I graduated in 2014 from Yokohama National University of science and technology.

I wrote a thesis, the title is “The Study on Content Creation by Peer Production in Virtual Space”.

The contents are following

Peer production and user-created content is becoming an important element in modern eLearning, supported by the development of the Internet from a one-way information distribution channel to a two-way communication channel.

In the peer production of eLearning content, the essential feature is that the learners are also acting as creators of the content and that the separation between an "author" and a "consumer" is blurring.

In practice, learners are no longer purely consumers but they actively participate in the learning process and thus influence it.

This fundamental feature is also imposing a different view on quality.

Peer production is not only a novel method to produce eLearning content, but it is also an approach to empower a wide variety of professionals to the learning content production. However, the quality management challenge related to this kind of content can undermine the merits of the method.

## Tell Me About Yourself

I was born in US, I went to Japan and graduated from Yokohama National University.

It is in Yokohama, Kanagawa

After graduating the University, I moved to California in May, 2020

I am a Full-Stack developer with 7 years of experience.

My good attitude to Clients and genuine interest in Web development led my life to success.

I experienced in building 20 + big websites and working on web technologies such as Javascript, React, Vuetify, Material-UI, NodeJS, Express, Shopify, Auth0, PHP, Laravel, CodeIgniter, MySQL

Certified in both Frontend and Backend technologies. Spearheaded successful transition from LAMP stack to MEAN which cut latency by 40% and increased effectiveness of database administrators by 20%

* **Most Recent Project**

**premiumteas.ca/**

In this project, the teas are presented in a way that makes it easier for visitors to scroll through and select their desired product. The page focuses on visual representation rather than being covered in too much text. When you click on any product, you’re taken to a different page that has a detailed description of the tea. This includes product information like aroma, caffeine level, brewing time, brewing temperature, and much more information that can be valuable for the buyer

The client asked my team to produce an ecommerce store to promote their newest teas product. I was responsible for building frontend, integrating Shopify API, Payment API such as Visa, paypal, mastercard.

At first, I built clean, modern, and sophisticated design for this store from Figma using ReactJS, Material-UI, React Hook, React Redux, Context.

As well as, I implemented mobile responsive of this site.

Then I collaborated with my manager and my team to effectively develop backend.

For improving authentication, I used JWT on request to get and post all data.

Also, I generated optimized architecture and achieved specific logic, concise coding style for backend. I used GitHub to manage source code with my team members.

After we received enthusiastic client approval, I deployed this store to AWS EC2 and released. It was extremely successful. We reached over 130,050 users through social media, and over 400,000 people through the other outlets. The company saw a 27% increase in incoming as a result of the campaign. After this project, my manager gave me a raise, and I continued to lead this team to complete over 9 successful campaigns for the remainder of my time at the company

**Uniclixapp.com**

This project is for users to manage their various social media accounts such as Twitter, Linked in, and Facebook synthetically.

There are Social media manager, Twitter Booster functions mainly in this project

In Social media manager, there are Manage posts, Media Library, Content Finder and Analytics, Accounts functions.

In Twitter booster, there are chart to see all tweets, Post tweets, Search tweets

The client required me to build website to control users’ various social media accounts

I was responsible for frontend developer

At first, I built smart, clean, and sophisticated design for this website from Figma using ReactJS, Material-UI, React Hook, React Redux, Context.

As well as, I implemented mobile responsive of this site.

Then I collaborated with my manager and my team to effectively fetch social media contents and to implement payment integration.

Also, I generated immutable architecture and used reusable components, performance optimization, and concise coding style for frontend. I used GitHub to manage source code with my team members.

After we released this website, the client gave me and our team enthusiastic feedback

After that the company saw a 17% increase in incoming as a result of the campaign.

* **Why Are You the Best Person for the Job?**

I am a self-motivated person who is willing to go above and beyond on any project, and to learn valuable skills on my own time. For example, I taught myself four computer programming languages in college, simply out of a passion to learn how to code.

My previous job working as a frontend expert provided me with the ideal experience for this position. For five years, I developed many of the skills required for this job, including performance optimization, data visualizing, Api integration, good communication skill, increasing site speed.

* Why Do You Want This Job?

I understand that this is a company on the rise. As I’ve checked your job, you’re planning to launch several new products in the coming months. I want to be a part of this business as it grows, and I know my experience in product development would help your company.

* Why Did You Leave Your Job

Unfortunately, in my last job, I wasn't able to use my training and experience fully.

I looked into the position and was intrigued by the role and by the company.

I wanted to get more exciting opportunity and ideal match for my qualifications.

* What Is Your Greatest Strength?

I have a solid work ethic. When I'm working on a project, I don't just want to meet deadlines. Rather, I prefer to complete the project well ahead of schedule. Last year, I even earned a bonus for completing three important issues one week ahead of time.

* What Are Your Career Goals

My current short-term goal is to develop and use my frontend and communications skills in a job like this one. However, I eventually want to develop into a position that allows me to continue to use these skills while also managing a team group. I will prepare for this goal by taking on leadership positions in team projects, and by developing my professional career through attending leadership conferences such as the one put on annually by your company.

Laravel benefit #1: Laravel makes implementing authentication very simple. Almost everything is configured out-of-the-box. Laravel also provides a simple way to organize authorization logic and control access to resources.

Laravel benefit #2. Laravel provides a clean, simple API over the popular SwiftMailer library. Laravel also provides drivers for SMTP, Mailgun, Mandrill, SparkPost, Amazon SES, PHP's “mail” function, and “sendmail”, allowing an application to quickly get started sending mail through a local or cloud-based service. In addition to support for sending email, Laravel provides support for sending notifications across a variety of delivery channels, including SMS (via Nexmo) and Slack.

Laravel benefit #3. Laravel supports popular cache backends like Memcached and Redis out-of-the-box. By default, Laravel is configured to use the file cache driver, which stores cached objects in the file system. For larger applications, it is better to use an in-memory cache such as Memcached or APC. However, with Laravel it is even possible to configure multiple cache configurations

Laravel benefit #4. Laravel helps to secure the web application by protecting it against the most serious security risks: SQL injection, cross-site request forgery, and cross-site scripting. Laravel itself is secure. We can tell you first hand that the codebase is fanatically guarded, and that the code has been vetted by several people.

Laravel benefit #5. Error and exception handling is already configured for any new Laravel-based project. In addition, Laravel is integrated with the Monolog logging library, which provides support for a variety of powerful log handlers.

Laravel benefit #6. Laravel is built with testing in mind. In fact, support for testing with PHPUnit is included out-of-the-box and a phpunit.xml file is already setup for the application. The framework also ships with convenient helper methods allowing for expressive testing of the applications. It provides easy ways for simulating basic behavior of users (making requests to the application and examining the output, for example, clicking links, filling out forms).

Laravel benefit #7. All Laravel routes are defined in the app/Http/routes.php file, which is automatically loaded by the framework. The most basic Laravel routes simply accept a URI and a Closure, providing a very simple and expressive method of defining routes.

Laravel benefit #8. Laravel is an MVC framework, so separation is already done. See the figure: the full MVC request cycle in a Laravel 5 application.

Laravel benefit #9. The Laravel queue service provides a unified API across a variety of different queue backends. Queues allow you to defer the processing of a time-consuming task, such as sending an e-mail, until a later time, which drastically speeds up web requests to your application.

Laravel benefit #10. In the past, developers have generated a Cron entry for each task they need to schedule. However, this is a headache. Such task schedule is no longer in source control, and developers must SSH into the server to add the Cron entries. The Laravel command scheduler allows for the fluent and expressive defining of command schedule within Laravel itself, and only a single Cron entry is needed on the server.

* **higher order component**

A higher-order component acts as a container for other components.

This helps to keep components simple and enables re-usability.

They are generally used when multiple components have to use a common logic.

* **higher order component with Routes**

When using React Router in React, one can use the Navigate component to navigate a user away from a page in case of a certain condition.

For example, the following example does not render a list if there is no data, but redirects a user to the home page instead:

import { Navigate } from 'react-router-dom';

const List = ({ data }) => {

if (!data.length) {

return <Navigate replace to='/home' />;

}

return (

<ul>

{data.map((item) => {

return <li key={item}>{item}</li>;

})}

</ul>

);

};

export default List;

In this case the redirect is well placed. However, if there is much logic happening before of the conditional, e.g. by using React Hooks, then the logic has to execute even though there may be a redirect.

import { Navigate } from 'react-router-dom';

const List = ({ data }) => {

// lots of hooks here

// which is bad, because they execute

// even though there may be a redirect

// and all the hooks logic may not be used after all

if (!data.length) {

return <Navigate replace to='/home' />;

}

return (

<ul>

{data.map((item) => {

return <li key={item}>{item}</li>;

})}

</ul>

);

};

export default List;

Therefore, you can use a higher-order component (HOC) for the redirect, because when wrapping the component into a HOC, the logic of the HOC would occur before the hooks from the wrapped component:

import { withRedirectIfBlank } from './withRedirect'

const List = ({ data }) => {

// lots of hooks here

return (

<ul>

{data.map((item) => {

return <li key={item}>{item}</li>;

})}

</ul>

);

};

export default withRedirectIfBlank({

redirectCondition: (props) => !props.data.length,

redirectTo: '/home',

})(List);

The HOC implementation could look like the following then:

import { Navigate } from 'react-router-dom';

const withRedirectIfBlank = (config) => (Component) => (props) => {

const { redirectCondition, redirectTo } = config;

if (redirectCondition(props)) {

return <Navigate replace to={redirectTo} />;

}

return <Component {...props} />;

};

export { withRedirectIfBlank };

Higher-Order Components are still useful these days, even though many React developers take them as legacy, because they are from a time when React Class Components where used. Especially when they are used to render conditional JSX. However, if not using any conditional JSX, using a Hook instead of a HOC is often a better design choice in modern React.

* How about pure component

A React component is considered pure if it renders the same output for the same state and props. For this type of class component, React provides the PureComponent base class. Class components that extend the React.PureComponent class are treated as pure components.

In practice, a React pure component looks like the following code.

class PercentageStat extends React.PureComponent {

render() {

const { label, score = 0, total = Math.max(1, score) } = this.props;

return (

<div>

<h6>{ label }</h6>

<span>{ Math.round(score / total \* 100) }%</span>

</div>

)

}

}

export default PercentageStat;

* **Usememo and UseCallback, UseRef**

useCallback

We can use the useCallback hook to return a memoized callback.

It takes a callback function as the first argument, and an array of value that changes for the callback in the first argument to be called.

This is useful when passing callbacks to optimized child components that rely on reference equality to prevent unnecessary renders.

useMemo

useMemo caches values that are computed from a function. The first argument is a function that computes the value, and the second argument is an array with the dependencies that are used to compute the returned value.

useMemo runs during rendering. It’s used as a performance optimization. React may forget previously memoized value and recalculate them on the next render in cases it needs to free memory for offscreen components for example.

Therefore, we should write code that works with useMemo and then add it to optimize performance.

useRef

The useRef hook returns a mutable ref whose current property is initialized to the passed argument. The returned object will persist for the full lifetime of the component.

* **How do you pass data from one component to one other component and what are those**

For passing the data from the child component to the parent component, we have to create a callback function in the parent component and then pass the callback function to the child component as a prop. This callback function will retrieve the data from the child component. The child component calls the parent callback function using props and passes the data to the parent component.

* **Redux and Context API**

Context API is a built-in React tool that does not influence the final bundle size, and is integrated by design.

Context provides a way to pass data through the component tree without having to pass props down manually at every level

Redux is an Open Source Library which provides a central store, and actions to modify the store. It can be used with any project using JavaScript or TypeScript,

Both are excellent tools for their own specific niche, Redux is overkill just to pass data from parent to child & Context API truly shines in this case.

Thus, Context API: Resourceful and ideal for small applications where state changes are minimal

Redux: Perfect for larger applications where there are high-frequency state updates

Create the context

1. Create a context using createContext() and pass the initial state as arguments. Context can also be defined without passing any arguments.

2. Define a function that will deliver the data through the Provider.

3. Using useReducer() hook accepts a Reducer with the default state, then returns the updated state and dispatches a function.

4. Inside the provider function, use useReducer() and pass the Reducer and the initial state as arguments. The state returned and dispatch are then passed as values in the Provider.

Redux requires the following blocks to function:

Actions, Reducers, Store

- Lifecycle methods - Exp of Mount fact, Unmount face

Mounting means putting elements into the DOM.

React has four built-in methods that gets called, in this order, when mounting a component:

constructor()

getDerivedStateFromProps()

render()

componentDidMount()

The render() method is required and will always be called, the others are optional and will be called if you define them

The constructor() method is called before anything else, when the component is initiated, and it is the natural place to set up the initial state and other initial values.

The constructor() method is called with the props, as arguments, and you should always start by calling the super(props) before anything else, this will initiate the parent's constructor method and allows the component to inherit methods from its parent (React.Component).

The getDerivedStateFromProps() method is called right before rendering the element(s) in the DOM.

This is the natural place to set the state object based on the initial props.

It takes state as an argument, and returns an object with changes to the state.

The componentDidMount() method is called after the component is rendered.

This is where you run statements that requires that the component is already placed in the DOM.

Unmounting

Unmounting is when a component is removed from the DOM

React has only one built-in method that gets called when a component is unmounted:

componentWillUnmount()

The componentWillUnmount method is called when the component is about to be removed from the DOM.

- Asynchronization

React-async provides a declarative API to perform any REST API calls using a single React component, allowing declarative programming to be used throughout the application. It takes care of handling errors, promise resolution, and retrying promises, and deals with local asynchronous state.

you can write the very same component as a function with the useEffect hook.

Also we can implement aynchronization by using async/await

- Virtual Dom

React uses Virtual DOM exists which is like a lightweight copy of the actual DOM(a virtual representation of the DOM).

So for every object that exists in the original DOM, there is an object for that in React Virtual DOM.

It is exactly the same, but it does not have the power to directly change the layout of the document.

Manipulating DOM is slow, but manipulating Virtual DOM is fast as nothing gets drawn on the screen.

So each time there is a change in the state of our application, virtual DOM gets updated first instead of the real DOM.

**- How to improve the performance of the application?**

1. Using Immutable Data Structures

2. Function/Stateless Components and React.PureComponent

3. Multiple Chunk Files

4. Using Production Mode Flag in Webpack

5. Dependency optimization

6. Use React.Fragments to Avoid Additional HTML Element Wrappers

7. Avoid Inline Function Definition in the Render Function

8. Throttling and Debouncing Event Action in JavaScript

9. Avoid using Index as Key for map

10. Avoiding Props in Initial States

11. Spreading props on DOM elements

12. Use Reselect in Redux to Avoid Frequent Re-render

13. Avoid Async Initialization in componentWillMount()

14. Memoize React Components

15. CSS Animations Instead of JS Animations

16. Using a CDN

17. Using Web Workers for CPU Extensive Tasks

18. Analyzing and Optimizing Your Webpack Bundle Bloat

19. Consider Server-side Rendering

20. Enable Gzip Compression on Web Server

**- Did you write any API Services? What type of Authentication did you use?**

1. I have ever used REST API and Fetch API.

A REST API is an API that follows what is structured in accordance with the REST Structure for APIs. REST stands for “Representational State Transfer”. It consists of various rules that developers follow when creating APIs.

The way a REST API is structured depends on the product it’s been made for — but the rules of REST must be followed.

The fetch() API is an inbuilt JavaScript method for getting resources from a server or an API endpoint. It’s similar to XMLHttpRequest, but the fetch API provides a more powerful and flexible feature set.

The fetch() API method always takes in a compulsory argument, which is the path or URL to the resource you want to fetch. It returns a promise that points to the response from the request, whether the request is successful or not. You can also optionally pass in an init options object as the second argument.

Once a response has been fetched, there are several inbuilt methods available to define what the body content is and how it should be handled.

2. I used Oauth, JWT in my React project

* **JWT? how it works?**

JWT is JSON Web Token

The React app requests a JWT from the authentication server whenever the user wants to sign on.

The authentication server generates a JWT using a private key and then sends the JWT back to React app.

My React app stores this JWT and sends it to your backend server whenever user needs to make a request.

The backend server verifies the JWT using a public key and then reads the payload to determine which user is making the request

* **JScript - Debouncing or throttling?**

Throttling

In a nutshell, throttling means delaying function execution. So instead of executing the event handler/function immediately, you’ll be adding a few milliseconds of delay when an event is triggered. This can be used when implementing infinite scrolling, for example. Rather than fetching the next result set as the user is scrolling, you can delay the XHR call.

Another good example of this is Ajax-based instant search. You might not want to hit the server for every key press, so it’s better to throttle until the input field is dormant for a few milliseconds

Throttling can be implemented a number of ways. I can throttle by the number of events triggered or by the delay event handler being executed.

Debouncing

Unlike throttling, debouncing is a technique to prevent the event trigger from being fired too often. If you are using lodash, I can wrap the function you want to call in lodash’s debounce function

- Closures in application?

Closures are inner functions that have access to the outer function’s variables and parameters. Even after the outer function’s execution is finished, the inner functions have access to the variables in the outer function. Closures are everywhere in JavaScript

1. What are the features of React?

JSX, Components, Virtual DOM, One-way data-binding, High performance

2. What is JSX?

JSX is a syntax extension of JavaScript. It is used with React to describe what the user interface should look like. By using JSX, we can write HTML structures in the same file that contains JavaScript code

3. What is the virtual DOM?

DOM stands for Document Object Model. The DOM represents an HTML document with a logical tree structure. Each branch of the tree ends in a node, and each node contains objects.

React keeps a lightweight representation of the real DOM in the memory, and that is known as the virtual DOM. When the state of an object changes, the virtual DOM changes only that object in the real DOM, rather than updating all the objects.

4. Why use React instead of other frameworks, like Angular?

Easy creation of dynamic applications: React makes it easier to create dynamic web applications because it provides less coding and provides more functionality, whereas, with JavaScript applications, code tends to get complex very quickly.

Improved performance: React uses virtual DOM, which makes web applications perform faster. Virtual DOM compares its previous state and updates only those components in the real DOM, whose states have changed, rather than updating all the components — like conventional web applications.

Reusable components: Components are the building blocks of any React application, and a single app usually consists of multiple components. These components have their own logic and controls, and they can be reused through the application, which, in turn, dramatically reduces the development time of an application.

Unidirectional data flow: React follows a unidirectional data flow. This means that when designing a React app, we often nest child components within parent components. And since the data flows in a single direction, it becomes easier to debug errors and know where the problem occurs in an application at the moment.

Dedicated tools for easy debugging: Facebook has released a chrome extension that we can use to debug React applications. This makes the process of debugging React to web applications faster and easier.

5. What is the difference between the ES6 and ES5 standards?

These are the few instances where ES6 syntax has changed from ES5 syntax:

Components and Function, exports vs export, require vs import

6. What is an event in React?

An event is an action that a user or system may trigger, such as pressing a key, a mouse click, etc.

React events are named using camelCase, rather than lowercase in HTML.

With JSX, you pass a function as the event handler, rather than a string in HTML.

7. What is an arrow function and how is it used in React?

An arrow function is a short way of writing a function to React.

It is unnecessary to bind ‘this’ inside the constructor when using an arrow function. This prevents bugs caused by the use of ‘this’ in React callbacks.

8. What are the components in React?

Components are the building blocks of any React application, and a single app usually consists of multiple components. A component is essentially a piece of the user interface. It splits the user interface into independent, reusable parts that can be processed separately.

There are two types of components in React:

Functional Components: These types of components have no state of their own and only contain render methods, and therefore are also called stateless components. They may derive data from other components as props (properties).

Class Components: These types of components can hold and manage their own state and have a separate render method to return JSX on the screen. They are also called Stateful components as they can have a state.

9. What is a state in React?

The state is a built-in React object that is used to contain data or information about the component. The state in a component can change over time, and whenever it changes, the component re-renders.

The change in state can happen as a response to user action or system-generated events. It determines the behavior of the component and how it will render.

10. What are props in React?

Props are short for Properties. It is a React built-in object that stores the value of attributes of a tag and works similarly to HTML attributes.

Props provide a way to pass data from one component to another component. Props are passed to the component in the same way as arguments are passed in a function.

11. What is a higher-order component in React?

A higher-order component acts as a container for other components. This helps to keep components simple and enables re-usability. They are generally used when multiple components have to use a common logic.

12. Explain the lifecycle methods of components

getInitialState(): This is executed before the creation of the component.

componentDidMount(): Is executed when the component gets rendered and placed on the DOM.

shouldComponentUpdate(): Is invoked when a component determines changes to the DOM and returns a “true” or “false” value based on certain conditions.

componentDidUpdate(): Is invoked immediately after rendering takes place.

componentWillUnmount(): Is invoked immediately before a component is destroyed and unmounted permanently.

So far, if you have any doubts about the above React interview questions and answers, please ask your questions in the section below

13. What is Redux?

Redux is an open-source, JavaScript library used to manage the application state. React uses Redux to build the user interface. It is a predictable state container for JavaScript applications and is used for the entire application’s state management.

14. What are the components of Redux?

Store: Holds the state of the application.

Action: The source information for the store.

Reducer: Specifies how the application's state changes in response to actions sent to the store.

15. What is the Flux

Flux is the application architecture that Facebook uses for building web applications. It is a method of handling complex data inside a client-side application and manages how data flows in a React application.

There is a single source of data (the store) and triggering certain actions is the only way way to update them.The actions call the dispatcher, and then the store is triggered and updated with their own data accordingly.

When a dispatch has been triggered, and the store updates, it will emit a change event that the views can rerender accordingly.

16. What is React Router?

React Router is a routing library built on top of React, which is used to create routes in a React application.

17. Why do we need to React Router?

It maintains consistent structure and behavior and is used to develop single-page web applications.

Enables multiple views in a single application by defining multiple routes in the React application.

* List some of the major advantages of React.

Some of the major advantages of React are:

It increases the application’s performance

It can be conveniently used on the client as well as server side

Because of JSX, code’s readability increases

React is easy to integrate with other frameworks like Meteor, Angular, etc

Using React, writing UI test cases become extremely easy

* What are the limitations of React?

Limitations of React are listed below:

React is just a library, not a full-blown framework

Its library is very large and takes time to understand

It can be little difficult for the novice programmers to understand

Coding gets complex as it uses inline templating and JSX

- NextJS

Advantages:

Easy installation, project build, modification, and required package found.

Optimal application performance due to the availability of automatic code splitting

Next JS allows optimized code bundles to be loaded lazily behind the scenes with the help of prefetching

It allows application code to use SSR or Server Side Rendering, thus offering SEO friendly flexibility, initial render to application view, and elimination of code download

Effective Hot-Module Replacement and powerful error reporting

- Javascript

Advantages:

1. Speed

Since JavaScript is an ‘interpreted’ language, it reduces the time required by other programming languages like Java for compilation. JavaScript is also a client-side script, speeding up the execution of the program as it saves the time required to connect to the server.

2. Simplicity

JavaScript is easy to understand and learn. The structure is simple for the users as well as the developers. It is also very feasible to implement, saving developers a lot of money for developing dynamic content for the web.

3. Popularity

Since all modern browsers support JavaScript, it is seen almost everywhere. All the famous companies use JavaScript as a tool including Google, Amazon, PayPal, etc.

4. Interoperability

JavaScript works perfect with other programming languages and therefore numerous developers prefer it in developing many applications. We can embed it into any webpage or inside the script of another programming language.

5. Server Load

As JavaScript operates on the client-side, data validation is possible on the browser itself rather than sending it off to the server. In case of any discrepancy, the whole website needs not to be reloaded. The browser updates only the selected segment of the page.

6. Rich Interfaces

JavaScript provides various interfaces to developers for creating catchy webpages. Drag and drop components or sliders may give a rich interface to the webpages. This leads to improved user-interactivity on the webpage.

7. Extended Functionality

Third-party add-ons like Greasemonkey (a Mozilla Firefox extension) allow the developers to add snippets of predefined code in their code to save time and money. These add-ons help the developers build JavaScript applications a lot faster and with much more ease than other programming languages.

8. Versatility

JavaScript is now capable of front-end as well as back-end development. Back-end development uses NodeJS while many libraries help in front-end development like AngularJS, ReactJS, etc.

9. Less Overhead

JavaScript improves the performance of websites and web applications by reducing the code length. The codes contain less overhead with the use of various built-in functions for loops, DOM access, etc.

How to use componentWillUnmount with Functional Components in React

Functional Components: Functional components are some of the more common components that will come across while working in React

Class Components: A class component requires you to extend from React. Component and create a render function which returns a React element.

There is no render method used in functional components.

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

function Example() {

const [count, setCount] = useState(0);

useEffect(() => {

document.title = `You clicked ${count} times`;

});

return (

<div>

<p> You clicked {count} times</p>

<button onClick={() => setCount(count + 1)}>

Click me

</button>

</div>

);

}

Class Component Example

import React, { Component } from "react";

class ClassComponent extends React.Component{

constructor(){

super();

this.state={

count :0

};

this.increase=this.increase.bind(this);

}

increase(){

this.setState({count : this.state.count +1});

}

render(){

return (

<div style={{margin:'50px'}}>

<h1>Welcome to Geeks for Geeks </h1>

<h3>Counter App using Class Component : </h3>

<h2> {this.state.count}</h2>

<button onClick={this.increase}> Add</button>

</div>

)

}

}

export default ClassComponent;

What is Provider Component?

The <Provider> components makes the Redux store available to any nested components that need to access the Redux store.

Since any React component in a React Redux app can be connected to the store, most applications will render a <Provider> at the top level, with the entire app’s component tree inside of it.

The Hooks and connect APIs can then access the provided store instance via React's Context mechanism.

1. HTML + CSS

No front-end dev is a stranger to HTML and CSS. The ability to work with and craft user interfaces is necessary to every organization. At a high level, React developers should be able to:

Work with and write semantic HTML tags

Work with and write CSS selectors

Implement a CSS reset

Understand the box model and how to reset to border-box

Understand flexbox

Work with and implement responsive web principles including the proper user of media queries

2. JSX

In React, you never really touch HTML proper. You work with a syntax extension that is truly one of the most remarkable parts of the React ecosystem: JSX. JSX looks so much like HTML you may think of it as HTML-flavored JavaScript. What’s cool about JSX is that if you know HTML and CSS, you intuitively know how to work with JSX.

JSX is an abstraction on top of the React.createElement() API. One reason it is vital to the library—and why the React team chose to go with it in the first place—is that the API would be too cumbersome to use in terms of scaling. One potentially could use React.createElement() to build out an entire application, however this wouldn’t be any more efficient than just using HTML proper. It may feel at first that we’ve taken a step backward by adding our Markup into our template logic, however a few quick minutes with JSX and you’ll be hooked on the style.

3. JavaScript Fundamentals + ES6

You can’t rock React without a firm understanding of the fundamental concepts that the JavaScript language provides, but these ES6 skills are also essential:

Variables and scoping

Understanding when and where you have access to the data you need is critical. Variables are the mechanism built into JavaScript that allow us to hold onto data in memory and access that data later on within our applications.

ES6 brought with it new keywords that we can use to store variables other than the traditional `var` keyword (like `let` and `const`). You may choose to live by the principle that unless you absolutely need `var`, use `const` until your linter tells you otherwise, then default to `let`.

Arrays and objects

React suggests a pattern that your `view is a function of your state`. Arithmetically put, that’s `v = f(s)`, and something you need to remember as you revisit your skills in the foundations of the library.

State is data that we want to show to users or items in memory that we can access in order to allow our users to interact with our data. We hold all of the data that we present on an object called state and access these bits of data via properties on this state object. This is how React received its name; when state changes, the view updates (`v = f(s);`). So your view ‘reacts’ to the changes that are made in your state object.

You should brush up on how to mutate objects and change values of properties on them. Don’t worry, React takes care of the mechanism through a nifty function called `setState()` to make this work to your advantage.

Array methods

It’s one thing to be able to store data and access it within arrays and objects. It’s another to be able to manipulate that data properly. The built-in JavaScript array methods are essential tools in every developer’s toolbox. Focus in on `.map`, `.filter` and `.reduce` for maximum impact.

Functions and arrow functions

In React, every single component you build is a function in one way or another. Remember that ‘classes’ are just `constructor functions` under the hood. Regardless of the syntax you’re using, when building `functional components` or `class components` you’re using some form of a function.

Don’t underestimate the importance of these fundamentals. Many practices out there today that lend themselves to functional programming. The chance to use JavaScript functions to build out small chunks of UI is like building a Lego set without instructions. Each piece of UI is an encapsulated function that contains the state data your elements need to present, the elements themselves and the formal component logic you need to use that logic. Each component is a Lego brick, and it’s up to you to fit them all together.

DOM Manipulation and event handlers

In React, manipulating the actual DOM elements is rare. Remember we now have the JSX abstraction at our disposal. The native event object that you get with normal DOM manipulation in React is actually wrapped up in what’s called the SyntheticEvent. Make sure you can attach different types of events to HTML elements such as `onclicks`, `onchange`, `mouseenter`, etc.

The “this” keyword

The ‘this’ keyword is one of the commonly misused features in JavaScript. Think of ‘this’ as a pointer to an object. For example, you can use the ‘this’ keyword to reference an object without having to refer to that object’s name.

Higher order functions and callback functions

The idea that functions can be passed around as arguments—in the case of high order functions and callbacks—is what drives the `input/output` model of functional programming.

You pass handlers around everywhere in React. Most of the time the handlers you pass around are in the form of methods that are chained onto an object and accessed as properties, which will be bound up in the prototype chain. However, there are moments that you need to reach outside of the React component paradigm or create a few different types of components that extend some of the functionality to one another. These patterns are commonly referred to as advanced React patterns and they’re finding their way into the better/common practices realm. React will push you to be innovative and creative as you scale along with it.

Prototypal inheritance and object creation

React lends itself to a functional programming paradigm in many aspects. However, you work in the world of classes and, as a result, a world of object creation. If you understand the basics of how the prototype chain works in JavaScript, you’ll know what you need to about how we achieve inheritance in JavaScript. Remember, that there are no traditional classes in JavaScript. The class keyword is just syntactic sugar on top of the `object prototype` chain in JavaScript.

The ‘class’ keyword

JavaScript classes aren’t the same as classes in a traditional programming sense. You create classes that encapsulate your template logic, formal JavaScript logic and even styles known as components. These components are the building blocks of any React application, and there are only two ways to write the basic component: either by declaring it as a function or declaring it as a class.

Assess your proficiency with classes by making sure you can answer:

How do I set up methods on a class?

How do I bind those methods together?

How do I access properties that are found on the constructor?

How do I create objects that would be considered ‘children’ of parent objects?

4. Git

Git is essential to every developer's toolkit for storing projects on solutions like GitHub, Bitbucket and GitLab. Skills that should just be part of your day-to-day include:

Tracking changes with add, commit, push and pull

Branching and merging strategies

Handling merge conflicts

5. Node + npm

Node may be a surprise to many. Why would you need to know how to work with Node in order to be a client-side React dev? While you can pull React into any HTML file, there will be many other packages out there that will allow you to extend the React library.

React developers need to have a solid understanding of the npm registry. This is the place where software developers can go to get software to help them build software. Sounds funny, but truly that’s all the npm is: a cloud storage for packages we call dependencies.

Yarn vs npm

Yarn is a package manager that is built to utilize the npm registry. Yarn actually optimizes your npm workflows. Yarn and npm somewhat compete today, but the mission of Yarn has been to solve a lot of the problems that are accepted in the Node/npm ecosystem. npm has been doing everything it can to follow the patterns and practices that Yarn presents.

6. Redux

Redux is a way to manage the “state” or we can say it is a cache or storage that can be accessed by all components with a structured way. It has to be accessed through a “Reducer” and “Actions”

5. Callbacks

In javascript, a callback is simply a function that is passed to another function as a parameter and is invoked or executed inside the other function. Here a function needs to wait for another function to execute or return value and this makes the chain of the functionalities (when X is completed, then Y executed, and it goes on.). This is the reason callback is generally used in the asynchronous operation of javascript to provide the synchronous capability.

- Promises

Promises are useful in asynchronous javascript operation when we need to execute two or more back to back operations (or chaining callback), where each subsequent function starts when the previous one is completed. A promise is an object that may produce a single value sometime in the future, either a resolved value or a reason that it’s not resolved (rejected).

- Async & Await

Stop and wait until something is resolved. Async & await just syntactic sugar on top of Promises and like promises it also provides a way to maintain asynchronous operation more synchronously. So in javascript asynchronous operation can be handled in various versions…

ES5 -> Callback

ES6 -> Promise

ES7 -> async & await

const showPosts = async () => {

const response = await fetch('https://jsonplaceholder.typicode.com/posts');

const posts = await response.json();

console.log(posts);

}

showPosts();

- Context

Context is designed to share data that can be considered “global” for a tree of React components, such as the current authenticated user, theme, or preferred language

const ThemeContext = React.createContext('light');

class App extends React.Component {

render() {

// Use a Provider to pass the current theme to the tree below.

// Any component can read it, no matter how deep it is.

// In this example, we're passing "dark" as the current value.

return (

<ThemeContext.Provider value="dark">

<Toolbar />

</ThemeContext.Provider>

);

}

}

// A component in the middle doesn't have to

// pass the theme down explicitly anymore.

function Toolbar() {

return (

<div>

<ThemedButton />

</div>

);

}

class ThemedButton extends React.Component {

// Assign a contextType to read the current theme context.

// React will find the closest theme Provider above and use its value.

// In this example, the current theme is "dark".

static contextType = ThemeContext;

render() {

return <Button theme={this.context} />;

}

}

- Ref

Refs are created using React.createRef() and attached to React elements via the ref attribute. Refs are commonly assigned to an instance property when a component is constructed so they can be referenced throughout the component.

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.myRef = React.createRef();

}

render() {

return <div ref={this.myRef} />;

}

}

Accessing Refs

When a ref is passed to an element in render, a reference to the node becomes accessible at the current attribute of the ref.

const node = this.myRef.current;

The value of the ref differs depending on the type of the node:

. When the ref attribute is used on an HTML element, the ref created in the constructor with React.createRef() receives the underlying DOM element as its current property.

. When the ref attribute is used on a custom class component, the ref object receives the mounted instance of the component as its current.

. You may not use the ref attribute on function components because they don’t have instances.

- PureFunction

Pure Function is a function (a block of code ) that always returns the same result if the same arguments are passed.

function pureFunction(param) {

return param + 1

}

- TypeScript

TypeScript is a super set of JavaScript.

TypeScript builds on top of JavaScript. First, you write the TypeScript code. Then, you compile the TypeScript code into plain JavaScript code using a TypeScript compiler

There are two main reasons to use TypeScript:

TypeScript adds a type system to help you avoid many problems with dynamic types in JavaScript.

TypeScript implements the future features of JavaScript a.k.a ES Next so that you can use them today.

- Component Lifecycle

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()

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()

Unmounting

This method is called when a component is being removed from the DOM:

componentWillUnmount()

- Closures

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time.

1. What is JavaScript?

JavaScript is a client-side and server-side scripting language inserted into HTML pages and is understood by web browsers. JavaScript is also an Object-based Programming language

3. What are JavaScript Data Types?

Following are the JavaScript Data types:

Number

String

Boolean

Object

Undefined

10. Write the code for adding new elements dynamically?

<html>

<head>

<title>t1</title>

<script type="text/javascript">

function addNode () { var newP = document. createElement("p");

var textNode = document.createTextNode(" This is a new text node");

newP.appendChild(textNode); document.getElementById("firstP").appendChild(newP); }

</script> </head>

<body> <p id="firstP">firstP<p> </body>

</html>

A JavaScript callback is a function which is to be executed after another function has finished execution.

A more formal definition would be - Any function that is passed as an argument to another function so that it can be executed in that other function is called as a callback function.

The Promise object represents the eventual completion (or failure) of an asynchronous operation and its resulting value.

Advantages of MongoDB over RDBMS

• Schema less − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.

• Structure of a single object is clear.

• No complex joins.

• Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.

• Tuning.

• Ease of scale-out − MongoDB is easy to scale.

• Conversion/mapping of application objects to database objects not needed.

• Uses internal memory for storing the (windowed) working set, enabling faster access of data.

Why Use MongoDB?

• Document Oriented Storage − Data is stored in the form of JSON style documents.

• Index on any attribute

• Replication and high availability

• Auto-Sharding

• Rich queries

• Fast in-place updates

• Professional support by MongoDB

Where to Use MongoDB?

• Big Data

• Content Management and Delivery

• Mobile and Social Infrastructure

• User Data Management

• Data Hub

* **Clouser**

A closure is the combination of a function bundled together (enclosed) with references to its surrounding state (the lexical environment). In other words, a closure gives you access to an outer function's scope from an inner function. In JavaScript, closures are created every time a function is created, at function creation time

* **Generator Function In Javascript**

A generator is a function that can stop midway and then continue from where it stopped.

Generators are functions that can be exited and later re-entered.

Their context (variable bindings) will be saved across re-entrances

Calling a generator function does not execute its body immediately;

an iterator object for the function is returned instead.

When the iterator's next() method is called, the generator function's body is executed until the first yield expression, which specifies the value to be returned from the iterator or, with yield\*, delegates to another generator function.

The next() method returns an object with a value property containing the yielded value and a done property which indicates whether the generator has yielded its last value, as a boolean.

Calling the next() method with an argument will resume the generator function execution, replacing the yield expression where an execution was paused with the argument from next().

A return statement in a generator, when executed, will make the generator finish (i.e. the done property of the object returned by it will be set to true).

If a value is returned, it will be set as the value property of the object returned by the generator.

Much like a return statement, an error thrown inside the generator will make the generator finished -- unless caught within the generator's body.

When a generator is finished, subsequent next() calls will not execute any of that generator's code, they will just return an object of this form: {value: undefined, done: true}.

* **What is Memoization in Javascript**

Memoization is a top-down, depth-first, optimization technique of storing previously executed computations. Whenever the program needs the result of these computations, the program will not have to execute that computation again. Instead, it will reuse the result of the previously executed computation. This way the program will not have to repeat expensive computations. An expensive function is a function that takes some time to execute.

## The "use strict" Directive

The purpose of "use strict" is to indicate that the code should be executed in "strict mode".

With strict mode, you can not, for example, use undeclared variables.

Hoisting is JavaScript's default behavior of moving all declarations to the top of the current scope

WebPack Plugins are the backbone of webpack. Webpack itself is built on the same plugin system that you use in your webpack configuration!

|  |  |
| --- | --- |
| [BannerPlugin](https://webpack.js.org/plugins/banner-plugin) | Add a banner to the top of each generated chunk |
| [CommonsChunkPlugin](https://webpack.js.org/plugins/commons-chunk-plugin) | Extract common modules shared between chunks |
| [CompressionWebpackPlugin](https://webpack.js.org/plugins/compression-webpack-plugin) | Prepare compressed versions of assets to serve them with Content-Encoding |
| [ContextReplacementPlugin](https://webpack.js.org/plugins/context-replacement-plugin) | Override the inferred context of a require expression |
| [CopyWebpackPlugin](https://webpack.js.org/plugins/copy-webpack-plugin) | Copies individual files or entire directories to the build directory |
| [DefinePlugin](https://webpack.js.org/plugins/define-plugin) | Allow global constants configured at compile time |
| [DllPlugin](https://webpack.js.org/plugins/dll-plugin) | Split bundles in order to drastically improve build time |
| [EnvironmentPlugin](https://webpack.js.org/plugins/environment-plugin) | Shorthand for using the [DefinePlugin](https://webpack.js.org/plugins/define-plugin) on process.env keys |
| [EslintWebpackPlugin](https://webpack.js.org/plugins/eslint-webpack-plugin) | A ESLint plugin for webpack |
| [HotModuleReplacementPlugin](https://webpack.js.org/plugins/hot-module-replacement-plugin) | Enable Hot Module Replacement (HMR) |
| [HtmlWebpackPlugin](https://webpack.js.org/plugins/html-webpack-plugin) | Easily create HTML files to serve your bundles |
| [IgnorePlugin](https://webpack.js.org/plugins/ignore-plugin) | Exclude certain modules from bundles |
| [LimitChunkCountPlugin](https://webpack.js.org/plugins/limit-chunk-count-plugin) | Set min/max limits for chunking to better control chunking |
| [MinChunkSizePlugin](https://webpack.js.org/plugins/min-chunk-size-plugin) | Keep chunk size above the specified limit |
| [MiniCssExtractPlugin](https://webpack.js.org/plugins/mini-css-extract-plugin) | creates a CSS file per JS file which requires CSS |
| [NoEmitOnErrorsPlugin](https://webpack.js.org/configuration/optimization/#optimizationemitonerrors) | Skip the emitting phase when there are compilation errors |
| [NormalModuleReplacementPlugin](https://webpack.js.org/plugins/normal-module-replacement-plugin) | Replace resource(s) that matches a regexp |
| [NpmInstallWebpackPlugin](https://webpack.js.org/plugins/install-webpack-plugin) | Auto-install missing dependencies during development |
| [ProgressPlugin](https://webpack.js.org/plugins/progress-plugin) | Report compilation progress |
| [ProvidePlugin](https://webpack.js.org/plugins/provide-plugin) | Use modules without having to use import/require |
| [SourceMapDevToolPlugin](https://webpack.js.org/plugins/source-map-dev-tool-plugin) | Enables a more fine grained control of source maps |
| [EvalSourceMapDevToolPlugin](https://webpack.js.org/plugins/eval-source-map-dev-tool-plugin) | Enables a more fine grained control of eval source maps |
| [TerserPlugin](https://webpack.js.org/plugins/terser-webpack-plugin/) | Uses Terser to minify the JS in your project |

**PUT vs PATCH**

PUT is a method of modifying resource where the client sends data that updates the entire resource. It is used to set an entity’s information completely. PUT is similar to POST in that it can create resources, but it does so when there is a defined URI. PUT overwrites the entire entity if it already exists, and creates a new resource if it doesn’t exist.

For example, when you want to change the first name of a person in a database, you need to send the entire resource when making a PUT request.

{“first": "John", "last": "Stone”}

To make a PUT request, you need to send the two parameters; the first and the last name.

Unlike PUT, PATCH applies a partial update to the resource.

This means that you are only required to send the data that you want to update, and it won’t affect or change anything else. So if you want to update the first name on a database, you will only be required to send the first parameter; the first name.

**What is the difference between HTTP and HTTPS?**

* **Protocol**

The HTTP protocol stands for Hypertext Transfer Protocol, whereas the HTTPS stands for Hypertext Transfer Protocol Secure.

* **Security**

The HTTP protocol is not secure protocol as it does not contain SSL (Secure Sockets Layer), which means that the data can be stolen when the data is transmitted from the client to the server. Whereas, the HTTPS protocol contains the SSL certificate that converts the data into an encrypted form, so no data can be stolen in this case as outsiders do not understand the encrypted text.

* **Port numbers**

The HTTP transmits the data over port number 80, whereas the HTTPS transmits the data over 443 port number. Under the documentation issued by Tim Berners-Lee, he stated that "if the port number is not specified, then it will be considered as HTTP".

When RFC 1340 was announced, then the IETF (Internet Engineering Task Force) provided port number 80 to the HTTP. When the new RFC was released in the year 1994, the HTTPS is assigned with a port number 443.

* **Layers**

The HTTP protocol works on the application layer while the HTTPS protocol works on the transport layer. As we know that the responsibility of the transport layer is to move the data from the client to the server, and data security is a major concern. HTTPS operates in the transport layer, so it is wrapped with a security layer.

* **SSL Certificates**

When we want our websites to have an HTTPS protocol, then we need to install the signed SSL certificate. The SSL certificates can be available for both free and paid service. The service can be chosen based on business needs.

The HTTP does not contain any SSL certificates, so it does not decrypt the data, and the data is sent in the form of plain text.

* **SEO Advantages**

The SEO advantages are provided to those websites that use HTTPS as GOOGLE gives the preferences to those websites that use HTTPS rather than the websites that use HTTP.

* **Online Transactions**

If we are running an online business, then it becomes necessary to have HTTPS. If we do not use the HTTPS in an online business, then the customers would not purchase as they are scared that their data can be stolen by the outsiders.

**Why Choose Tailwind CSS?**

Tailwind CSS’s [official documentation](https://tailwindcss.com/) describes it as a “utility-first CSS framework” that comes with classes equipped to build custom UI designs directly in the users’ markup. It is handy to implement inline styling to rustle up a stunning UI without writing any CSS.

* **Highly Customizable:**Tailwind CSS comes with a default configuration, but it can be overridden with a **tailwind.config.js** file. This enables easy customization of styling, themes, spacing, palettes, etc. Use Tailwind’s utilities to enable easy project management and develop a website that delights customers.
* **Carries Commonly used Utility Pattern:** With Tailwind CSS, users can cut down on having to name too many classes. It comes with common utility patterns to deal with standard requirements: specifying and organizing classes, cascading classes, etc. In simple terms, creating custom components become so much easier. Instead of hard-coding, just use the theme() function to derive values from configuration files.
* **PurgeCSS Optimization:**[PurgeCSS](https://purgecss.com/) reduces file size by scanning HTML code and eliminating unused classes. In combination with Tailwind CSS, this is particularly useful. As a project expands, so does the size of accompanying CSS files. Optimizing via PurgeCSS reduces and cleans up CSS file size, making it infinitely more manageable, especially before deployment.
* **Responsiveness Made Easy:** By default, Tailwind CSS utilizes a mobile-first approach. To quote the [website](https://tailwindcss.com/docs/responsive-design), “*Every utility class in Tailwind can be applied conditionally at different breakpoints, which makes it a piece of cake to build complex responsive interfaces without ever leaving your HTML.*”Utility classes simplify the creation of complex responsive layouts but allowing them to be used across [multiple breakpoints](https://www.browserstack.com/guide/responsive-design-breakpoints). The result? Hassle-free implementation of responsive design.
* **Effortless Community Interaction**: Tailwind’s active community is an excellent place for users to get their questions answered, especially when dealing with stubborn issues. Get CSS-related queries solved and create excellent websites faster.

**Why Choose Bootstrap?**  
[Bootstrap](https://getbootstrap.com/) is an open-source framework containing CSS and JavaScript-based templates for interface components.

* **Massive Ecosystem:**Among front-end frameworks, Bootstrap’s ecosystem is unmatched. It offers, out of the box, a vast library of layouts, themes, UI elements, panels, modals, buttons, alerts, cards, etc., that devs and designers can choose from and implement. Additionally, Bootstrap is backed by best-in-industry community support.
* **Accelerated prototyping:**When using Bootstrap, designers can just write out their HTML code, include the relevant CSS classes, and achieve website responsiveness. They don’t have to spend time adjusting for browser incompatibility, CSS positioning, and the like.
* **Twitter Support**: Unsurprisingly, when a significant commercial player backs an open-source project, users can be assured that it is here to stay and carries high confidence among people who know the industry. The fact that Bootstrap grew out of, and is backed by Twitter, establishes its efficacy.
* **Supports SASS and LESS**: Although most developers don’t use LESS, significant projects rely on it. Obviously, SASS support is also highly desirable. Not too many CSS frameworks other than Bootstrap support both.

**Define SASS?**

SASS means Syntactically Awesome Style sheets. It is a CSS preprocessor which is used to reduce repetition with CSS and save time. It adds power and elegance to the basic language and facilitates you to add variables, nested rules, mixins, inline imports, inheritance and more, all with fully CSS-compatible syntax.

What are the most significant advantages of using GraphQL over REST?

A list of the most significant advantages of using GraphQL over REST:

* There is only one endpoint in GraphQL, but REST has multiple endpoints. That's why GraphQL is more cost-effective than REST. You don't have to use your resources for various endpoints.
* In REST API, you have to use multiple requests to retrieve a complex data-set, but in GraphQL, you can execute a complex query easily in just a single request.
* You can change the request data format in GraphQL, but it is not possible to do the same in REST.
* The development speed in GraphQL is faster than REST.
* GraphQL provides high consistency across all platforms, while In REST, it is hard to get consistency across all platforms.
* GraphQL doesn't support an automatic caching system, while REST uses caching automatically.

### 25) What is Apollo in GraphQL?

Apollo is a platform for the implementation of GraphQL. As we know that GraphQL is a query language, so in order to use this query language easily, we need a platform, Apollo provides that platform.

Apollo provides two open-sourced libraries to create client and server. Here, the client is used to fetch data from a GraphQL server, and the server is used to create an API for GraphQL client.

### 26) What do you know by Mutation in GraphQL?

Mutation is one of the most important operations in GraphQL. It is used for write operation when you want to add delete and edit data.

### 27) What is subscription in GraphQL?

In GraphQL, the subscription is used for listening for any data changes. The server sends a notification message to the client after any data changes, if the client is subscribed to that event.

### 28) What do you mean by a Query in GraphQL?

A GraphQL query is used to read data. It is similar to the GET request we use in REST APIs. The GraphQL queries are used to retrieve data from the GraphQL server.

**What does useEffect do?**

By using this Hook, you tell React that your component needs to do something after render. React will remember the function you passed (we’ll refer to it as our “effect”), and call it later after performing the DOM updates. In this effect, we set the document title, but we could also perform data fetching or call some other imperative API.

**Why is useEffect called inside a component?**

Placing useEffect inside the component lets us access the count state variable (or any props) right from the effect. We don’t need a special API to read it — it’s already in the function scope. Hooks embrace JavaScript closures and avoid introducing React-specific APIs where JavaScript already provides a solution.

**Does useEffect run after every render?**

Yes! By default, it runs both after the first render and after every update. Instead of thinking in terms of “mounting” and “updating”, you might find it easier to think that effects happen “after render”. React guarantees the DOM has been updated by the time it runs the effects.

## Private Key

The private key is used to both encrypt and decrypt the data. This key is shared between the sender and receiver of the encrypted sensitive information. The private key is also called symmetric being common for both parties. Private key cryptography is faster than public-key cryptography mechanism.

## Public Key

The public key is used to encrypt and a private key is used decrypt the data. The private key is shared between the sender and receiver of the encrypted sensitive information. The public key is also called asymmetric cryptography.

Public and private keys can also be used to create a digital signature. A digital signature assures that the person sending the message is who they claim to be.

Typically, we use the recipient’s public key to encrypt the data and the recipient then uses their private key to decrypt the data. However, using the scheme of digital signatures, there’s no way to authenticate the source of the message. Mike could get a hold of Alice’s public key (since it’s public) and pretend that Bob is the person sending a message to Alice.

To create a digital signature, Bob digitally signs his email to Alice using his private key. When Alice receives the message from Bob, she can verify the digital signature on the message came from Bob by using his public key. As the digital signature uses Bob’s private key, Bob is the only person who could create the signature.

## ****What Is a REST API?****

REST is basically an architectural style of the web services that work as a channel of communication between different computers or systems on the internet. The term REST API is something else.

Those application programming interfaces that are backed by the architectural style of REST architectural system are called REST APIs. REST API compliant web services, database systems, and computer systems permit requesting systems to get robust access and redefine representations of web based resources by deploying a predefined set of stateless protocols and standard operations.

By these protocols and operations and redeploying the manageable and updatable components without causing the effect on the system, REST API systems deliver fast performance, reliability, and more progression.

## ****What Is a SOAP API?****

SOAP is a standard communication protocol system that permits processes using different operating systems like Linux and Windows to communicate via [HTTP](https://dzone.com/articles/the-http-series-part-1-overview-of-the-basic-conce) and its [XML](https://dzone.com/articles/writing-and-reading-xml-file). SOAP based APIs are designed to create, recover, update and delete records like accounts, passwords, leads, and custom objects.

These offers over twenty different kinds of calls that make it easy for the API developers to maintain their accounts, perform accurate searches and much more. These can then be used with all those languages that support web services.

SOAP APIs take the advantages of making web based protocols such as HTTP and its XML that are already operating the all operating systems that are why its developers can easily manipulate web services and get responses without caring about language and platforms at all.

### ****Differences:****

* REST API has no has no official standard at all because it is an architectural style. SOAP API, on the other hand, has an official standard because it is a protocol.
* REST APIs uses multiple standards like HTTP, JSON, URL, and XML while SOAP APIs is largely based on HTTP and XML.
* As REST API deploys multiple standards, so it takes fewer resources and bandwidth as compared to SOAP that uses XML for the creation of Payload and results in the large sized file.
* The ways both APIs exposes the business logics are also different. REST API takes advantage of URL exposure like @path("/WeatherService") while SOAP API use of services interfaces like @WebService.
* SOAP API defines too many standards, and its implementer implements the things in a standard way only. In the case of miscommunication from service, the result will be the error. REST API, on the other hand, don't make emphasis on too many standards and results in corrupt API in the end.
* REST API uses Web Application Description Language, and SOAP API used Web Services Description language for describing the functionalities being offered by web services.
* REST APIs are more convenient with JavaScript and can be implemented easily as well. SOAP APIs are also convenient with JavaScript but don't support for greater implementation.

**What is Destructuring in Javascript?**

The destructuring assignment syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.