1. **Promises in Sequence**

Promises in sequence refer to the execution of asynchronous operations one after the other, where each operation depends on the completion of the previous one.

Each .then() method returns a new promise and allow to chain asynchronous operations.

The next .then() will only execute after the previous promises resolved.

promise1()  
 .then(result1 => {  
 console.log("Promise 1 resolved:", result1);  
 return promise2(result1);  
 })  
 .then(result2 => {  
 console.log("Promise 2 resolved:", result2);  
 return promise3(result2);  
 })  
 .catch(error => {  
 console.error("An error occurred:", error);  
 });

1. **Array Pollyfill method**: forEach(), Map(), Filter(), reduce(), find() and splice() etc.
2. **Prototype and Prototype inheritance**

// Parent object

let animal = {

eat: function() {

console.log("Eating...");

}

};

// Child object

let dog = Object.create(animal);

dog.bark = function() {

console.log("Woof!");

};

dog.eat(); // Output: Eating...

dog.bark(); // Output: Woof!

In this example, dog inherits the eat() method from animal using Object.create(). This demonstrates prototypal inheritance in JavaScript.

1. **Flatten Array**

In JavaScript, a flattened array is an array that has been transformed from a nested array structure to a one-dimensional array. Here's an example:

JavaScript

// Nested array

let nestedArray = [1, 2, [3, 4], [5, [6, 7]]];

// Flattened array

let flattenedArray = [1, 2, 3, 4, 5, 6, 7];

To flatten an array in JavaScript, you can use the flat() method:

JavaScript

let nestedArray = [1, 2, [3, 4], [5, [6, 7]]];

let flattenedArray = nestedArray.flat(Infinity);

console.log(flattenedArray); // Output: [1, 2, 3, 4, 5, 6, 7]

**Promise.all():** promise.all() method returns a single promise from a list of promises, when all promises

Fulfill. Its handling multiple API request simultaneously.

If any promise rejects, the return promise reject with the reason of first promise that rejected.

**Promise.any()** Promise.any() method is settled as soon as any of the promises you feed it is

fulfilled or they are all rejected.

**Promise.race()** Promise. race() returns a promise that resolves or reject to the value from the first settled promise.

**Promise.allSettled()** Promise.allSettled() is useful when you need to handle the results of multiple asynchronous operations, even if some of them fail. It allows you to examine the outcome of each promise individually, rather than being stopped by a single rejection.

Promise.allSettled() is a JavaScript method that takes an array of promises as input and returns a single promise.

**How to handle browser resize or scroll events in React?**

Using window.addEventListener: You can add an event listener to the window object in the useEffect hook.

**Lifting state up, what is it and when do you do it?**

Lifting state up is a design pattern in React where you move the state from a child component to a parent component or a higher-level component in the component tree. This allows multiple components to share and access the same state.

**What is the difference between package.json and package-lock.json?**

package.json defines the dependency installation requirements, while package-lock.json ensures that the exact same versions are installed across all environments. This helps prevent version conflicts.

**What is call by value vs call by reference?**

Call by Value:

* Primitive types (number, string, boolean, null, undefined) are passed by value.
* A copy of the value is passed to the function.
* Changes made inside the function don't affect the original variable.
* function changeObject(obj) {
* obj.name = "John"; // This will affect the original object
* obj = { name: "Jane" }; // This won't affect the original object
* }
* let person = { name: "Alice" };
* console.log("Before function call:", person);
* changeObject(person);
* console.log("After function call:", person);

**Call by reference:**

JavaScript is technically "call by value," but when you pass an object (including arrays), the value is a reference to the object. This means you can mutate the object's properties within a function, but reassigning the variable won't affect the original.

Example:

function changeName(person) {

person.name = 'Jane'; // This change will be reflected outside

person = { name: 'John', age: 30 }; // This won't affect the original object

}

let person = { name: 'Alice', age: 25 };

changeName(person);

console.log(person); // Output: { name: 'Jane', age: 25 }

Explain example:

In this example, person.name is changed to 'Jane' because the function mutates the object that person references. However, when the function assigns a new object to person, it doesn't change the original person variable outside the function; it only changes the local person variable within the function

**How to optimize performance with a heavy banner/logo in React when the client insists on using it.**

1. **Image Compression**: Compress the image to reduce its file size without compromising on quality. Tools like TinyPNG, ImageOptim etc. can help for optimize performance.
2. **Lazy Loading:** Implement lazy loading for the banner/logo so it loads only when it comes into view using libraries like react-lazy-loading.

**Implement a debounce function with parameters (fn, delay) and explain with examples.**

//example 1

function debounce(fn, delay) {

  let timeoutId;

  return function (...args) {

    clearTimeout(timeoutId);

    timeoutId = setTimeout(() => {

      fn(...args);

    }, delay);

  };

}

//explanation

// The debounce function takes a function fn and a delay delay as parameters.It returns a new function

// that wraps the original function. When the wrapped function is called,

// it clears any existing timeout and sets a new timeout to call the original function after the specified delay.

**Key Components:**

1. **Continuous Integration (CI):** Developers integrate their code changes into a central repository, triggering automated builds and tests to ensure the codebase remains stable and functional.
2. **Continuous Delivery (CD):** Once the code is built and tested, it's prepared for deployment to production. This stage ensures the software is ready for release, but the actual deployment may require manual approval.
3. **Continuous Deployment (CD):** This stage takes Continuous Delivery a step further by automatically deploying the software to production after passing all tests and checks.

**Pipeline Stages:**

1. **Source:** Code changes are committed to a version control system (e.g., Git).
2. **Build:** The code is compiled and built into an executable format.
3. **Test:** Automated tests (unit tests, integration tests, UI tests) are run to ensure the code works as expected.
4. **Deploy:** The built and tested code is deployed to a staging or production environment.
5. **Monitor:** The deployed application is monitored for performance, errors, and other issues.

**Tools:**

Popular CI/CD tools include:

* Jenkins
* GitLab CI/CD
* CircleCI
* Travis CI
* GitHub Actions

By implementing a CI/CD pipeline, development teams can improve the speed, quality, and reliability of their software releases.

**Q. What security features should be taken while designing API's?**

* Authentication, Authorization, Data Encryption like (HTTPS – SSL/TLS)
* Input validation, Error Handling, Secure Storage, Content Security Policy. Two Factor authentication etc.

**Q. What is API throttling?**

API throttling in react refers to limiting the number of API requests made to a server within a certain time frame. This technique helps reduce server load. Minimize the risk of hitting rate limits.

**Q. Debouncing**

Debouncing ensures that an API request is made only after a specified delay has passed since the last user interaction.

**Q. What if in a react app we need to develop a feature of auto save the inputs of a form?**

For auto save the input, we can follow the combination of

1. Debouncing for save the operation.

2. Local Storage or API to store the auto saved the data.

**Q. If you have to inform the backend developers about some API's are failing how will you do that?**

I will follow some steps for backend team.

1. API names and endpoint that are failing
2. Which error message or code received

All these things will say to backend developers and request to investigate and fix ASAP.

**Q. Which tool is used to improve code standards in react application to show warnings for developers?**

Ans:- ESLint and JSHint for detecting the syntax errors and code quality issues.

**Q. Which cloud is used in your app development?**

1. Google cloud 2. Amazon web services 3. Microsoft Azure

**What are limitations of arrow functions in JavaScript?**1. Arrow functions are introduced in ES6. They are simple and shorter way to write functions in JavaScript.  
2. Arrow functions cannot be accessed before initialization  
3. Arrow function does not have access to arguments object  
4. Arrow function does not have their own this. Instead, they inherit this from the surrounding code at the time the function is defined.  
5. Arrow functions cannot be used as constructors. Using them with the 𝙣𝙚𝙬 keyword to create instances throws a Type Error.  
6. Arrow functions cannot be used as generator functions.

**About The Projects**

* 1. **Recruit Web Application**

Recruit web application is a job portal designed to connect job seekers with potential employers. So I created

**1st - Job listing page:** where user can search a job with filters for location, job type and industry.

**2. – Job Application Page:** where user can apply for jobs by uploading their resume and covers letters.

**3. - User Profile Page**: Where job seekers can create a profile to showcase their skills, experience and educa.

**4. – Employer Dashboard:** Where employers can manage job postings, view candidate application etc.

And the responsive design application is optimized for desktop, tablets and mobile devices.

I used tack stacks in their application **for frontend development**

1. HTML5, CSS3, JavaScript, React, Tailwind and Material UI etc.
2. Integration API and I worked closely with the backend team to ensure communication and data exchange.

**Challenging task:**

1. State Management, responsive UI across different devices and screen sizes.
2. Integration with Backend API’s to fetching and sending the data to the server-side application

That’s All.

**2 - Deluxe Banking Web Application**

I worked on an online banking system web application project, which is used to provide customers with a secure way to manage their account online.

So, as a frontend developer, my role was to get the design, convert and implement the UI components and user experience.

So, I was responsible for developing **the user authentication, registration, training, scan, rescan, account management** features as well as micro frontend technologies used in that project and API integration etc.

I used tack stacks in their application for frontend development.

1. HTML5, CSS3, JavaScript, React, Bootstrap and Material UI, GIT etc.
2. Integration API and I worked closely with the backend team to ensure communication and data exchange.

**Security Measures:**

1. **HTTPS**: Implemented secure communication between client and server.
2. **Data Encryption**: **Crypto-JS**- A popular JavaScript library for encryption.
3. **Two-Factor Authentication**: like **time-based OTP (TOTP)** and other authenticator apps like **Google Authenticator.**

That’s all about the project.