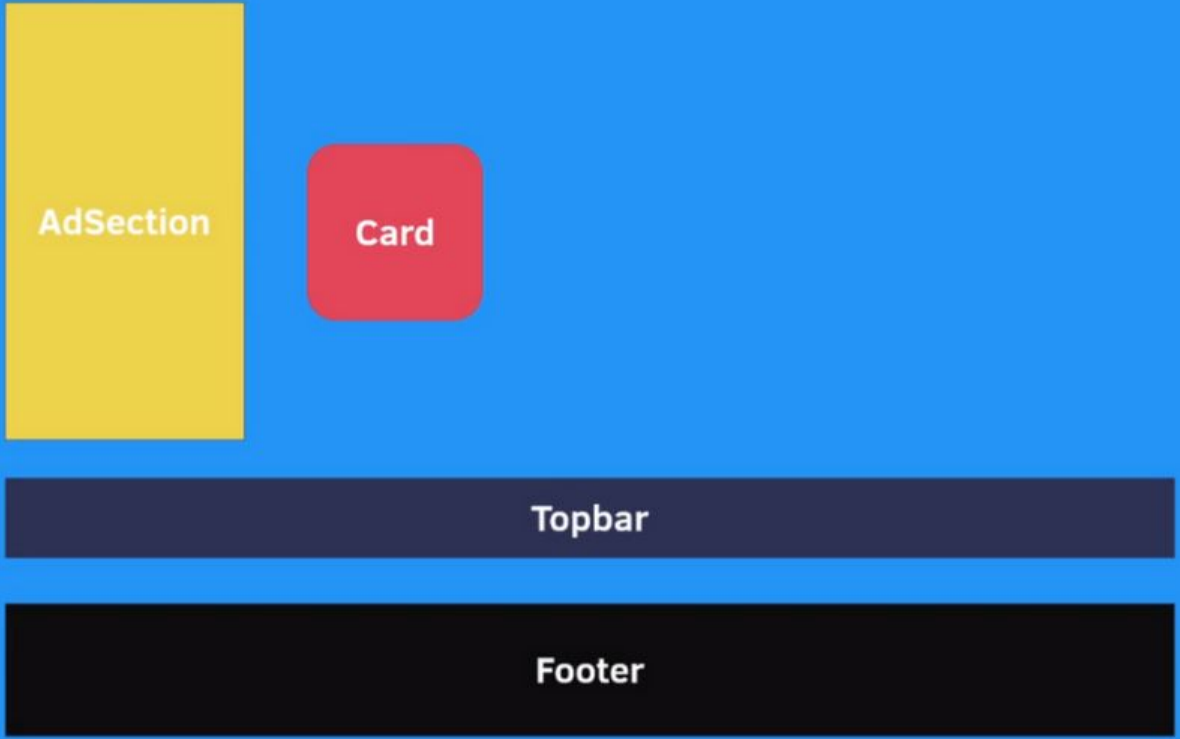
When you open a SPA in browser. It brings all the content in the first load itself. Now whenever you change routes or click on dropdowns it doesn’t needs to refresh the page to fetch data because all the required data is already present.

Some latest Frontend Frameworks like React, Angular, Vue, Ember etc enable us to create SPAs.

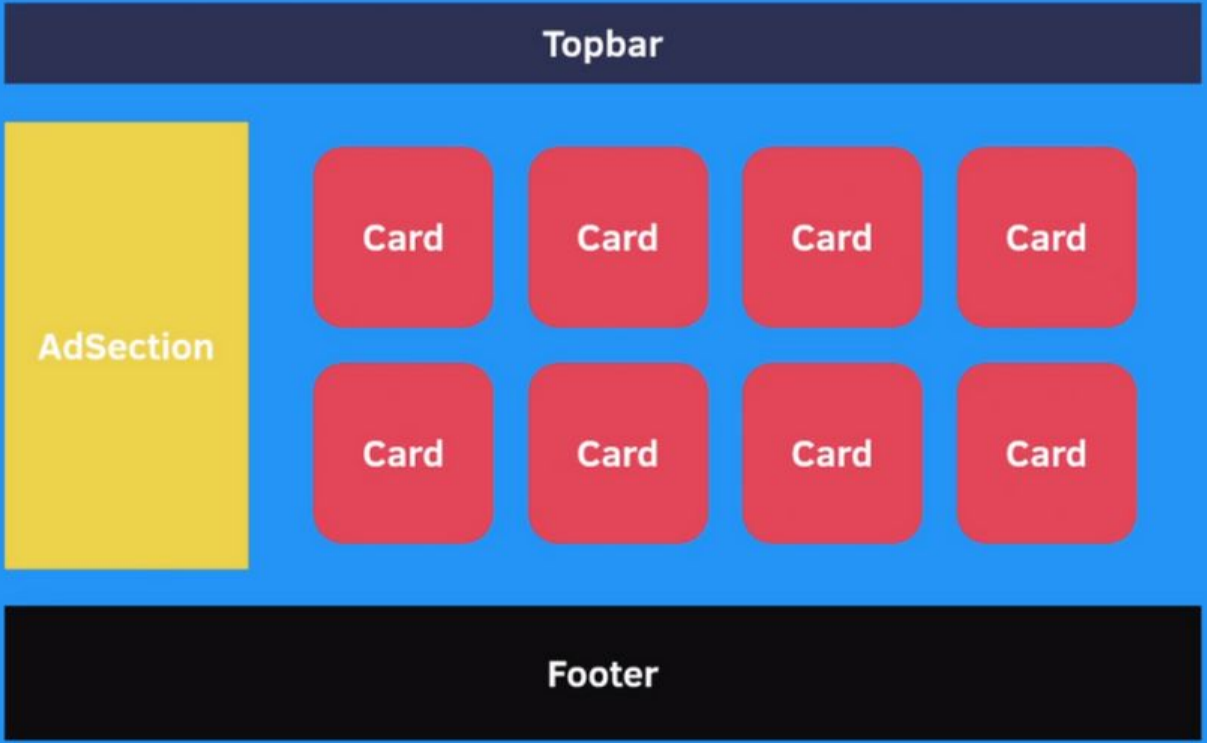
Who is using React?

Facebook Prime video Netflix Instagram Twitter

Components: The Building Blocks



Components: The Building Blocks



TRAVELLER Trips About Contact

Favorite Places

Santorini Islands, Greece

Bagan, Myanmar

Li River, China

[Santorini Islands, Greece]

[Bagan, Myanmar]

[Li River, China]

[Danxia Landforms, China]

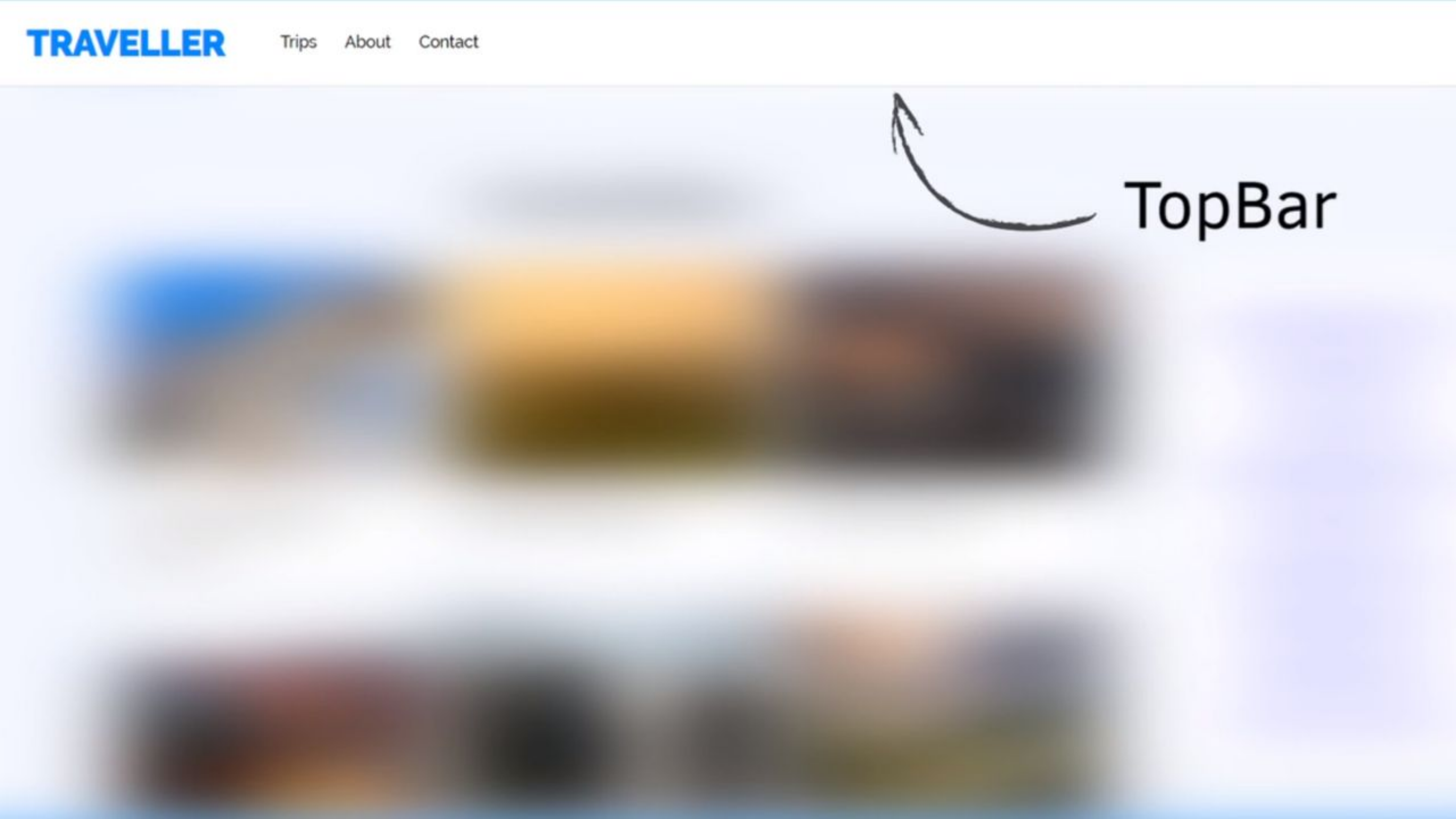
[Meteora, Greece]

[Yosemite Valley, USA]

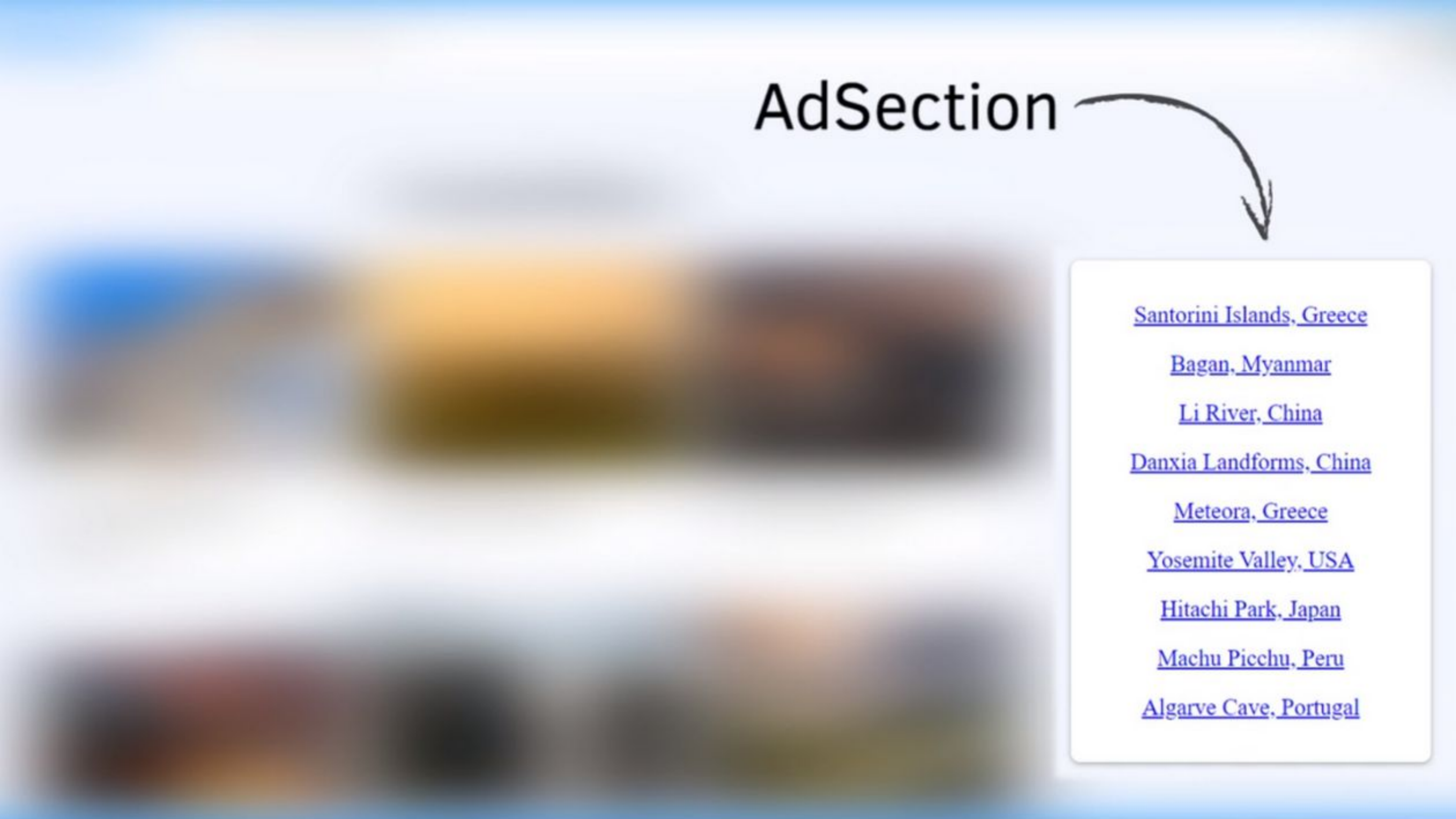
[Hitachi Park, Japan]

[Machu Picchu, Peru]

[Algarve Cave, Portugal]


TRAVELLER
Trips
About
Contact
TopBar

AdSection



Santorini Islands, Greece

Bagan, Myanmar

Li River, China

Danxia Landforms, China

Meteora, Greece

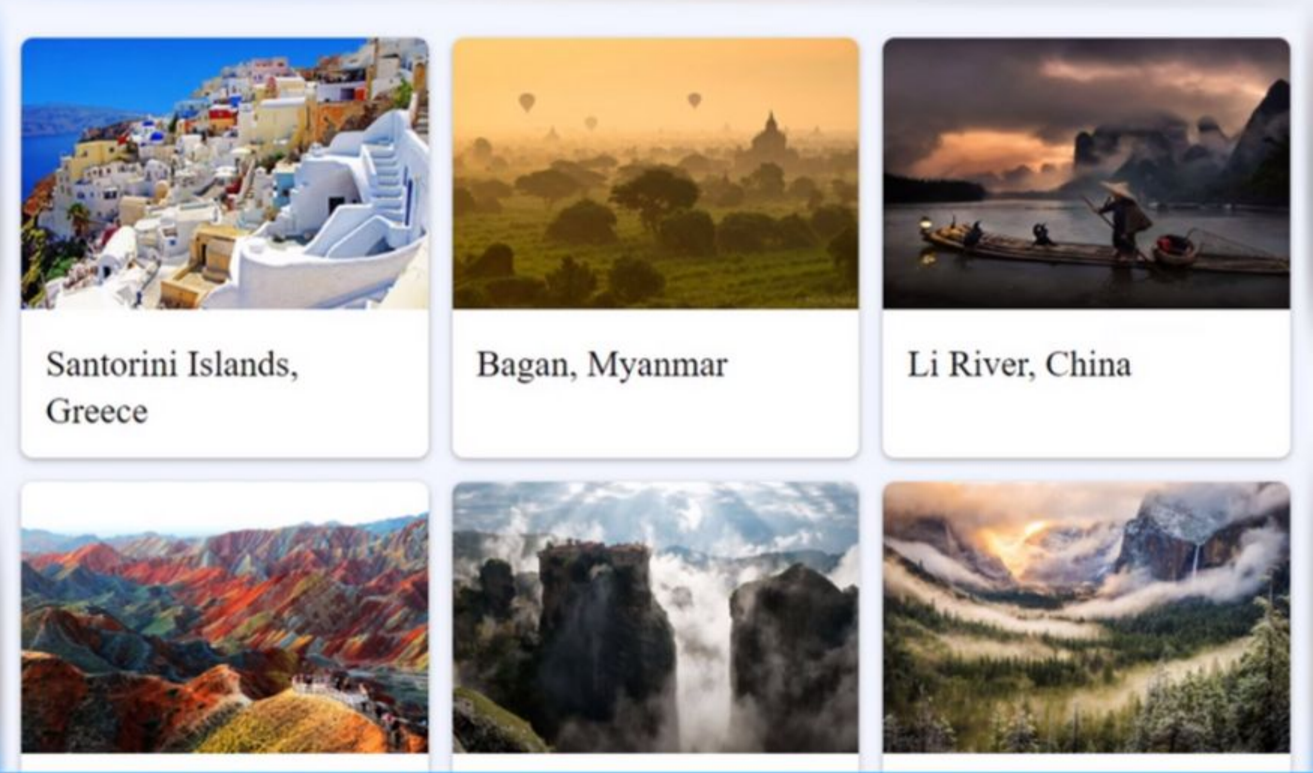
Yosemite Valley, USA

Hitachi Park, Japan

Machu Picchu, Peru

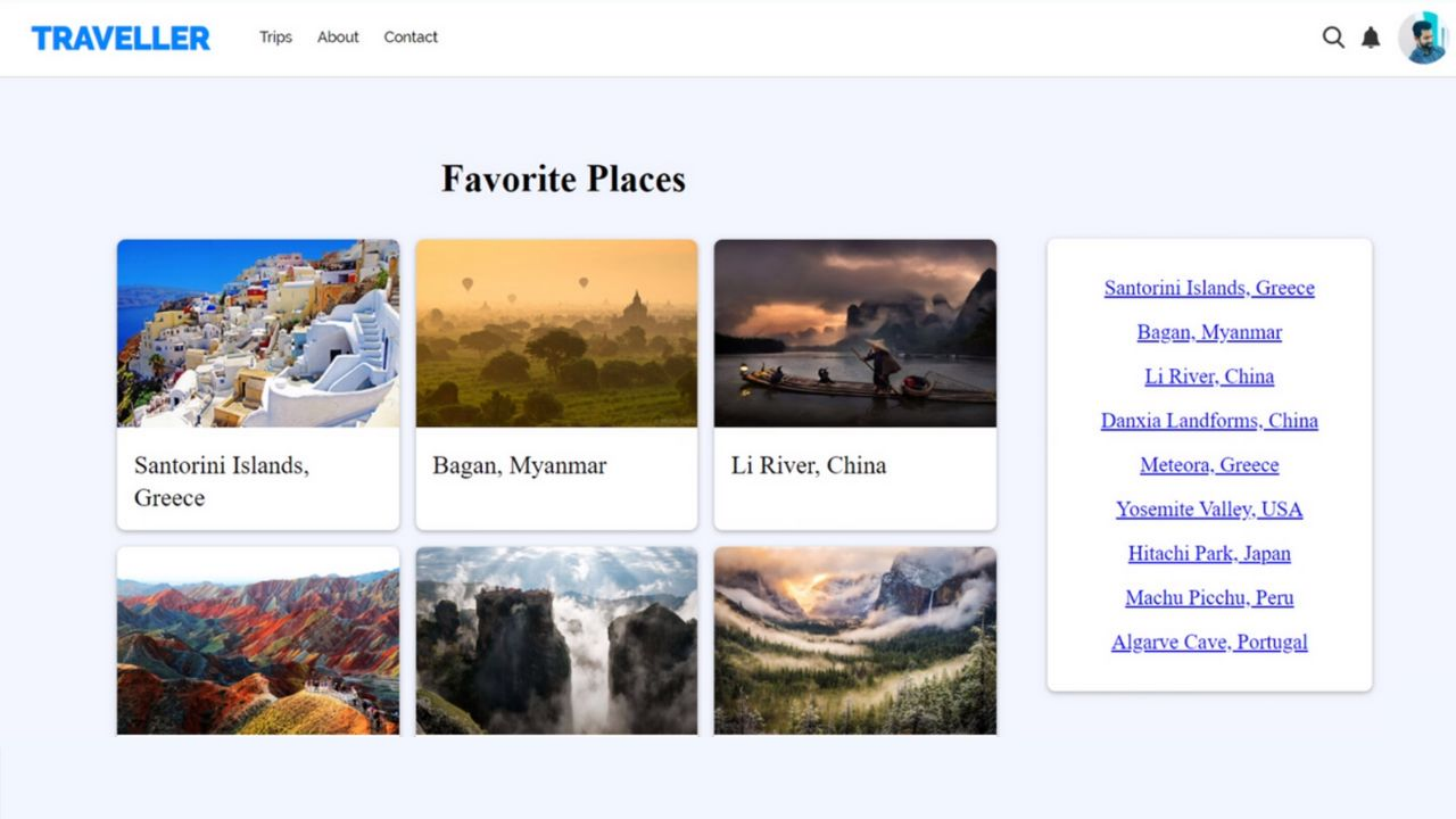
Algarve Cave, Portugal

PlacesGrid


Santorini Islands, Greece

Bagan, Myanmar

Li River, China



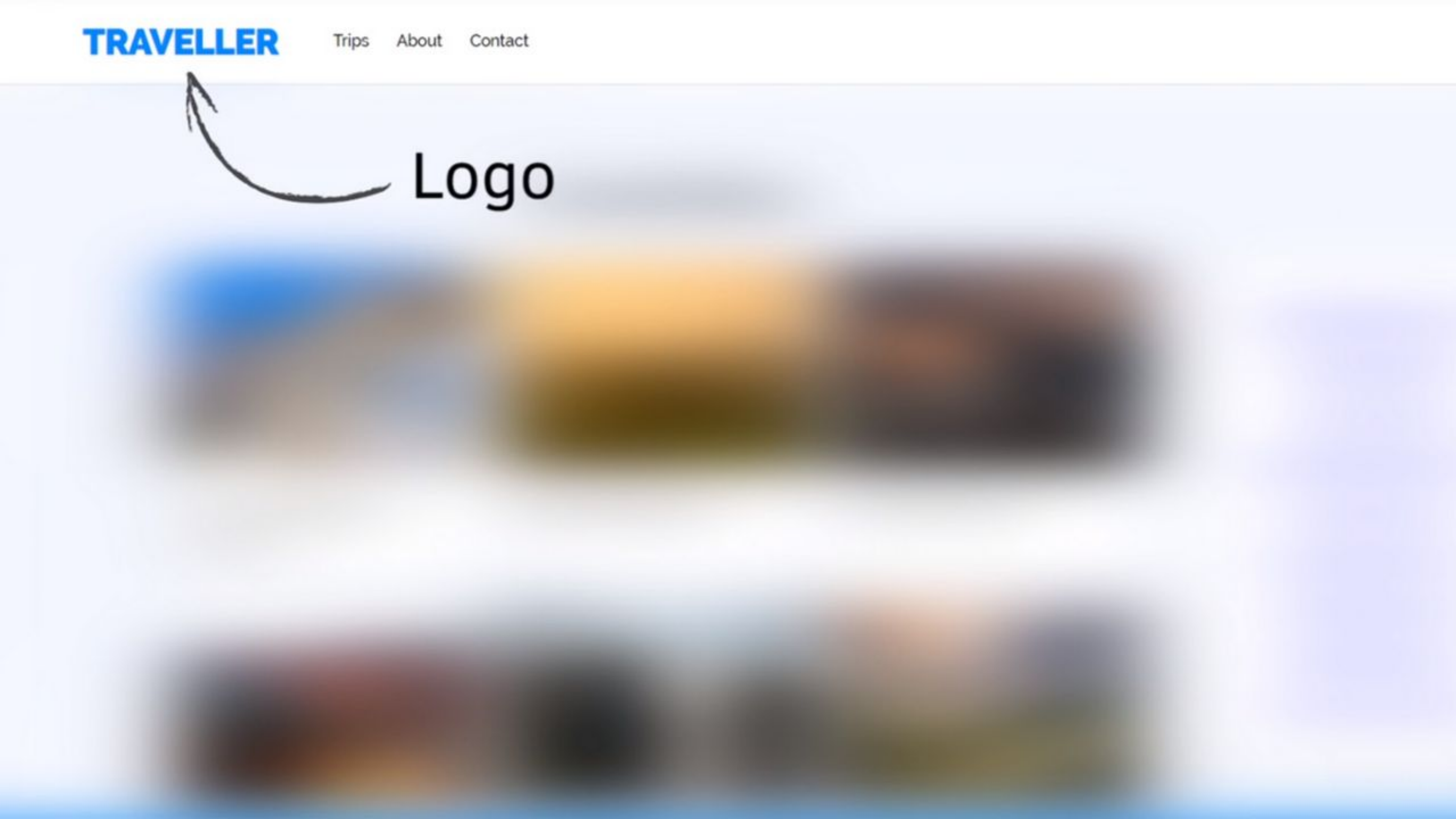
Favorite Places

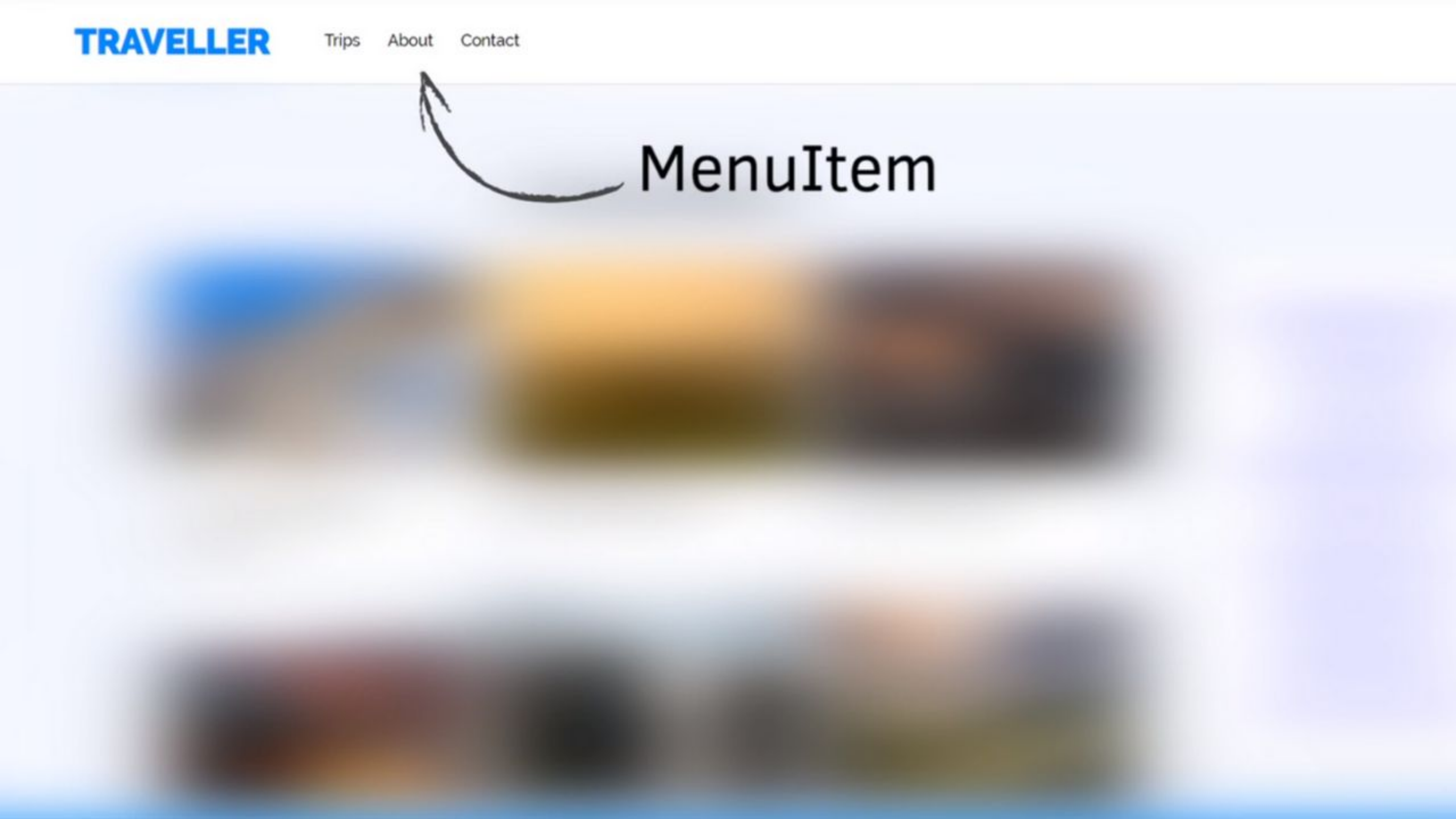
Santorini Islands, Greece

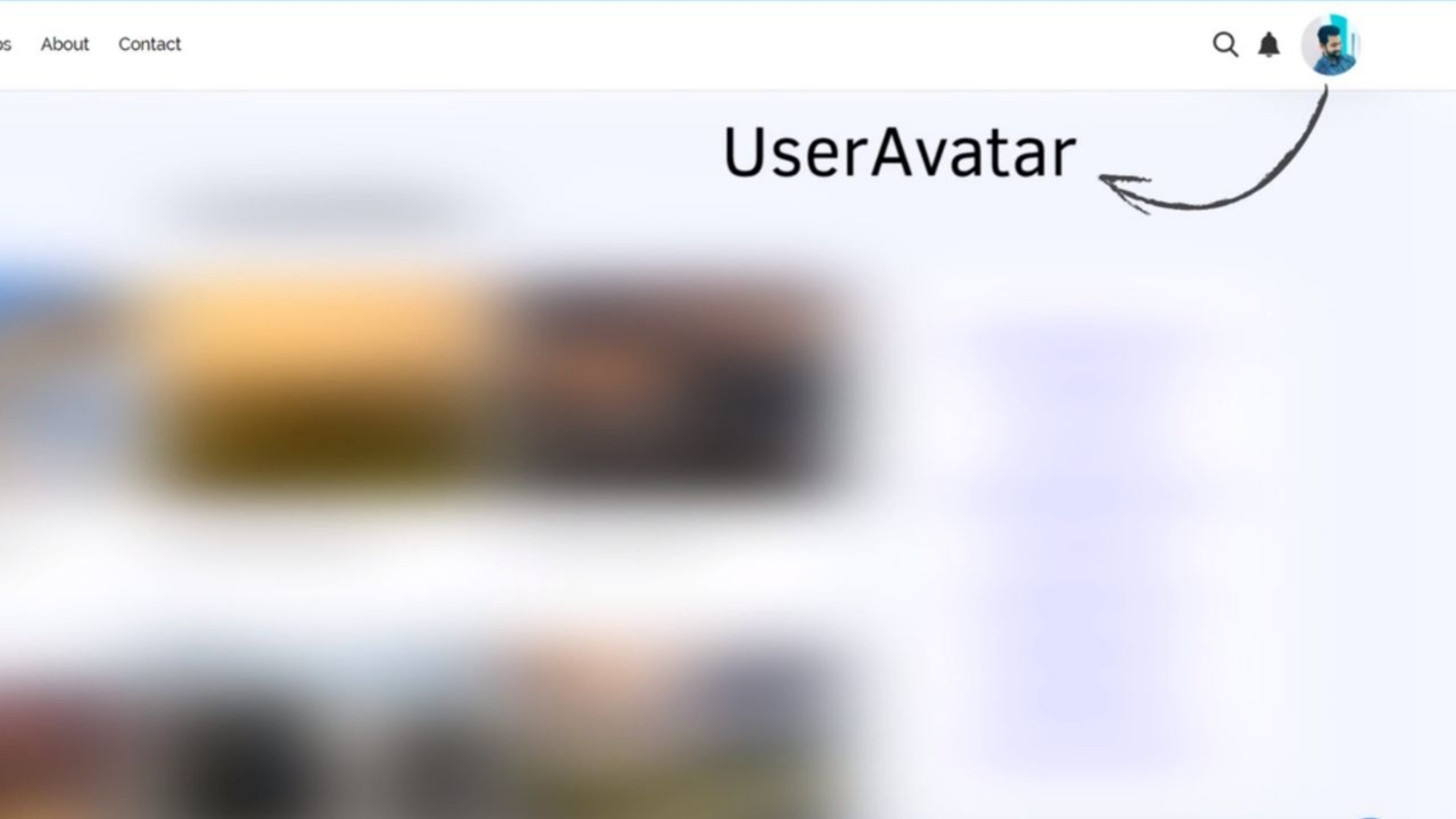
Bagan, Myanmar

Li River, China

Santorini Islands, Greece
Bagan, Myanmar
Li River, China
Danxia Landforms, China
Meteora, Greece
Yosemite Valley, USA
Hitachi Park, Japan
Machu Picchu, Peru
Algarve Cave, Portugal

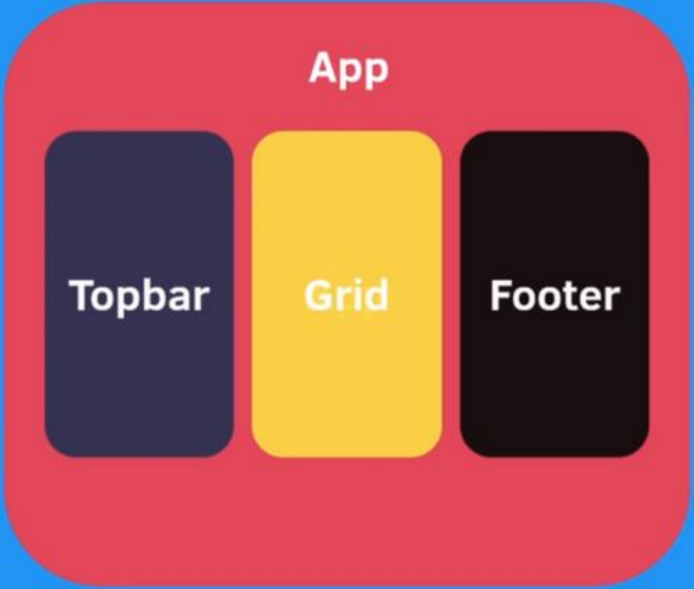

TRAVELLER
Trips
About
Contact
Logo


TRAVELLER
Trips
About
Contact
MenuItem

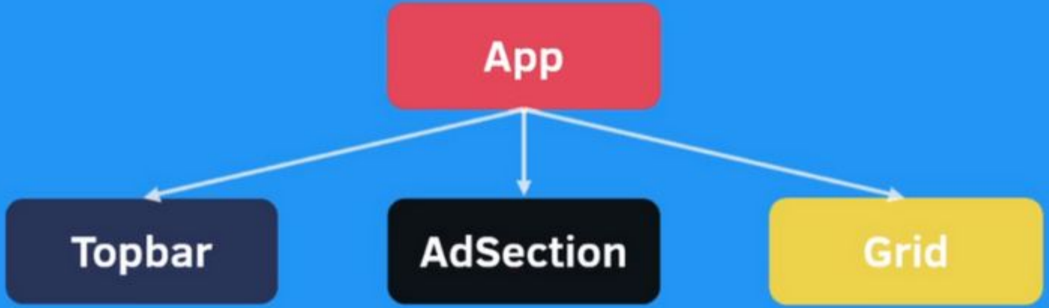

os
About
Contact
UserAvatar

React App Structure

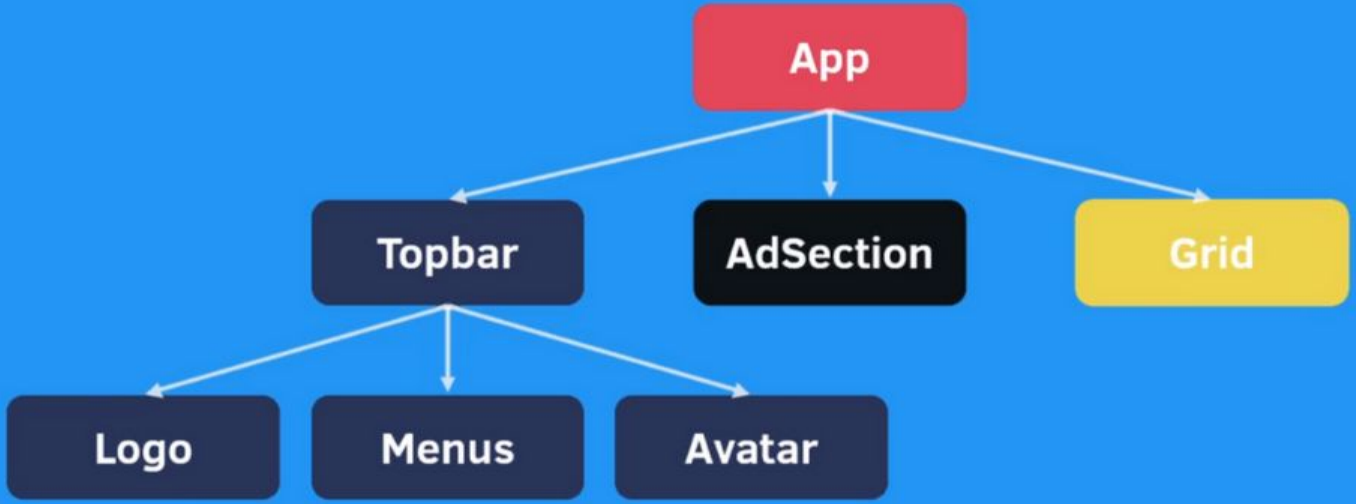
React App Structure



React App Structure



React App Structure



Hello World App!!

Create React App is an officially supported way to create single-page React applications. It offers a modern build setup with no configuration.

You’ll need to have Node >= 8.10 on your local development machine.

To create a new app, follow these steps:

node -v (version), npm -v()

npx create-react-app my-app

cd my-app

npm start

Once the local server starts, then open http://localhost:3000/ to see your app.

INTRO TO JSX

Understanding JSX

This funny tag syntax is neither a string nor HTML. It is called JSX, and it is a syntax extension to JavaScript.

It is used to describe what the UI should look like. It brings the HTML markup and the display logic together. It couples the rendering logic with other UI logic.

There are some {} in this JSX code. Inside these curly brackets you can write JavaScript expressions (which generate some value by default)

For example,

```
const element = <span>Hello {name}!!</span>
```

JSX = Write Lesser Code

For eg, Adding Elements Dynamically.

HTML

```
<ol id="demo"></ol>
```

Vanilla JavaScript

```
var firstName = “John”
var list= document.getElementById(demo).value;
var entry = document.createElement('li');
entry.appendChild(document.createTextNode(firstName));
list.appendChild(entry);
```

JSX

```
<ol><li>{firstname}</li></ol>
```

JSX Basics

1. Embed Expressions.

For eg,

```
const greetings = <h1>Welcome, {firstname}</h1>


fullName = (firstName,  lastName) => {
    return firstName + lastName
}
const greetings = <h1>Welcome, {fullName(“John”, “Lark”)} </h1>
```

JSX Basics

2. HTML Attributes.

For eg,

<a className=”GoogleLink” href=”https://www.google.com” >Link</a>

3. Dynamic Attributes.

For eg,

const profilePic = <img src={user.profilePic} alt=”Profile Pic” />

React Components

To use JSX we need to import the ‘React’ module from the installed ‘react’ package.

```
import React from ‘react’;

const MenuItem = () {

        return <span>Hello React</span>

}

export default MenuItem;
```

React.createElement()

We can also create elements using the React.createElement() method.

When React creates component/element, it calls this method, which takes three arguments.

- The component/element name.

- An object representing the element’s properties also called props.

- An array of the element’s children.

Create Component from Lists

We can use map to iterate list items and generate dynamic HTML elements.

```
const blogs = [{title:’’, desc:’’}, {title:’’, desc:’’}, ...]

const blogList = blogs.map((item, pos) => {

    return <div key={pos}> <h3>{item.title}</h3> <p>{item.desc}</p> </div>

});
```

Key helps React identify which components have changed. All the elements inside an array should have a unique key.

COMPONENT STYLES

Adding Inline Styles

To add inline styles you can simply use the styles attribute and give it an object.

For eg,

```
<div styles={{backgroundColor: ‘blue’, fontSize: ‘16px’}}>Hello<div>


const styles = {
    backgroundColor: ‘blue’,
    fontSize: ‘16px’
}
<div styles={styles}>Hello<div>
```

Adding External Styles

To add external styles, just need to import the stylesheets file.

To select JSX elements use the className property.

For eg,

```
import ‘./App.css’;
<h3 className=”Heading”> Welcome to React </h3>


.Heading {
    color: red;
}
```

How to create Mobile Responsive Components?

To create mobile responsive components, we can use Media Queries.

For eg,

```
@media (max-width: 720px) {
    background-color: white;
}
```

Homework!!

- Design the Blog WebApp using React.
- Read about diff between react and angular.
- Read about top 5 features of React? Why you should use React??

COMPONENT IN DETAILS

Functional Component

The simplest way to define a component is to write a JavaScript function.

For eg,

```
const blogList = () => {
    return() {
        <div>
            <h3>Blog 1</h3>
            <p>Blog Desc 1</p>
        </div>
    }
}
```

Modules Export and Import

ES6 enables developers to write modular code. Basically, we can split reusable code into separate files.

Export makes a module accessible in other files. Modules can be exported in two ways - *default* and *named* export.

Import is used to add different modules in a JavaScript file. It has different syntax based on the export method used by the target module.

Component Props

This props is used to pass data to the components. It is an object with some key-value pairs to hold passed data. It is a read-only object.

```
<BlogItem title={'---'} desc={'----'} />
const BlogItem = (props) => {
  return (<div>
          <h3>{props.title}</h3>
          <p>{props.desc}</p>
     </div>);
}
```

Introduction to CSS Modules

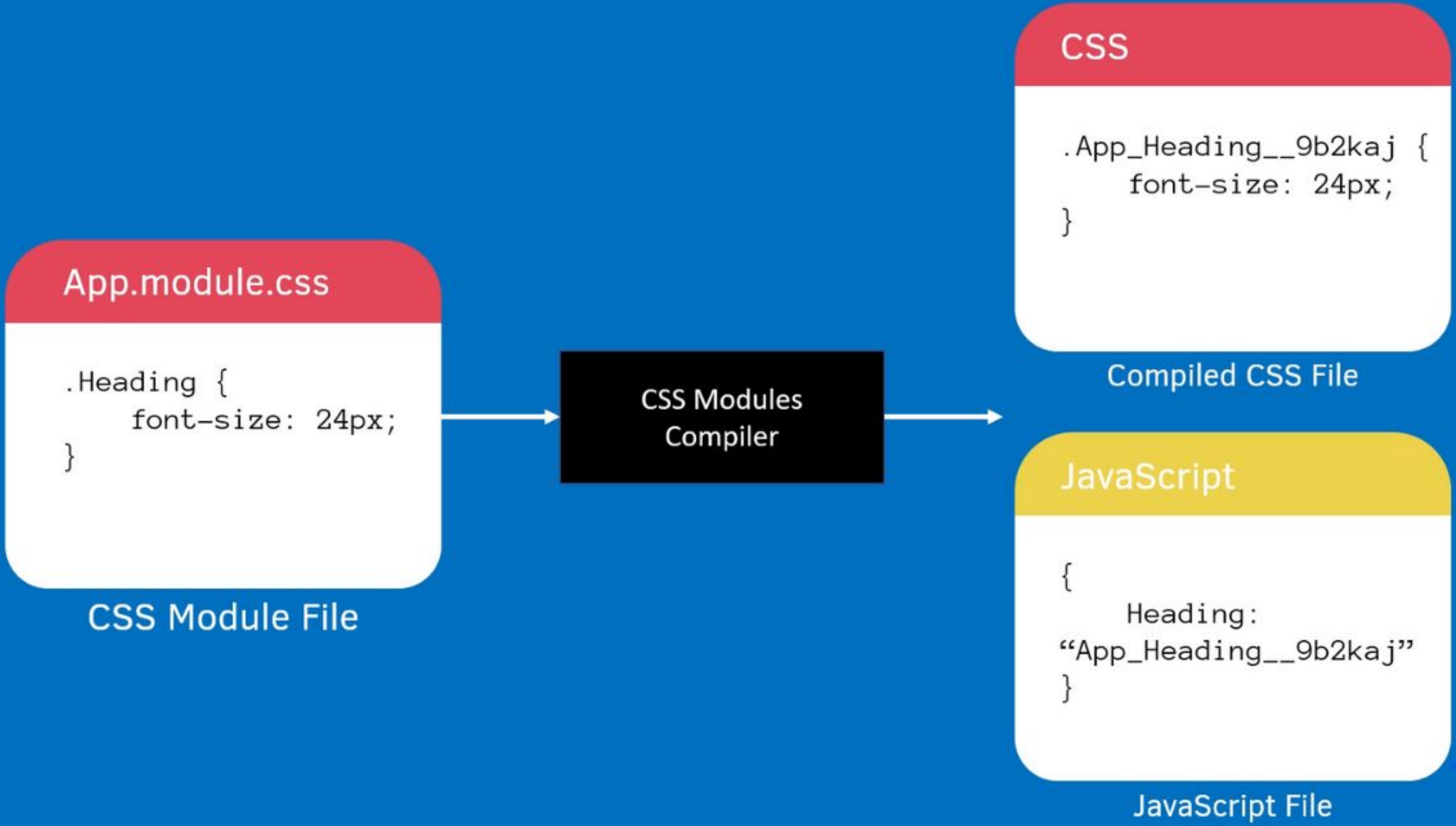
Components are modular and independant. Having global styles is the only dependency between different components. To make components truly independent we need to make style classes scoped locally to the component.

This is where CSS Module comes into picture. It creates a local scope for the CSS classes. It makes our components completely independent and saves our app from all the troubles of global classes.

It creates a unique class name for all the classes that we write using the file name, class name and hash value. For eg,

heading -> app_heading__93BS23

Create CSS Modules File: FileName.module.css



```
.Heading {
    font-size: 24px;
}
```

```
.App_Heading__9b2kaj {
    font-size: 24px;
}
```

```
{
    Heading:
“App_Heading__9b2kaj”
}
```

Class-based Component

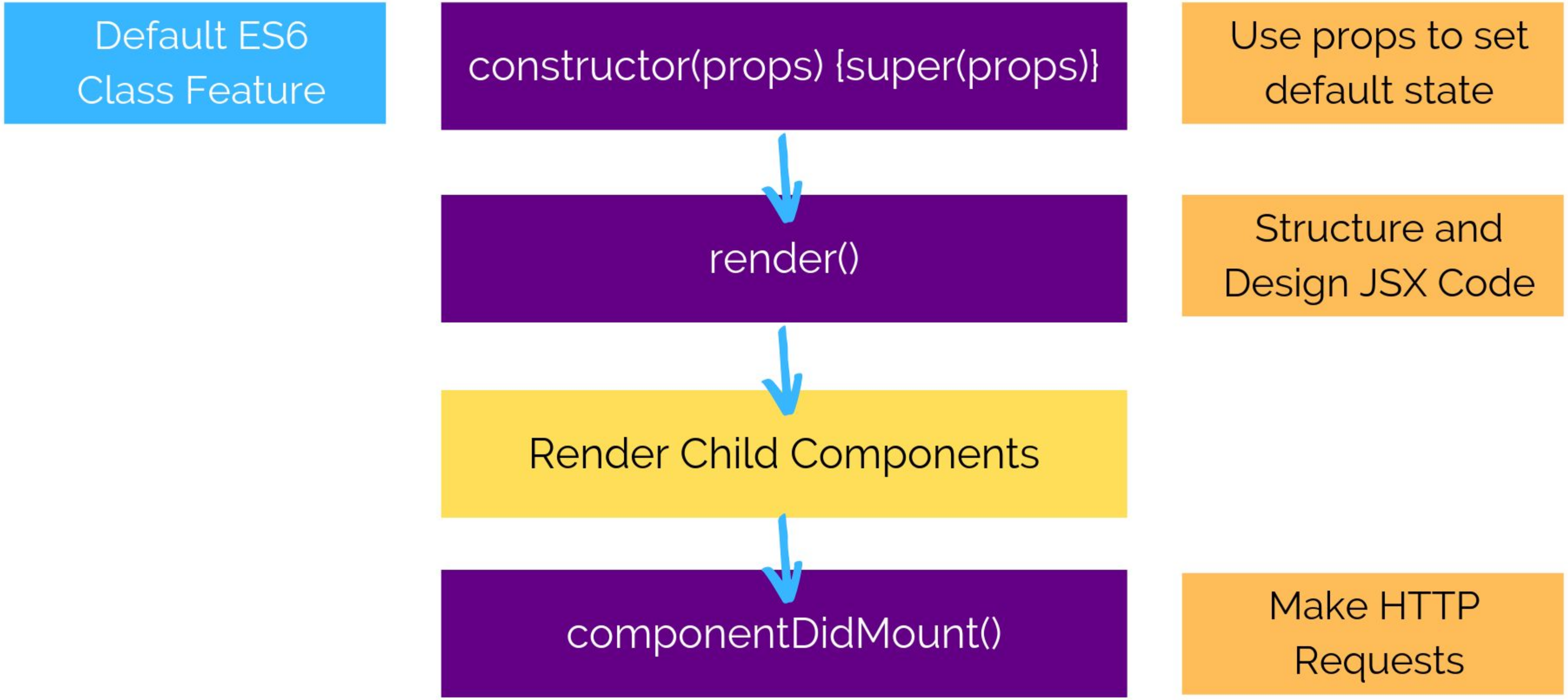
Class-based components introduce a new concept called state. It is an object that determines how that component renders & behaves. It is similar to props, but it is private and fully controlled by the component.

For eg, a like button by default it could represent liked or normal state and when a user clicks on it, it updates the value.

To define class based components we use ES6 class syntax. A class-based component inherits the Component class of React. This Component class gives the component access to some properties and methods like state and setState etc.

It has a render method. So everytime the setState method is called it results in calling the render method.

Component Lifecycle - Creation


Default ES6 Class Feature
constructor(props) {super(props)}
Use props to set default state
render()
Structure and Design JSX Code
Render Child Components
componentDidMount()
Make HTTP Requests

Component Lifecycle - Updation

shouldComponentUpdate(nextProps, nextState)	Decide whether to update or not?
render()	Structure and Design JSX Code
Render Child Components	
componentDidUpdate()	Make HTTP Requests

shouldComponentUpdate()

Rendering is a heavy process. Especially when the layout is very complex.

Everytime the setState() function is called. It starts the entire cycle of component update. A new virtual DOM is generated, compared with the old version of virtual DOM and then it updates the actual DOM. This can hurt the performance of a React App drastically if done wrong.

So, the rule is simple. Keep re-renders as minimum as possible.

We can avoid unnecessary re-renders in the shouldComponentUpdate() lifecycle hook. It gives us access to nextProps and nextState. We can compare if the value is changed or not. If the value is changed then we can re-render or else we can can avoid the re-render.

Syntax:
```
shouldComponentUpdate(nextProps, nextState) {

        return true/false;

}
```

What is DOM?

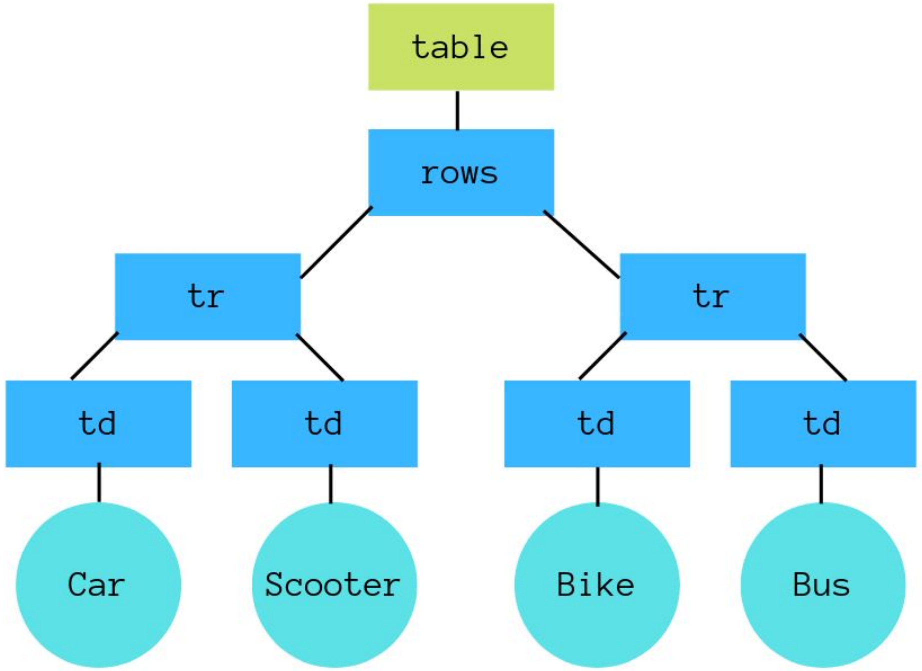
It stands for Document Object Model.

It is the tree representation of the HTML document. In other words, it shows the structure of an HTML document.

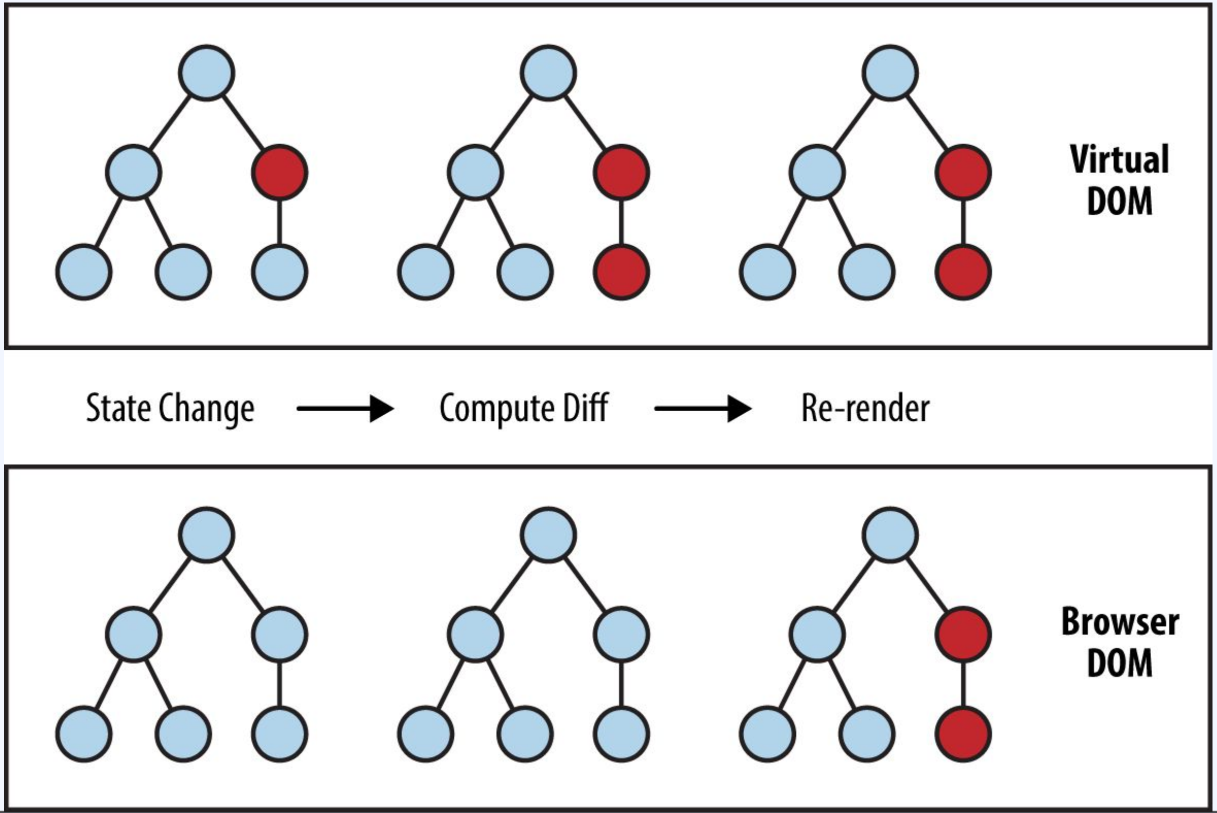
When you update HTML elements, it results in repaints and reflows. Both of these are pretty heavy tasks. That’s why updating DOM is a heavy process. Frequently, updating DOM results in performance issues, in turn making the application slow.

What is DOM?

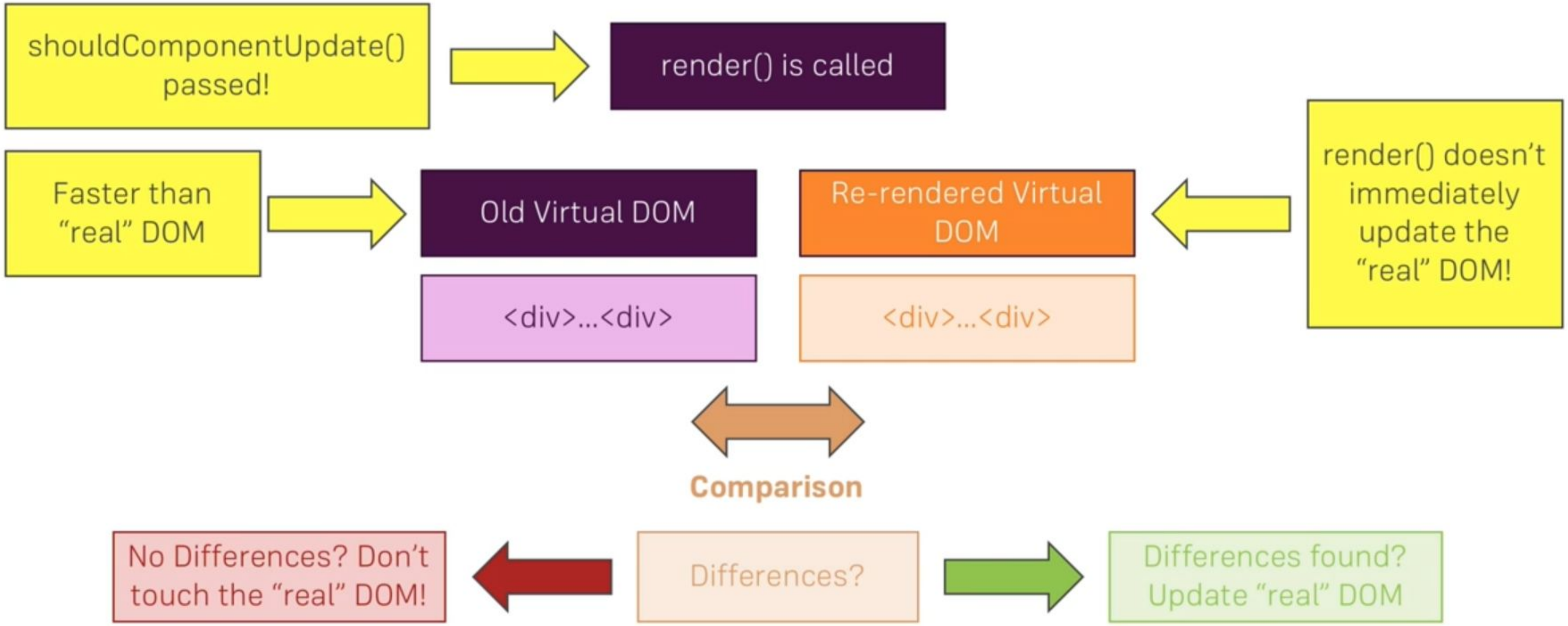
```
<table>
  <rows>
    <tr>
      <td>Car</td>
      <td>Scooter</td>
    </tr>
    <tr>
      <td>Bike</td>
      <td>Bus</td>
    </tr>
  </rows>
</table>
```



Virtual DOM


Virtual DOM
State Change
Compute Diff
Re-render
Browser DOM

How React Updates The DOM


shouldComponentUpdate() passed!
render() is called
Faster than "real" DOM
Old Virtual DOM
<div>…<div>
Re-rendered Virtual DOM
<div>…<div>
render() doesn't immediately update the "real" DOM!
Comparison
No Differences? Don't touch the "real" DOM!
Differences?
Differences found? Update "real" DOM

Pure Components

PureComponent is another class provided by React. It automatically checks if the state or props has not changed then it will avoid a re-render.

It does a shallow comparison of props and states data. This doesn’t gives us a lot of control but works well for cases when the data is simple.

For eg,

```
import {PureComponent} from ‘react’;

class Person extends PureComponent {

        //Component Code

}
```

Homework!!

Write the answers in Github Wiki. Create a new repo for React.

- Write about What is CSS Modules and Why do you need it??
- Difference between stateful and stateless component?
- Difference between props and state?
- Limitation of PureComponent
- Create the counter app with Count in Topbar. //Just make a couple of changes instead of Count. Make it like and dislike. Two buttons to inc/dec likes. Two buttons to inc/dec dislikes.

DEBUGGING

Using React DevTools

React provides us with a tool to inspect the props, states and structure of our components.

It is different from Browser Elements Explorer. With elements explorer you just see the HTML elements but with React Dev Tools you see the components as well.

To Install

1. Search React Developer Tool Extension

2. Install the extension

3. Inspect page and goto React Tab.

CONNECTING REACT APPS TO WEB

HTTP Requests in React

React is a library and it doesn’t comes with out-of-the-box tools to handle HTTP requests.

To handle HTTP request we can use packages like fetch and axios.

Axios is the one of the most popular library to handle HTTP requests in React.

To install Axios:

1. Search axios npm in the browser.
2. Open the webpage.
3. Copy the command to install axios.
4. Open your terminal and run the copied command.

Axios Syntax

```
import axios from axios;

axios.get(“url”)

.then((response) => {

    // handle success

})

.catch((error) => {

    // handle error

})

console.log(‘ABC’)
```

Test APIs: https://jsonplaceholder.typicode.com/posts

Async/Await

Async and await is used to write asynchronous code.

For eg,

```
const mFunc = async () => {

    const response = await axios.get().then().catch()

    return Promise.resolve(reponse);

}
```

Putting the keyword async before a function tells the function to return a Promise. If the code returns something that is not a Promise, then JavaScript automatically wraps it into a resolved promise with that value.

Inside a function marked as async, you are allowed to place the await keyword in front of an expression that returns a Promise. When you do, the execution is paused until the Promise is resolved.

Sending Data to Server

For sending data to backend we could use the post() function of axios.

```
import axios from axios;

const postData={}

axios.post(“url”, postData,  { headers: {Authorization: ‘Token Azu09hdh’}})

.then((response) => {

    // handle success

})

.catch((error) => {

    // handle error

})
```

Deleting Data on Server

For deleting data at backend we could use the delete() function of axios.

```
import axios from axios;

axios.delete("url/id", { headers: {Authorization: 'Token Azu09hdh'}})

.then((response) => {

    // handle success

})

.catch((error) => {

    // handle error

})
```

ROUTING

Setting up React-Router

React is a library and it doesn’t comes with out-of-the-box tools to handle page redirection. For this purpose we have a package called React Router DOM. The React Router DOM package requires another package called React Router.

To install React Router:

1. Search React Router npm in the browser.
2. Open the webpage.
3. Copy the command to install React Router.
4. Open your terminal and run the copied command.
5. Similarly do it for React Router DOM.

Setup

To create endpoints use the <Links to={}> component.

To map these endpoints with components use <Route path={} render={() => <Component />}>

To use the above components we need to wrap the main App in the BrowserRouter component.

For eg,

<Route path={“/”} component={HomePage} for homepage

<Route path={“/details”} render={() => <DetailsPage />} for the details page

To select one route at a time, we can use Switch component. With switch the sequence matters a lot.

Practice Problem

Create a WebApp:

- Create a Topbar with 3 links -> Home, About, Contact
- When user clicks on Home show a component which says "This is Home page"
- When user clicks on About show a component which says "This is About page"
- When user clicks on Contact show a component which says "This is Contact page"

Passing Route Parameters

To pass normal params you can simply pass the value the normal way.

To pass the url params, you need to add “:id” in the endpoint to create a variable for id.

To pass this id down the component just spread and send the props passed by the Router to component. Now you can access this variable in the match object.

For eg,

<Router path={“/details/:id”} render={(props) => <DetailsPage {...props}/>} for the details page

Practice Problem

- Create a homepage. Load the video list. Show just the title.
- Create a details page which shows the video title.
- On the video card click -> redirect to the detail page.
- Once the details page is loaded send a call to the backend and get details of the video to show the title.

API Endpoints:

Video Listing: http://5d76bf96515d1a0014085cf9.mockapi.io/playlist/

Video Details: http://5d76bf96515d1a0014085cf9.mockapi.io/playlist/1

Handling 404 Pages

To show a 404 page when the entered URL does not matches, you can use the “Redirect” component provided by React Router DOM.

For eg,

<Route exact to={‘/’} render={() => <HomePage />} //We are using exact so only “/” should match for HomePage.

<Route render={() => <NotFound />} /> //Just add this route at the last so that if nothing matches it just renders this component.

Redirect Component

To redirect your webapp to an endpoint, you can use the “Redirect” component provided by React Router DOM.

For eg,

<Route to={/register} render={() => <RegisterPage />} />

<Redirect to={‘/login”} /> to redirect a user to login page.

Higher Order Component (HOCs)

These are components which wrap around other components to give an added functionality.

For eg,

Page Loader. It is required by almost all the components which load data from backend. Now to show a loader you can simply pass some props and based on that either you can show the actual content or the loader.

FORMS

Handling Forms

There are two ways to create Forms in React - **Uncontrolled Forms and Controlled Forms.**

We can create forms using the normal HTML way, this is called uncontrolled form because the state of the form is handled by the form itself.

Another way is to handle the entire state using React, this is called controlled form.

On submit, we can extract values from the form or fetch it from our state and send it to the backend using an HTTP post call.

PropTypes

PropTypes

The PropTypes are used to validate the props which are being passed to a component. It ensures that the component receives correct type of data in the props. **It works for both function and class-based components.**

Installation: To use Proptypes, you gotta install **prop-types** package from npm.

Usage:

```
import PropTypes from 'prop-types';

VideoCard.propTypes = {

   id: PropTypes.number.isRequired,

   title: PropTypes.string.isRequired,

   thumbnail: PropTypes.string

}
```

PropTypes

We can also set default values for a component prop.

Usage:

```
VideoCard.defaultProps = {
  isPlaylistCard: false
}
```

Full Reference: https://reactjs.org/docs/typechecking-with-proptypes.html#proptypes

REDUX

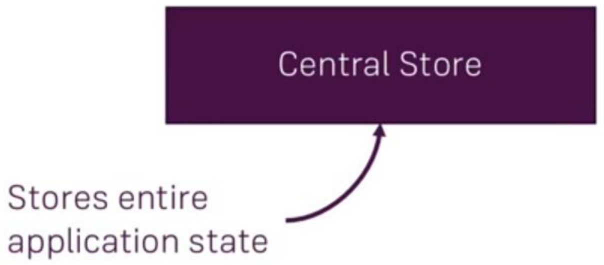
Intro to Redux

Redux is an open-source JavaScript library for managing application state. **It is independent of React.**

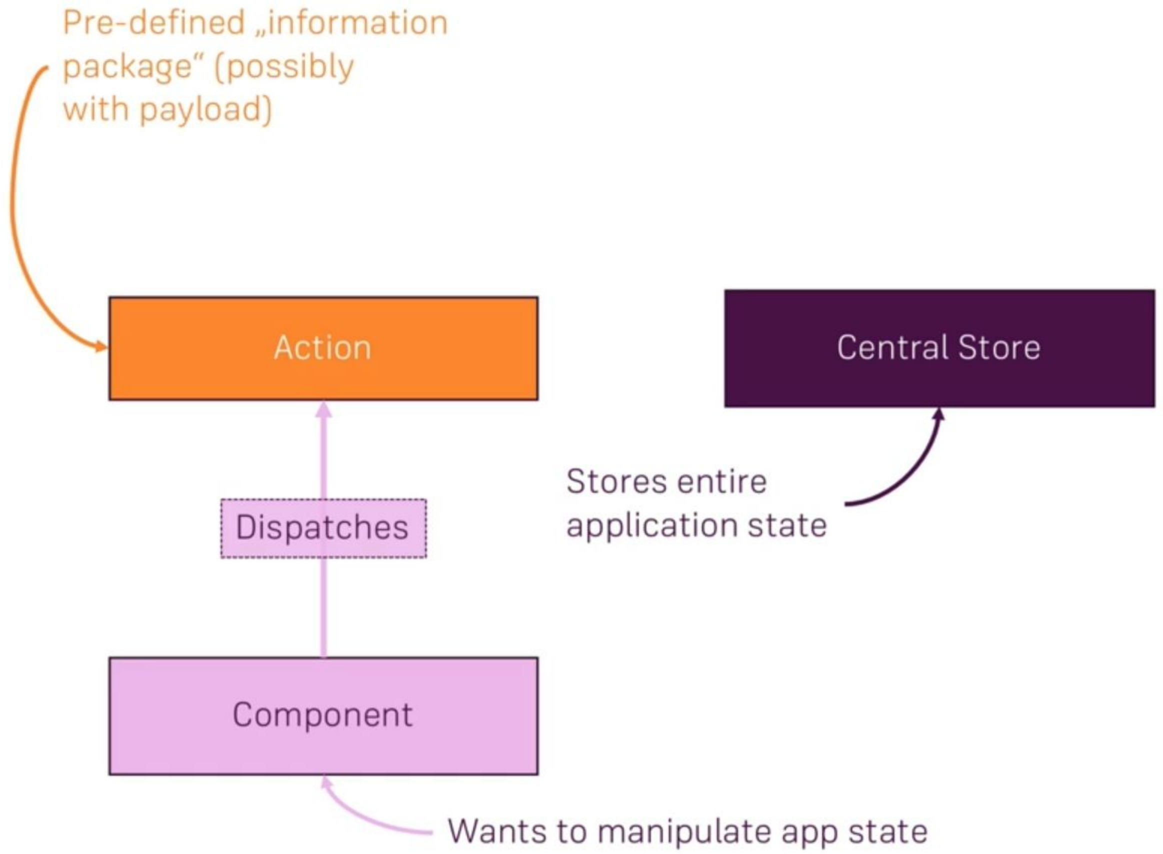
To install Redux:

1. Search Redux npm in the browser.
2. Open the npm webpage.
3. Copy the command to install Redux.
4. Open your terminal and run the copied command.
5. Start your app.

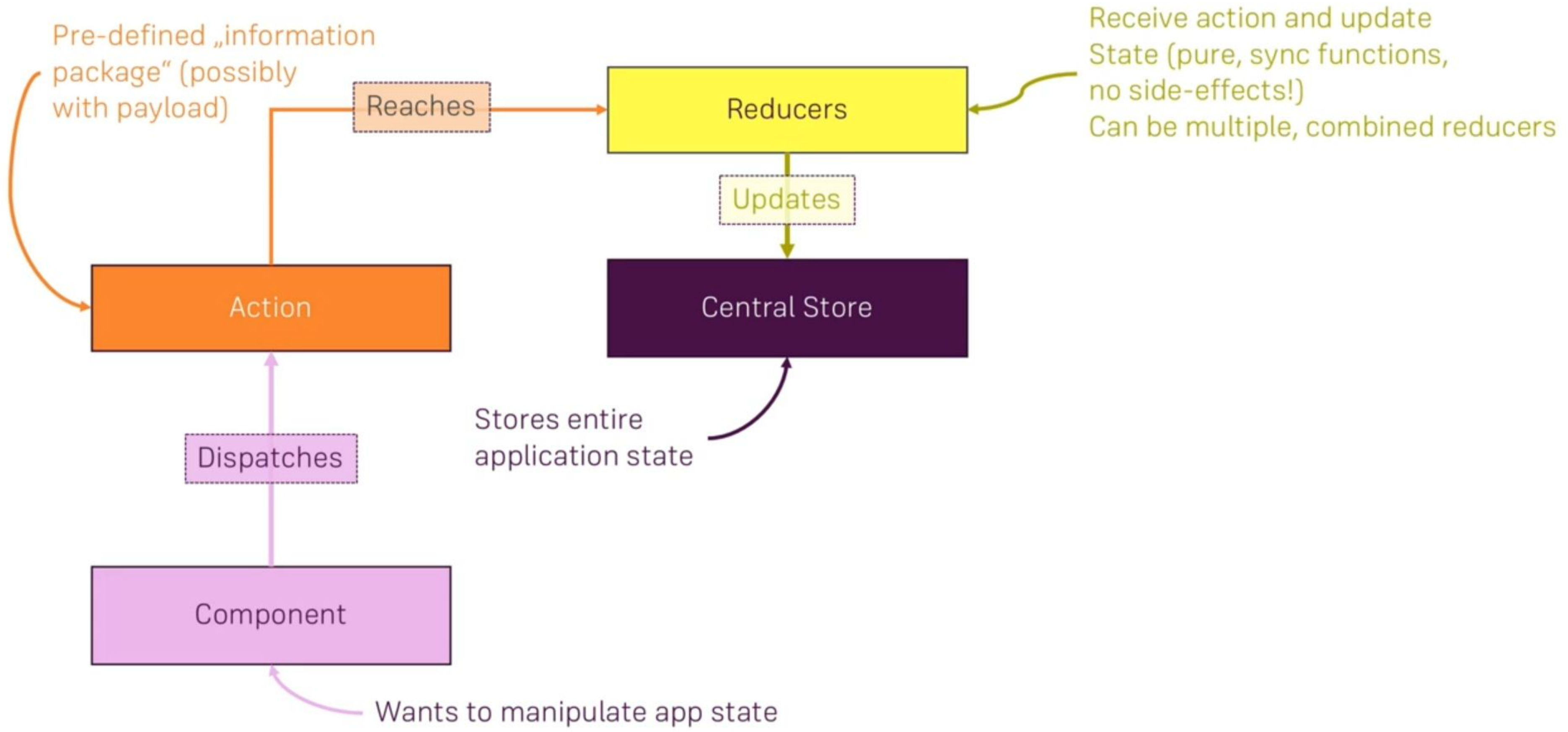
Redux to the Rescue!



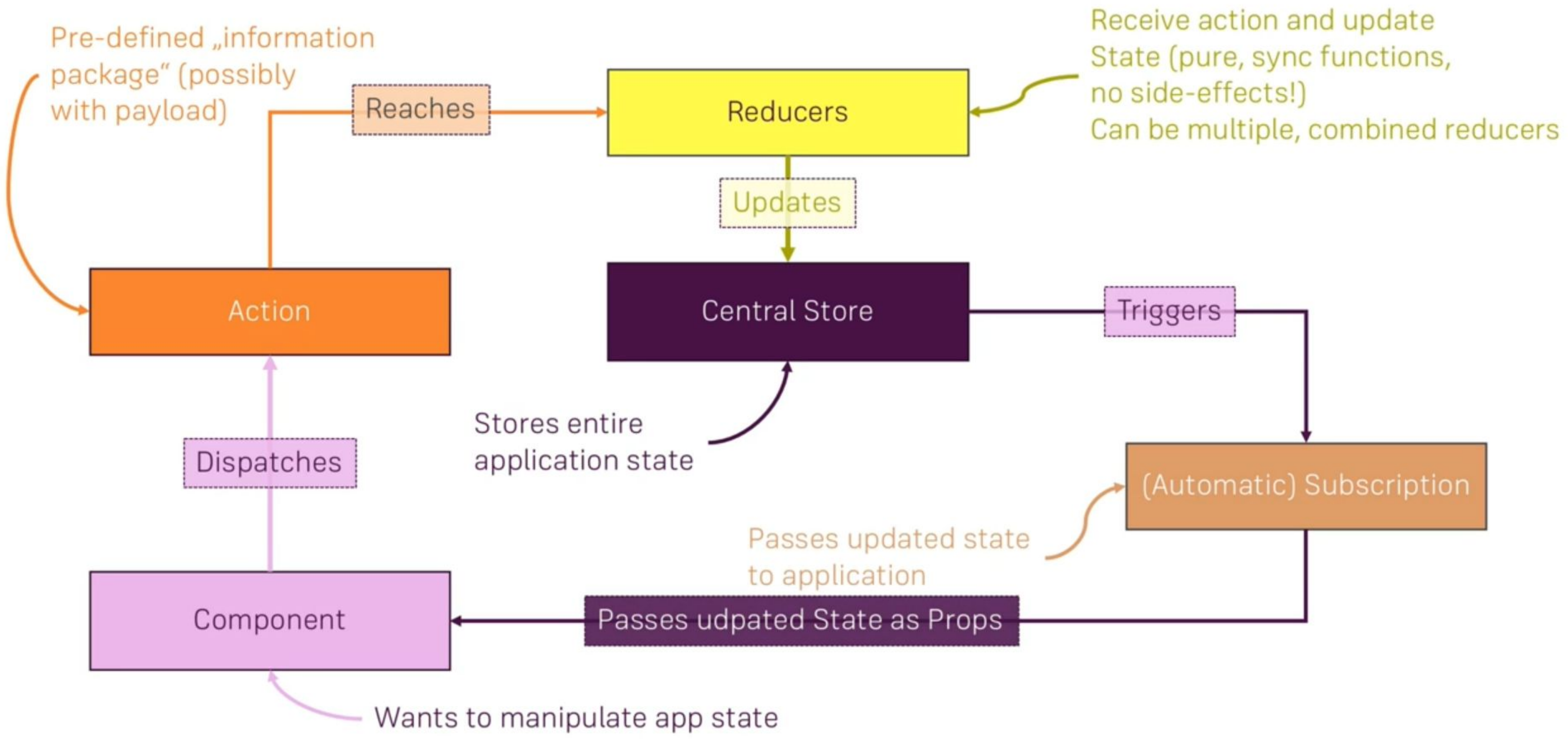
Redux to the Rescue!


Pre-defined „information package“ (possibly with payload)
Action
Central Store
Dispatches
Stores entire application state
Component
Wants to manipulate app state

Redux to the Rescue!


Pre-defined „information package“ (possibly with payload)
Reaches
Reducers
Receive action and update State (pure, sync functions, no side-effects!)
Can be multiple, combined reducers
Updates
Action
Central Store
Stores entire application state
Dispatches
Component
Wants to manipulate app state

Redux to the Rescue!


Pre-defined „information package“ (possibly with payload)
Reaches
Reducers
Receive action and update State (pure, sync functions, no side-effects!)
Can be multiple, combined reducers
Updates
Action
Central Store
Triggers
Stores entire application state
Dispatches
(Automatic) Subscription
Passes updated state to application
Component
Passes udpated State as Props
Wants to manipulate app state

Connecting Redux to React

To connect Redux with React, we need another package, called React-Redux.

To install React-Redux:

1. Search React-Redux npm in the browser.
2. Open the npm webpage.
3. Copy the command to install React-Redux.
4. Open your terminal and run the copied command.
5. Start your app.

Connecting Redux to React

```
import {createStore} from 'redux';

import MainReducer from './.../MainReducer';

import {Provider} from 'react-redux';


let globalStore = createStore(MainReducer);


ReactDOM.render(<Provider store={globalStore}><App /></Provider>, document.getElementById('root'));
```

Connecting Redux to React

```
import {connect} from 'react-redux';


<p>Total Likes: {props.topbarTotLikes}</p>

const mapGlobalStatetoProps = (globalState) => {

    return {

        topbarTotLikes: globalState.totalLikes

    }

}

export default connect(mapGlobalStatetoProps)(Topbar);
```

Connecting Redux to React

```
<button onClick={this.props.onIncrementLike}>Increment Like</button>


const mapDispatchToProps = (dispatch) => {
    return {
        onIncrementLike: () => dispatch({type: ACTION_INCREMENT_LIKE}),
        onDecrementLike: () => dispatch({type: ACTION_DECREMENT_LIKE}),
    }
}
export default connect(null, mapDispatchToProps)(HomePage);
```

HOOKS

Intro to Hooks

Hooks are functions that let you “hook into” **React state and lifecycle features** from function components.

Hooks don’t work inside class based components they only work inside the function based components.

React provides a few built-in Hooks like useState, useEffect etc. You can also create your own Hooks to reuse stateful behavior between different components

Simple Hooks Example

Initializing State:

Class-based Components

```
state = {
  totalCount: 0,
}
```

Hooks:

```
const [totalCount, setCount] = useState(0)
```

Simple Hooks Example

Updating State:

Class-based Components

```
increaseCount = () => {
    const updatedVal = this.state.totalCount + 1;
    this.setState({totalCount: updatedVal})
}
```

Hooks:

```
const [totalCount, setCount] = useState(0)
increaseCount = () => { setCount(totalCount + 1) }
```

Simple Hooks Example

Creating multiple state variables:

```
const [totalCount, setCount] = useState(0)

const [totalLikes, setLikes] = useState(0)

const [totalDislikes, setDislikes] = useState(0)

const [videoList, setVideoList] = useState([])
```

Making Side Effects

```
useEffect(() => {
    Axios.get()
    .then(() => {
        //Update the state here!!
    })
})
```

DEPLOYMENT

Deployment Steps

Check (& Adjust) Basepath	`<BrowserRouter basename=“/my-app/”>`
Build & Optimize Project	**npm run build** in create-react-app project
Server must ALWAYS serve index.html (also for 404 cases)	To ensure that Routing works correctly
Upload Build Artifacts to (static) Server	In **/build** folder when using create-react-app

Hosting with Firebase

To host your React App with Firebase, please follow these commands.

Run command *npm install -g firebase-tools* to install firebase-cli.

Go to your project directory.

Run command *firebase login* to login your firebase account in firebase-cli.

Run command *firebase init* to initialize firebase for your project -> Choose hosting -> Choose your firebase project -> Add path to your directory where you have code for deployment(build) -> Select whether it's a SPA(Select Yes) -> Select No to prevent firebase from overriding your index.html.

Run command *firebase serve* to check if everything is configured properly.

Run command *firebase deploy* to deploy your website.

WEBPACK

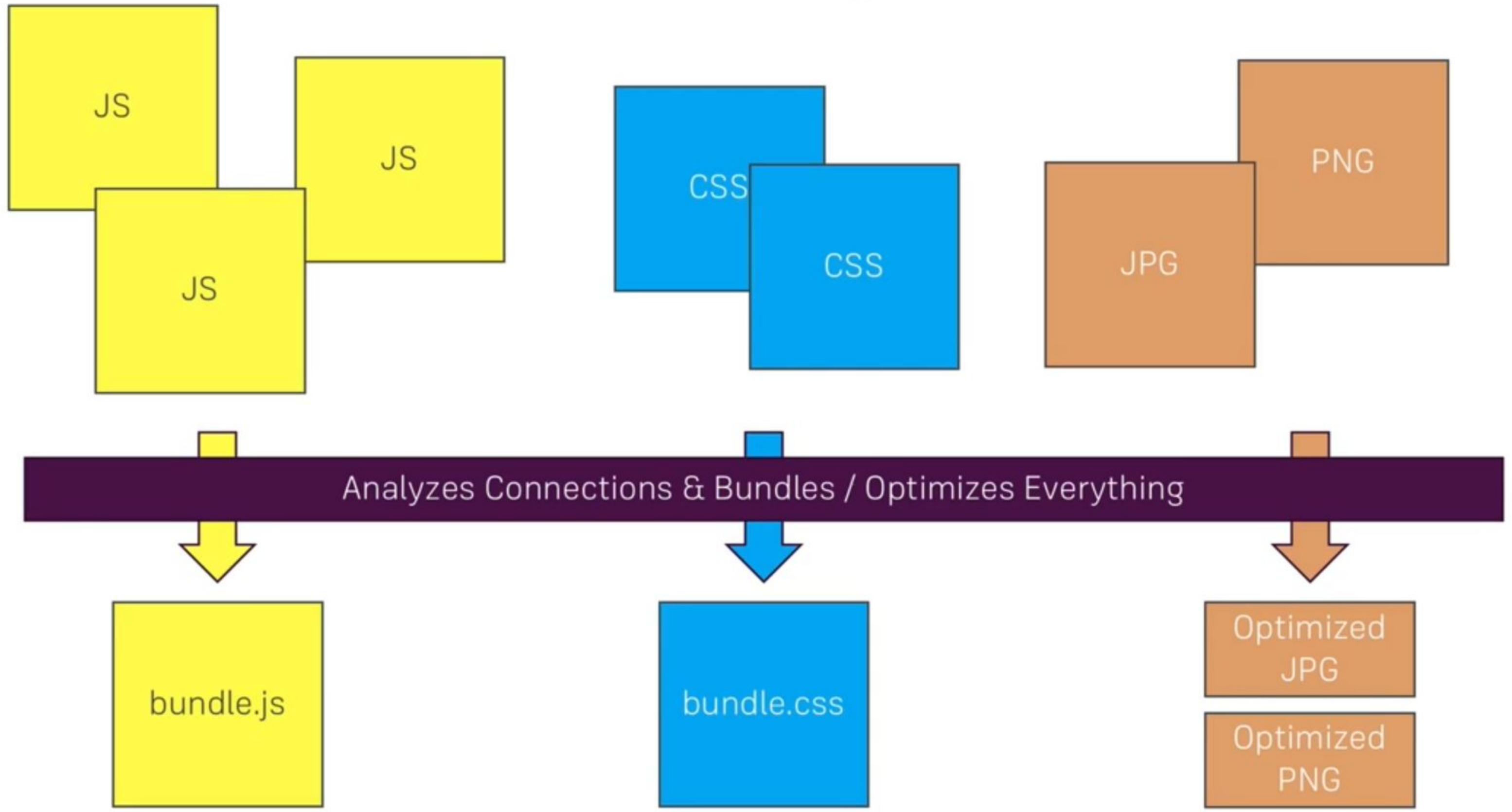
What is Webpack?

Webpack is an open-source JavaScript module bundler.

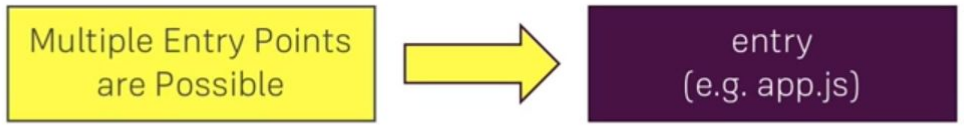
It is a module bundler primarily for JavaScript, but it can transform front-end assets like HTML, CSS, and images if the corresponding plugins are included.

It takes modules with dependencies and generates static assets representing those modules.

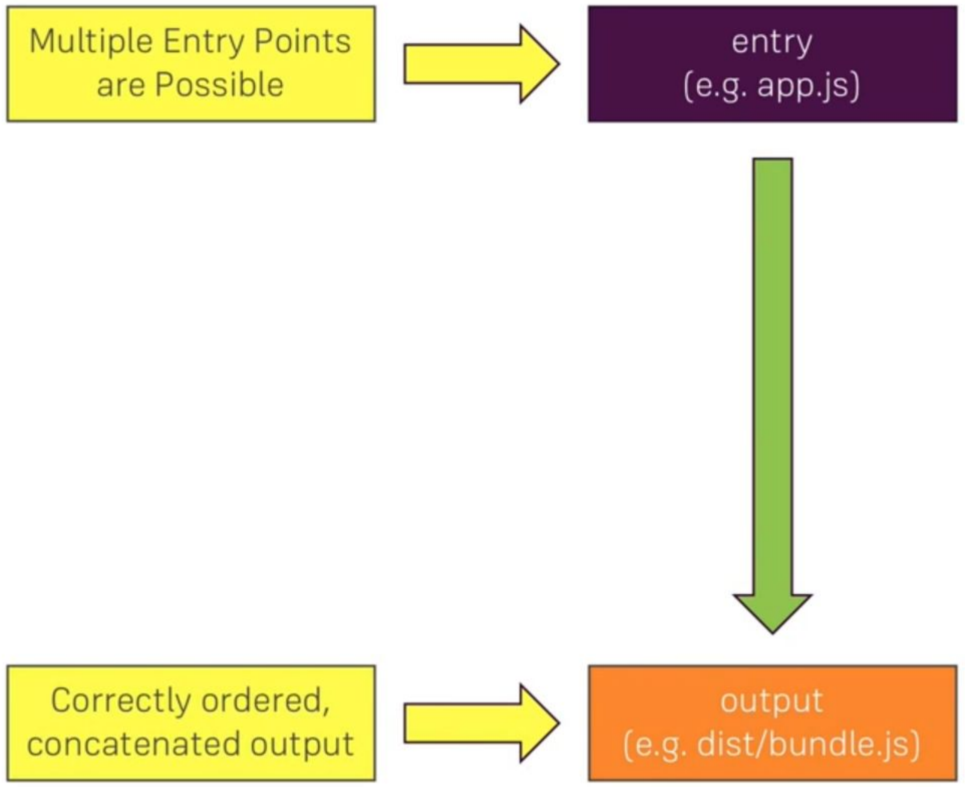
What is Webpack?



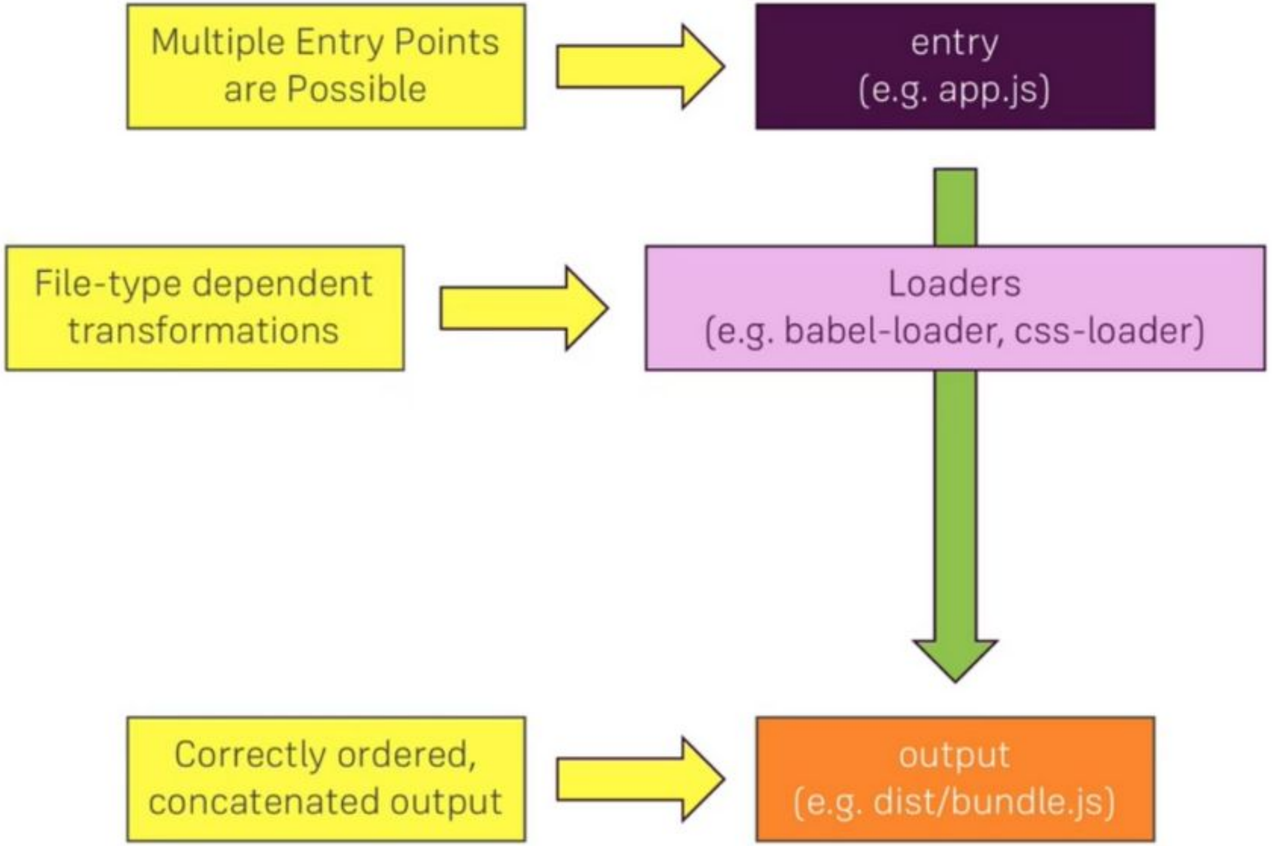
How it Works



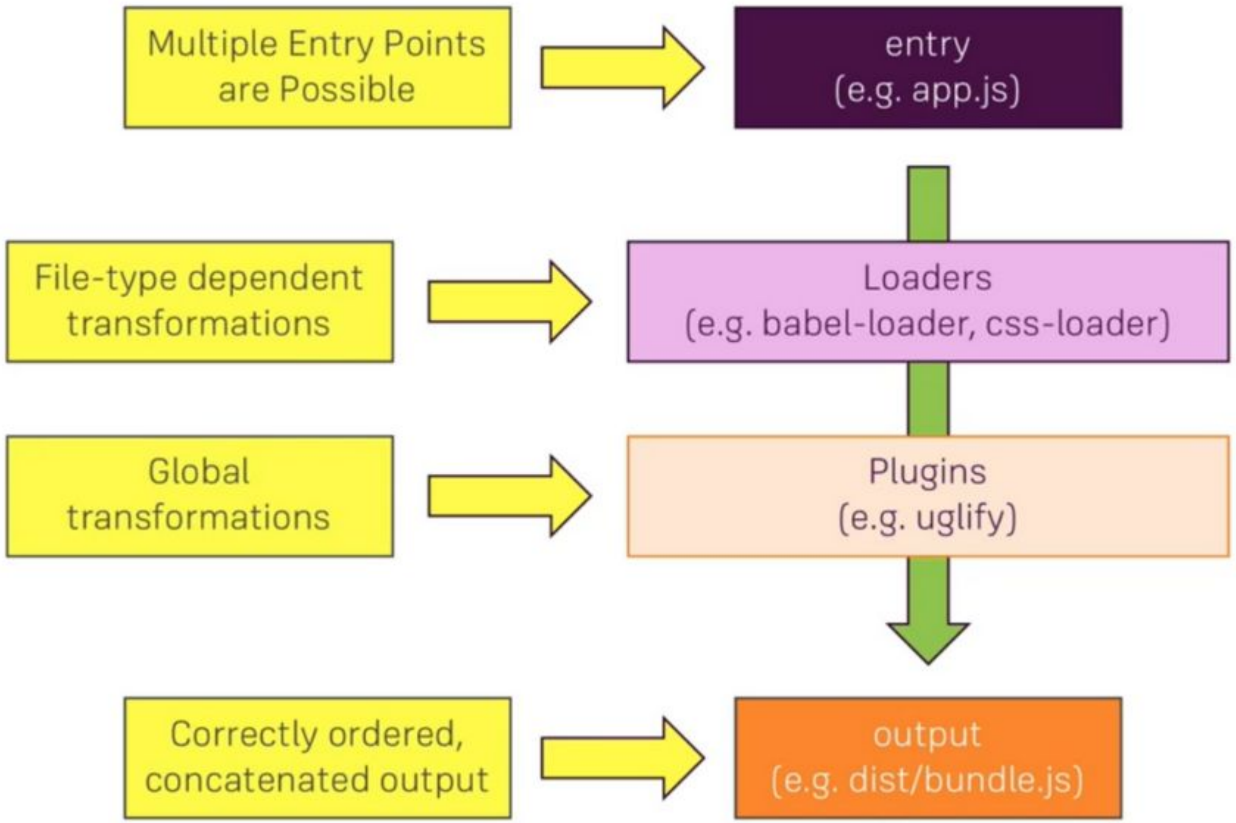
How it Works


Multiple Entry Points are Possible
entry (e.g. app.js)
Correctly ordered, concatenated output
output (e.g. dist/bundle.js)

How it Works


Multiple Entry Points are Possible
entry (e.g. app.js)
File-type dependent transformations
Loaders (e.g. babel-loader, css-loader)
Correctly ordered, concatenated output
output (e.g. dist/bundle.js)

How it Works


Multiple Entry Points are Possible
entry (e.g. app.js)
File-type dependent transformations
Loaders (e.g. babel-loader, css-loader)
Global transformations
Plugins (e.g. uglify)
Correctly ordered, concatenated output
output (e.g. dist/bundle.js)

TESTING

Testing Tools

Test Runner	Executes Tests and provides Validation Library	Jest
Testing Utilities	"Simulates" the React App (mounts components, allows you to dig into the DOM)	Enzyme

KEEP LEARNING!!