1. What is ReactJs?

React is an **open-source frontend JavaScript library** which is used for building Single page applications user interfaces.

1. What are the features/advantages of ReactJs?
   1. Uses JSX – single document consists both javascript and HTML code
   2. Virtual DOM – Improves performance
   3. Has reusable components - Components are independent and reusable bits of code. These components can be shared across various applications having similar functionality. The re-use of components increases the pace of development.
2. What is SPA (Single-page application) & MPA(Multi-page application)?

An SPA is a web app implementation that loads only a single web document, and then updates the body content of that single document via JavaScript APIs. Therefore, allows users to use websites without loading whole new pages from the server, SPAs don't need to load each new web page from scratch

Whereas, Multi-page applications renders new page from server every time. Every change eg. display the data or submit data back to server requests rendering a new page from the server in the browser.

SPA is faster than MPA

1. Disadvantages of SPA-
   1. It does not support legacy browser eg ie8
   2. Efforts require to implement navigation and maintain state
2. What is JSX?
   1. JSX stands for JavaScript XML.
   2. It allows us to write HTML inside JavaScript and place them in the DOM without using functions like appendChild( ) or createElement( ).
3. What is difference between real and virtual DOM?

|  |  |
| --- | --- |
| **Real DOM** | **Virtual  DOM** |
| 1. It updates slow. | 1. It updates faster. |
| 2. Can directly update HTML. | 2. Can’t directly update HTML. |
| 3. Creates a new DOM if element updates. | 3. Updates the JSX if element updates. |
| 4. DOM manipulation is very expensive. | 4. DOM manipulation is very easy. |
| 5. Too much of memory wastage. | 5. No memory wastage. |

1. How react work?

Instead of manipulating the browser's DOM directly, React creates a virtual DOM in memory, where it does all the necessary manipulating, before making the changes in the browser DOM. React finds out what changes have been made, and changes **only** what needs to be changed.

1. When to use a Class Component over a Function Component?

If the component needs *state or lifecycle methods* then use class component otherwise use function component. *However, from React 16.8 with the addition of Hooks, you could use state , lifecycle  methods and other features that were only available in class component right in your function component.* So, it is always recommended to use Function components, unless you need a React functionality whose Function component equivalent is not present yet, like Error Boundaries

1. **Functional Components vs Class Components:**
   1. **Functional component:** These are nothing but simple JavaScript functions

const Car = () => {

return <h2>Hi</h2>

}

* 1. **Class components:** These components are just like simple class and must have render function.

class Car extends React.Component {

render() {    
     return <h2>Hi, I am a Car!</h2>;

  } }

|  |  |
| --- | --- |
| **Functional Components** | **Class Components** |
| A functional component is just a plain JavaScript function that accepts props as an argument and returns a React element. | 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. | It must have the render method returning HTML |
| Also known as Stateless components as they simply accept data and display them in some form, that they are mainly responsible for rendering UI. | Also known as Stateful components because they implement logic and state. |
| React lifecycle methods (for example, componentDidMount) cannot be used in functional components. | React lifecycle methods can be used inside class components (for example, componentDidMount). |
| Hooks can be easily used in functional components.  example: const [name,SetName]= React.useState(‘ ‘) | It requires different syntax inside a class component to implement hooks.  example: constructor(props) {     super(props);     this.state = {  name: “ ”  } } |
| Constructors are not used | Constructor are used as it needs to store state |

1. When does react components re-renders?
   1. “setState” is called in Component
   2. “props” values are updated
   3. This.forceUpdate() is called
2. How to inherit class?

By using extends keyword. By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

Eg: class Model extends Care

1. What is Arrow function?
   1. Arrow functions allow us to write shorter function syntax .

Before: hello = function() {  return "Hello World!";}

After: hello = () => {  return "Hello World!";}

* 1. With arrow function, no binding required while with normal function, we need to bind function with this keyword

This.hello=this.hello.bind(this)

1. What is stateful and stateless component?

|  |  |
| --- | --- |
| **Stateful Component** | **Stateless Component** |
| 1. Stores info about component’s state change in memory | 1. Calculates the internal state of the components |
| 2. Have authority to change state | 2. Do not have the authority to change state |
| 3. Contains the knowledge of past, current and possible future changes in state | 3. Contains no knowledge of past, current and possible future state changes |
| 4. Stateless components notify them about the requirement of the state change, then they send down the props to them. | 4. They receive the props from the Stateful components and treat them as callback functions. |

1. How to prevent re-renders in React?

By using ShouldComponentUpdate() method. Use shouldComponentUpdate( ) method only when you are sure that it’s a **static**component.

shouldComponentUpdate() {

  return false;

}

1. What is the significance of keys in React?

A key is a special string attribute you **should** include when creating arrays of elements. *Key* prop helps React identify which items have changed, are added, or are removed. Keys are used for identifying unique Virtual DOM Elements with their corresponding data driving the UI. They help React to optimize the rendering by recycling all the existing elements in the DOM. These keys must be a unique number or string, using which React just reorders the elements instead of re-rendering them. This leads to increase in application’s performance.

* 1. Using *indexes* for *keys* is **not recommended** if the order of items may change. This can negatively impact performance and may cause issues with component state. Better not to use index as key because anytime array can get re-order

1. What is variable?

Variable is in which the value is stored

there are three ways of defining your variables: var, let, and const

1. Var a=5;
   1. If you use var outside of a function, it belongs to the global scope.
   2. If you use var inside of a function, it belongs to that function.
   3. If you use var inside of a block, i.e. a for loop, the variable is still available outside of that block.
   4. var has a ***function* scope, not a *block* scope.**
2. Let a=5;
   1. let has a ***block* scope**
   2. let is limited to the block (or expression) where it is defined.
   3. If you use let inside of a block, i.e. a for loop, the variable is only available inside of that loop.
3. Const a=5;    
   a. const is a variable that once it has been created, its value can never change.    
   b. const has a ***block* scope.**

1. What is Shallow Comparison?

The values saved in the variable can be either a primitive or reference type.

Example **“var a = 10”,**here the value saved in the variable “a” is of primitive type.

 The data stored in Objects and Array can be referred to as Reference type data.**Comparing Primitive Values is not a concern, problems arise when we have reference values during the comparison.**

When we compare two different objects with the same properties, they equate to false. JavaScript looks for the object reference (Starting Address of the Object). Since the references are different, then even if the property values are the same, it results in “false” value. We can see the same in the code below.

1. var initialData = {    
      name: "Mayank Gupta",    
      age: 30    
   }
2. var finalData = {    
      name: "Mayank Gupta",    
      age: 30    
   }

O/P - False 

we have two objects, *userInfo* and *cloneData*. We copy the value of userInfo in cloneData object. Both these variables are now pointing to the same object, since Objects are copied by reference. Updating any of the objects, update the other object as well, since they are referring to the same object. On comparison for equality, it returns true. So for any object, JavaScript look for the base address of the objects being referred.

var userInfo = {    
   name: "Mayank Gupta",    
   age: 30, designation: "Developer"    
}  

var cloneData = userInfo;

Output: true

If we want to copy the object into a new object, we can use the spread operator to create a separate reference for the object.

var userInfo = {    
   name: "Mayank Gupta",    
   age: 30,    
   designation: "Developer"    
}    
var cloneData = { ...userInfo };

Output: false

1. **Why should we not update the state directly?**

If state is updated directly, then it won't re-render the component.

//Wrong    
this.state.message = 'Hello world'

//Correct    
this.setState({ message: 'Hello World' })

1. What is the difference between HTML and React event handling?
   1. In HTML, the event name usually represents in *lowercase* as a convention:

Whereas in React it follows *camelCase* convention:

* 1. In HTML, you can return false to prevent default behavior:

<a href='#' onclick='console.log("The link was clicked.");

 return false;' />

Whereas in React you must call preventDefault() explicitly:

function handleClick(event) {   
  event.preventDefault()   
  console.log('The link was clicked.')   
}

1. What is use of preventdefault()?

**The preventDefault() method cancels the event if it is cancelable**, meaning that the default action that belongs to the event will not occur. For example, this can be useful when: Clicking on a "Submit" button, prevent it from submitting a form.

1. What is the use of refs?

The *ref* is used to return a reference to the element. They *should be avoided* in most cases, however, they can be useful when you need a direct access to the DOM element or an instance of a component.

class MyComponent extends React.Component {

 constructor(props) {

super(props)   
    this.myRef = React.createRef()   
  } 

  render() {

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

1. Which is preferred option with in callback refs and findDOMNode()?

It is preferred to use *callback refs* over findDOMNode() API. The **legacy** approach of using findDOMNode

1. What is Lifting State Up in React?

When several components need to share the same changing data then it is recommended to *lift the shared state up* to their closest common ancestor. That means if two child components share the same data from its parent, then move the state to parent instead of maintaining local state in both of the child components.

1. What is reconciliation?

When a component's props or state change, React decides whether an actual DOM update is necessary by comparing the newly returned element with the previously rendered one. When they are not equal, React will update the DOM. This process is called *reconciliation*.

1. How to set state with a dynamic key name?

handleInputChange(event) {   
  this.setState({ [event.target.id]: event.target.value })   
}

1. How to enable production mode in React?

 use Webpack's DefinePlugin method to set NODE\_ENV to production, by which it strip out things like propType validation and extra warnings.

1. What is the recommended way for naming components?

It is recommended to name the component by reference instead of using displayName.

Using displayName for naming component:

export default React.createClass({ 

  displayName: 'TodoApp',   
  // ...   
})

The **recommended** approach:

export default class TodoApp extends React.Component {   
  // ...   
}

also

const TodoApp = () => {   
//...   
}   
export default TodoApp;

1. What is strict mode in React?

React.StrictMode is a useful component for highlighting potential problems in an application. Just like <Fragment>, <StrictMode> does not render any extra DOM elements. It activates additional checks and warnings for its descendants. These checks apply for *development mode* only.

import React from 'react'   
   
function ExampleApplication() { 

  return (   
    <div> 

      <Header />   
      <React.StrictMode>

        <div> 

          <ComponentOne /> 

<ComponentTwo />   
        </div>

      </React.StrictMode>

      <Header />   
    </div>   
  )   
}

1. Why should component names start with capital letter?

Because react throws error. You can define component class which name starts with lowercase letter, but when it's imported it should have capital letter. Here lowercase is fine:

class myComponent extends Component {

  render() {   
    return <div />   
  }   
}   
   
export default myComponent

While when imported in another file it should start with capital letter:

import MyComponent from './MyComponent'

1. What is the difference between super() and super(props) in React using ES6 classes?

When you want to access this.props in constructor() then you should pass props to super() method.

**Using super(props):**

class MyComponent extends React.Component {

  constructor(props) {   
    super(props)   
    console.log(this.props) // { name: 'John', ... }   
  }   
}

**Using super():**

class MyComponent extends React.Component { 

  constructor(props) {   
    super()   
    console.log(this.props) // undefined   
  }   
}

Outside constructor() both will display same value for this.props.

1. How to re-render the view when the browser is resized?

You can listen to the resize event in componentDidMount() and then update the dimensions (width and height). You should remove the listener in componentWillUnmount() method.

class WindowDimensions extends React.Component {   
  constructor(props){   
    super(props);   
    this.updateDimensions = this.updateDimensions.bind(this);   
  }   
   
  componentWillMount() {   
    this.updateDimensions()   
  }

  componentDidMount() {   
    window.addEventListener('resize', this.updateDimensions)   
  }   
   
  componentWillUnmount() {   
    window.removeEventListener('resize',this.updateDimensions)}

  updateDimensions(){this.setState(

{width:window.innerWidth,height:window.innerHeight})   
  }   
   
  render() {   
    return <span>{this.state.width} x {this.state.height}</span>   
  }   
}

1. What is the difference between setState() and replaceState() methods?

When you use setState() the current and previous states are merged. replaceState() throws out the current state, and replaces it with only what you provide. Usually setState() is used unless you really need to remove all previous keys for some reason.

You can also set state to false/null in setState() instead of using replaceState().

1. Why you can't update props in React?

The React philosophy is that props should be *immutable* and *top-down*. This means that a parent can send any prop values to a child, but the child can't modify received props.

1. How to define constants in React?

You can use ES7 static field to define constant.

class MyComponent extends React.Component {

  static DEFAULT\_PAGINATION = 10   
}

1. What’s the difference between forceUpdate vs setState

 setState() re-render the component if some state or props of that component is changed. When we call setState() the lifecycle method shouldComponentUpdate() method calls automatically that decide if the component should re-render or not. The shouldComponentUpdate() method exit the update life cycle if there is no reason for re-render.

Whereas the forceUpdate() method re-render the component without even changing the state or props. When we call forceUpdate(), it will re-render the component and skip shouldComponentUpdate() method.

|  |  |
| --- | --- |
| **forceUpdate** | **setState** |
| It re-render the component without even changing the state or props. | It re-render the component if some state or props of that component changed. |
| It skips the lifecycle shouldComponentUpdate method. | It calls the lifecycle shouldComponentUpdate method. |

1. What are the ES6 features of Javascript?

ES6 is the version of JavaScript and there are a lot of features of ES6. Few features are - *Arrow Functions, Let and Const, Class*,*and ‘this’* keyword.

1. What is React Router?

**React Router** is a standard library for routing in React. It enables the navigation among views of various components in a React Application, allows changing the browser URL, and keeps the UI in sync with the URL.

* **BrowserRouter:** It is the parent component that is used to store all of the other components and can have only single child
* **Routes:** It’s a new component introduced in the v6 and a upgrade of the component. The main advantages of Routes over Switch are:
  + Routes are chosen based on the best match instead of being traversed in order and pick first match.
* **Route:** Route is the conditionally shown component that renders some UI when its path matches the current URL.

Case1: If you use exact and strict together, then the location.pathname will only match exactly as provided in path props.

Example: <Route exact strict path="/one/" component={About}/>   
  Will only match /one/ but not /one and /one/two.

case 2: If you use only strict, then the location.pathname will match which have trailing slash.

Example: <Route strict path="/one/" component={About}/>   
  Will match /one/ and /one/two but not /one.

case 3: If you use only exact, then the location.pathname will match exact location path.

Example: <Route exact path="/one" component={About}/>   
 Will match /one or /one/. The exact props doesn't care for trailing slash. But it will not match /one/two.

**2. path:** Path specifies a pathname we assign to our component.

**3. element:** It refers to the component which will render on matching the path.

* **Link:** Link component is used to create links to different routes and implement navigation around the application. It works like HTML [anchor tag](https://www.geeksforgeeks.org/html-a-tag/).

Link component uses the **to** prop to describe the location where the links should navigate to.

|  |
| --- |
| <li>     <Link to="/">Home</Link>  </li> |

* The <NavLink> is a special type of <Link> that can style itself as “active” when its to prop matches the current location.

<NavLink to="/react" activeClassName="hurray"> React </NavLink>

1. What is the purpose of push() and replace() methods of history?

A history instance has two methods for navigation purpose.

1. push()
2. replace()

If you think of the history as an array of visited locations, push() will add a new location to the array and replace() will replace the current location in the array with the new one.

1. How to get query parameters in React Router v4?

The ability to parse query strings was taken out of React Router v4 because there have been user requests over the years to support different implementation. So the decision has been given to users to choose the implementation they like. The recommended approach is to use query strings library.

const queryString = require('query-string');   
const parsed = queryString.parse(props.location.search);

You can also use URLSearchParams if you want something native:

const params = new URLSearchParams(props.location.search)   
const foo = params.get('name')

1. Why you get "Router may have only one child element" warning?

You have to wrap your Route's in a <Switch> block because <Switch> is unique in that it renders a route exclusively. At first you need to add Switch to your imports:

import { Switch, Router, Route } from 'react-router'

Then define the routes within <Switch> block:

<Router>   
  <Switch>   
    <Route {/\* ... \*/} />   
    <Route {/\* ... \*/} />   
  </Switch>

</Router>

1. What is history?

History: The history library lets you easily manage **session**history anywhere JavaScript runs. I. It is an array which maintain all url where user has navigated. You can push or replace url

**<**Push location**=**{{

  pathname**:** '/home',

  search**:** '?the=query',

  hash**:** '#the-hash'

  state**:** { some**:** 'state' }

}}**/>**

 Or

this.props.history.push({

                    pathname: '/',

                    state: 4

                })

1. How to pass params to history.push method in React Router v4?

While navigating you can pass props to the history object:

this.props.history.push({   
  pathname: '/template',   
  search: '?name=sudheer',   
  state: { detail: response.data }   
})

The search property is used to pass query params in push() method.

1. How to implement *default* or *NotFound* page?

A <Switch> renders the first child <Route> that matches. A <Route> with no path always matches. So you just need to simply drop path attribute as below

<Switch>   
  <Route exact path="/" component={Home}/>   
  <Route path="/user" component={User}/>   
  <Route component={NotFound} />   
</Switch>

1. How to perform automatic redirect after login?

The react-router package provides <Redirect> component in React Router. Rendering a <Redirect> will navigate to a new location.

import React, { Component } from 'react'

import { Redirect } from 'react-router'   
export default class LoginComponent extends Component {   
  render() {   
    if (this.state.isLoggedIn === true) {

     return <Redirect to="/your/redirect/page" />   
    }

 else {   
      return <div>{'Login Please'}</div>   
    } }}

1. What is the purpose of registerServiceWorker in React?

React creates a service worker for you without any configuration by default. The service worker is a web API that helps you cache your assets and other files so that when the user is offline or on a slow network, he/she can still see results on the screen, as such, it helps you build a better user experience, that's what you should know about service worker for now. It's all about adding offline capabilities to your site.

  import React from 'react'; 

import ReactDOM from 'react-dom'; 

import App from'./App'; 

import registerServiceWorker from './registerServiceWorker'; 

ReactDOM.render(<App />, document.getElementById('root'));

registerServiceWorker();

1. What is Node?

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.

1. What is Node-sass?

Node-sass is a library that provides binding for Node.js to **[LibSass](https://github.com/sass/libsass" \t "_blank), the C version of the popular stylesheet preprocessor, Sass.**

It allows you to natively compile .scss files to css at incredible speed and automatically via a connect middleware.

1. What are Component lifecycle?
2. Mounting:
   1. constructor()

Before the start of the mounting phase, we may need to initialize our component using a constructor() method. This is used when we need to initialize state and bind methods to our component. This is the only place where this.state is explicitly assigned. (*Do not use setState here.*)

Again, two important things happen in this step:

* 1. An object is assigned to this.state
  2. Methods are passed/bound to the instance of the class through props
  3. GetDerivedStateFromProps():

The getDerivedStateFromProps() method is used when the state of a component depends on changes of props. getDerivedStateFromProps(props, state) is a static method that is called just before render() method in both mounting and updating phase in React. It takes updated props and the current state as arguments. We have to return an object to update state or null to indicate that nothing has changed.

static getDerivedStateFromProps(props, state) {

**if**(props.name !== state.name){

            //Change in props

**return**{

                name: props.name

            };

        }

**return** **null**; // No change to state

    }

/\* if props changes then after getDerivedStateFromProps

       method, state will look something like

    {

        name: props.name

    }

\*/

* 1. render():

The render() method is the only method that the component is ***required*** to have. It will always be called and its job is to mount the component to the DOM.

* 1. componentDidMount():

 This method will be immediately invoked after the render function is executed. We can make an API call and update the components state based on its response. We can fill in the content from data we load from another endpoint.

1. Updating:
   1. shouldComponentUpdate():

As the name suggests, this method gives you control over whether or not a component should update due to a change in its props or state. By default, a component will always re-render when updated. This method can either return a true or false. Also, this method receives nextProps and nextState as arguments so you can always compare it with the component’s current prop and state values. This method can help with performance optimization but it is also rarely used.

* 1. render():

If shouldComponentUpdate() returns true, the render function is invoked immediately.

* 1. getSnapshotBeforeUpdate():

In this method, we are given access to the props and state value before the update is committed to the DOM. Even though the render function was already called, we are still able to see the previous values. The use of this case is uncommon but can be used to capture any information we may need. It is important to note that this method works for hand in hand with componentDidUpdate(), which is the next lifecycle method that we will discuss.

* 1. componentDidUpdate():

This method is the last one invoked in this phase. Like the previous method, it also receives the former props and state values as arguments but it also receives the return value getSnapshotBeforeUpdate() as a third argument (if present).It is typically used to make more fetch requests on the condition of comparing the current and previous props and state values. Therefore, you may call setState but within a conditional statement.

1. Unmounting:
   1. componentWillUnmount()

This method is executed right before the component is unmounted from the DOM. You can think of this method as a way to clean up anything that is needed to be removed before the component is destroyed.

1. Error handling methods:

There two methods associated:

1. static getDerivedStateFromError():

If an error were to occur in the child component, this method is invoked. The error will be passed as an argument and this method would return a value that will update the state of the class component. This method is useful for rendering a custom fallback UI instead of rendering a broken component.

1. componentDidCatch():

This method receives the error as an argument as well as an object that has information about the error. This method is perfect for logging the error!

Timeline

Description automatically generated with medium confidence

1. What is Babel?

Babel is a JavaScript compiler

Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments. Here are the main things Babel can do for you:

* Transform syntax
* Polyfill features that are missing in your target environment (through a third-party polyfill such as [core-js](https://github.com/zloirock/core-js))

1. Why you should add babel polyfill for every React application?

Because using babel alone does not guarantee that your code will run on all browsers. As you are aware, the latest ES6 features such as promises, arrow functions, etc may not be supported by older browsers, so we use Babel to convert the code to ES5 code which all browsers understand.

When we use create-react-app to create a React application, it includes Babel by default for the conversion of ES6 to ES5 code. But that is not sufficient to support all browsers.

So if your browser understands ES6, then your code will work fine, but for older browsers like Internet explore version 11 or lower, this code will not work.

Therefore, if your application is using promises directly or indirectly using axios or superagent library, then your application will not work on internet explorer or older browsers.

Let’s consider Array.prototype.includes method (added in ES7) which is used to check if a particular element exists in an array. Graphical user interface, text, application

Description automatically generated

As you can see, for the Array.prototype.includes method, the ES5 code is the same as the input, so your application will not work on Internet Explorer if you are using this method anywhere in your application.

**So long story short**, just using babel is not enough for your application to work because all the latest Javascript features are not supported in all browsers. So to fix this problem, we need to use a polyfill.

1. What is React.lazy & React.Suspense?

React.lazy will automatically load the bundle containing the OtherComponent when this component is first rendered. The lazy component should then be rendered inside a Suspense component, which allows us to show some fallback content (such as a loading indicator) while we’re waiting for the lazy component to load.

 The React.lazy function lets you render a dynamic import as a regular component. React.lazy takes a function that must call a dynamic import(). This must return a Promise which resolves to a module with a default export containing a React component.

Before: import OtherComponent from './OtherComponent';

After: const OtherComponent = React.lazy(() => import('./OtherComponent'));

import React, { Suspense } from 'react';

const OtherComponent = React.lazy(() => import('./OtherComponent'));   
   
function MyComponent() {   
  return (   
    <div>

<Suspense fallback={<div>Loading...</div>}>   
<OtherComponent />   
</Suspense>

    </div>   
  );   
}

The fallback prop accepts any React elements that you want to render while waiting for the component to load. You can place the Suspense component anywhere above the lazy component. You can even wrap multiple lazy components with a single Suspense component.

      <Suspense fallback={<div>Loading...</div>}>

        <section>   
          <OtherComponent />

          <AnotherComponent />   
        </section>   
      </Suspense>   
    </div>   
  );   
}

1. Does React router lazy load?

on demand using React. lazy() and dynamic import() . Using this technique, pages that are not required on the home page can be split out into separate bundles, thereby decreasing **load** time on the initial page and improving performance

const About = React.lazy(() => import("./pages/About"));

const Dashboard = React.lazy(() => import("./pages/Dashboard"));

export default function App() {

  return (

    <div>

    <Routes>

        <Route path="/" element={<Layout />}>

          <Route index element={<Home />} />

          <Route path="about" element={

              <React.Suspense fallback={<>...</>}>

                <About />

              </React.Suspense>

            }

          />

          <Route path="dashboard/\*" element={

              <React.Suspense fallback={<>...</>}>

                <Dashboard />

              </React.Suspense>

            }

          />

          <Route path="\*" element={<NoMatch />} />

        </Route>

      </Routes>

    </div>

  );

}

1. What is the use of lazy loading?

The benefits of lazy loading include: **Reduces initial load time** – Lazy loading a webpage reduces page weight, allowing for quicker page load time. Bandwidth conservation – Lazy loading conserves bandwidth by delivering content to users only if it's requested.

Graphical user interface, diagram

Description automatically generated