1. **Describe the role of Postman in API testing. Explain steps for testing GET, POST, PUT, and DELETE requests.**

Postman is a powerful API development and testing tool that simplifies the process of building, testing, and managing APIs. It allows developers to send HTTP requests, analyze responses, automate workflows, and collaborate efficiently. With support for multiple authentication methods, request body formats, and automated testing, Postman has become one of the most popular tools for modern API development.

- Supports GET, POST, PUT, PATCH, and DELETE for comprehensive API interaction.
- Send data in form-data, URL-encoded, raw (JSON, XML, text), or binary formats.
- Built-in support for API keys, OAuth, Bearer Tokens, and Basic Auth.
- Organize related API requests for better collaboration and project management.
- Write test scripts in JavaScript and schedule runs for continuous integration.
- Generate and share API docs directly from requests and collections.
- Reuse and share workspaces without starting from scratch.

Steps for Testing API Requests in Postman:

1. Create a New Request:

- Open Postman and click on the + button or "New" to create a new request.

2. Configure the Request:

- **URL:** Enter the API endpoint URL in the request URL field.
- **HTTP Method:** Select the appropriate HTTP method (GET, POST, PUT, DELETE) from the dropdown menu.

3. Add Headers (if required):

- Go to the "Headers" tab and add any necessary headers (e.g., Content-Type, Authorization, Accept).

4. Add Parameters (if required):

- Go to the "Params" tab to add query parameters (for GET requests) or path parameters (as part of the URL).

5. Add Request Body (for POST and PUT requests):

- Go to the "Body" tab.
- Select the appropriate format (e.g., raw and JSON) and enter the request body data.

6. Send the Request:

- Click the "Send" button to execute the request.

7. Analyze the Response:

- **Status Code:** Check the HTTP status code (e.g., 200 OK, 201 Created, 404 Not Found).
- **Response Body:** Examine the response data in the "Body" tab.
- **Response Headers:** Review the response headers in the "Headers" tab.
- **Tests (Optional):** Write assertions in the "Tests" tab using JavaScript to automate response validation.

Specific Steps for Each Method:

GET Request (Retrieve Data):

- Select "GET" as the HTTP method.
- Enter the API endpoint URL.
- Add any necessary query parameters in the "Params" tab.
- Click "Send" and verify the retrieved data in the response body.

POST Request (Create Data):

- Select "POST" as the HTTP method.
- Enter the API endpoint URL.
- In the "Body" tab, select raw and JSON, then enter the data to be created.
- Click "Send" and verify the successful creation (e.g., 201 Created status) and the created resource in the response.

PUT Request (Update Data):

- Select "PUT" as the HTTP method.
- Enter the API endpoint URL, including the ID of the resource to be updated.
- In the "Body" tab, select raw and JSON, then enter the updated data.
- Click "Send" and verify the successful update (e.g., 200 OK status) and the updated resource in the response.

DELETE Request (Remove Data):

- Select "DELETE" as the HTTP method.
- Enter the API endpoint URL, including the ID of the resource to be deleted.
- Click "Send" and verify the successful deletion (e.g., 200 OK or 204 No Content status).

2. **Explain how error handling is implemented in Fetch API and Axios.**

Error handling in Fetch API and Axios, while both promise-based, differs in their approach to what constitutes an "error" that triggers a catch block.

Fetch API Error Handling:

The Fetch API's fetch() call only rejects the promise and triggers the .catch() block in cases of network errors (e.g., no internet connection, DNS resolution failure) or if something prevents the request from completing (e.g., request canceled). For HTTP status codes indicating server-side errors (like 404 Not Found or 500 Internal Server Error), fetch() resolves the promise, but the Response object's ok property will be set to false.

Therefore, to handle HTTP error status codes with Fetch, explicit checking of response.ok or response.status within the .then() block is necessary.

JavaScript

```
fetch('https://api.example.com/data')
  .then(response => {
    if (!response.ok) {
      // Handle HTTP error status codes (e.g., 404, 500)
      return response.json().then(errorData => {
        throw new Error(errorData.message || `HTTP error! status: ${response.status}`);
      });
    }
    return response.json();
  })
  .then(data => {
    // Process successful response data
    console.log(data);
  })
  .catch(error => {
    // Handle network errors or errors explicitly thrown in the .then() block
    console.error('Fetch error:', error);
  });
```

Axios Error Handling:

Axios, by default, provides a more streamlined error handling mechanism. It automatically rejects the promise and triggers the .catch() block not only for network errors but also for any HTTP status code that falls outside the 2xx range (e.g., 4xx client errors, 5xx server errors).

The error object received in the .catch() block typically contains a response property, which holds details about the HTTP error, including status and data (the error response body).

JavaScript

```
axios.get('https://api.example.com/data')
  .then(response => {
    // Process successful response data (2xx status codes)
    console.log(response.data);
  })
  .catch(error => {
    // Handle network errors and HTTP error status codes (e.g., 4xx, 5xx)
    if (error.response) {
      // The request was made and the server responded with a status code
```

```
    // that falls out of the range of 2xx
    console.error('Axios error - response data:', error.response.data);
    console.error('Axios error - status:', error.response.status);
  } else if (error.request) {
    // The request was made but no response was received
    console.error('Axios error - no response:', error.request);
  } else {
    // Something happened in setting up the request that triggered an Error
    console.error('Axios error - message:', error.message);
  }
});
```

3 Explain React Hooks. Discuss useState and useEffect with examples.

ReactJS Hooks, introduced in React 16.8, are among the most impactful updates to the library, with over 80% of modern React **projects** adopting them for state and lifecycle management. They let developers use state, side effects, and other React features without writing class components. Hooks streamline code, improve readability, and promote a functional programming style, making functional components as powerful—if not more—than their class-based counterparts.

ReactJS is the powerful JavaScript library used for building user interfaces especially single-page applications(SPAs). Two of the most commonly used hooks in React

are **useState** and **useEffect**. These hooks make functional components more powerful by allowing them to manage the state and handle side effects.

What is useState?

useState is the hook that allows you to add state to functional components in React. State refers to data that can change over time and trigger re-renders of components. Before the introduction of **useState** only class components could manage state.

When we call **useState**, it returns an array with two values:

- The Current state value
- The Function to update the state

This allows you to manage dynamic data in your component. Every time the state changes, React re-renders the component to reflect updated values.

- **useState** hook gives us a state variable and the function to update it.
- The Component re-renders when the state changes.
- You can have multiple state variables by calling **useState** multiple times.

Example: count is the state variable and setCount is a function used to update count. When the button is clicked setCount(count + 1) increases the value of the count by 1.

```
// App.js
import React, { useState } from 'react';

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

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}

export default Counter;
```

What is useEffect?

useEffect is the hook that allows you to perform side effects in functional components. The side effect is any operation that affects something outside a component such as data fetching, interacting with the browser DOM or setting up timers.

useEffect takes two arguments:

- 1st argument to **useEffect** is function that contains side effect code.
- 2nd argument is an array of dependencies. The effect runs only when one of these dependencies changes. If array is empty then effect runs only once when component mounts.

useEffect is also used for cleanup like removing event listeners by returning function inside it. The function containing side effect logic (e.g., fetching data or updating document title). An optional array of dependencies that tells React when to re-run effect. If array is empty then effect runs only once when component mounts.

Example: In this example every time count changes **useEffect** runs and updates document title with new count.

```
// App.js
import React, { useState, useEffect } from 'react';

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

  useEffect(() => {
    document.title = `You clicked ${count} times`;
  }, [count]);  // This effect runs only when 'count' changes
```

```
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}

export default Counter;
```

4. **What is Context API? Explain its working and advantages with example code.**

Context API in React is used to share data across the components without passing the props manually through every level. It allows to create global state of data providing global access to all the components.

Here, we will discuss about the context API in React and its uses with implementation.

What is Context API?

Context API is used to pass global variables anywhere in the code without the prop drilling. It helps when there is a need for sharing state between a lot of nested components. It is light in weight and easier to use, to create a context just need to call React.createContext(). No need to install other dependencies or third-party libraries like redux for state management.

Why is Context API used?

Context API solves the problem of prop drilling in React. Prop Drilling occurs when data is to be passed between multiple layers before finally sending it to the required component. This makes the application slower. This problem is solved by Context API as it creates global variables to be used throughout the application without any middle components involved. It is also easier to use than React Redux

Working

To work with Context API we need React.createContext(). It has two properties Provider and Consumer. The Provider acts as a parent it passes the state to its children whereas the Consumer uses the state that has been passed.

- **Provider:** This is used to provide the context to components.

- **Consumer:** This is used to consume the context value in child components.

```
import React, { createContext } from 'react';
export const UserContext = createContext();

const UserProvider = ({ children }) => {
  const user = { name: 'John Doe', id: 1 };

  return (
    <UserContext.Provider value={user}>
      {children}
    </UserContext.Provider>
  );
};

export default UserProvider;
```

5. Compare Material UI and Bootstrap. How do they improve UI design?

- **Bootstrap** is a powerful, free, open-source CSS framework created by Twitter as their internal framework to make developer's life easier by implementing predefined classes for building responsive websites and web applications. Since Twitter made it open-source in 2011, Bootstrap is the most popular and most used CSS framework with a large community. Many big companies like Airbnb, Apple Music, Dropbox, Coursera, and so on.
- The **Material UI** is in 2017, which mainly works on Facebook's React framework and Google's Material Design guidelines. It provides an exhausting framework for creating websites and applications with high interactive and mobile-first UI with responsive designs. Material Design and material UI are strongly connected with each other and material UI is just a component library without material design. Material Design is used by Google in all of its products while the newly trending Material UI is currently used by NASA, Unity, Amazon, JPMorgan and the list is going to increase in upcoming years

Bootstrap	Material UI
It is an HTML, CSS & JS framework to make responsive websites that are mobile-friendly and easy to create.	Material UI is a highly interactive & customizable framework based on React UI and Material Design.
Developed by Twitter, initially named Twitter Blueprint.	Developed by a small team of passionate developers by referring to Google's material design.
Currently, an open-source project maintained by Mark Otto, Jacob Thornton, and a small group of core developers, as well as a large community of contributors	Maintained by founding team and group of core contributors as well as the material community.

Bootstrap	Material UI
High speed of development because of reusable code	Speed of development is lesser than compared to bootstrap but can be increased by extensive use of reusable components and templates.
12-column grid system for responsive design	12-column grid system, same as bootstrap
Bootstrap's information layout provides a clear and consistent UI for all platforms.	Also, it a mobile-first and supports all platforms, but excess use of customizations, transitions, and animations might affect accessibility on some platforms.

6. Discuss responsive design principles. Explain media queries and responsiveness frameworks.

Responsive design principles focus on creating websites that adapt seamlessly to various screen sizes and devices, providing an optimal viewing experience for users. Key principles include:

- **Fluid Grids:**

Layouts are built using relative units (percentages, em, rem) instead of fixed pixels, allowing elements to scale proportionally with the viewport size.

- **Flexible Images and Media:**

Images and other media are sized using relative units or CSS properties like max-width: 100% to prevent overflow and ensure they scale appropriately.

- **Media Queries:**

These CSS rules apply specific styles based on device characteristics like screen width, height, orientation, and resolution, enabling tailored layouts for different breakpoints.

- **Mobile-First Approach:**

Designing for smaller screens first and then progressively enhancing the experience for larger screens helps ensure core content and functionality are prioritized and accessible on all devices.

- **Touch Targets and Readability:**

Ensuring interactive elements are large enough for touch interaction and text remains readable without excessive zooming or scrolling on smaller screens.

- **Performance Optimization:**

Optimizing images, minimizing code, and using efficient loading techniques to ensure fast loading times across all devices.

Media Queries:

Media queries are a core component of responsive design, allowing developers to apply different CSS styles based on the characteristics of the user's device or browsing environment. They are conditional rules that target specific media types (e.g., screen, print) and features (e.g., max-width, min-width, orientation, resolution).

Here is an example:

Code

```
/* Styles for screens smaller than 768px */
@media screen and (max-width: 767px) {
  body {
    font-size: 14px;
  }
  .navigation {
    flex-direction: column;
  }
}

/* Styles for screens larger than or equal to 768px */
@media screen and (min-width: 768px) {
  body {
    font-size: 16px;
  }
  .navigation {
    flex-direction: row;
  }
}
```

Responsiveness Frameworks:

Responsiveness frameworks are pre-built collections of HTML, CSS, and sometimes JavaScript components designed to streamline the creation of responsive websites. They offer a structured approach to building layouts and handling responsiveness, often including:

- **Grid Systems:**

Predefined classes for creating fluid, responsive grid layouts (e.g., 12-column grids).

- **Pre-styled Components:**

Ready-to-use UI elements like navigation bars, buttons, forms, and cards that are inherently responsive.

- **Utility Classes:**

Helper classes for common styling tasks like spacing, alignment, and display properties.

- **JavaScript Plugins:**

Functionality for interactive elements like carousels, modals, and dropdowns, often designed with responsiveness in mind.

Popular responsiveness frameworks include:

- **Bootstrap:**

A widely used, comprehensive framework offering a vast array of components and a robust grid system.

- **Tailwind CSS:**

A utility-first CSS framework that provides low-level utility classes for building custom designs directly in HTML.

- **Foundation:**

Another popular framework known for its mobile-first approach and semantic HTML.

7. Explain the steps to deploy a React application on Firebase Hosting.
When developing any project we must host it somewhere so that the whole world can see our hard-work. Hosting websites can be hectic sometimes, but you don't need to worry as we can now host our React project on **Firebase** within a minute or two with a very simple setup. Deploying a React application on Firebase Hosting involves several steps:

- **Create a Firebase Project:**
    - Navigate to the Firebase console and create a new project. Follow the on-screen instructions to set up the project.
- **Install Firebase CLI:**
    - Open your terminal or command prompt and install the Firebase Command Line Interface (CLI) globally using npm:

Code

```
npm install -g firebase-tools
```

- **Authenticate with Firebase:**
    - Log in to your Firebase account through the CLI:

Code

```
firebase login
```

- This will open a browser window for you to authenticate with your Google account.
- **Initialize Firebase in your React Project:**
    - Navigate to the root directory of your React project in the terminal.
    - Initialize Firebase for your project:

Code

```
firebase init
```

- During the initialization process:
    - Select "Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys."
    - Choose "Use an existing project" and select the Firebase project you created earlier.
    - When prompted for the public directory, enter build (this is where React's production-ready files are located).
    - Configure as a single-page app by answering "Yes" to "Configure as a single-page app (rewrite all URLs to /index.html)?"
- **Build Your React Application:**
    - Create a production build of your React application. This generates the optimized static files in the build directory:

Code

```
npm run build
```

- **Deploy to Firebase Hosting:**
    - Once the build is complete, deploy your application to Firebase Hosting:

Code

```
firebase deploy
```

- The CLI will upload the contents of your build directory to Firebase Hosting and provide you with a URL where your application is live.

8. Describe Vercel deployment workflow for modern web applications.

A **deployment** on Vercel is the result of a successful build of your project. Each time you deploy, Vercel generates a unique URL so you and your team can preview changes in a live environment.

Vercel supports multiple ways to create a deployment:

- Git
- Vercel CLI
- Deploy Hooks
- Vercel REST API

Deployment Methods

Git

The most common way to create a deployment is by pushing code to a connected Git repository. When you import a Git repository to Vercel, each commit or pull request (on supported Git providers) automatically triggers a new deployment.

Vercel supports the following providers:

- GitHub
- GitLab
- Bitbucket
- Azure DevOps

You can also create deployments from a Git reference using the Vercel Dashboard if you need to deploy specific commits or branches manually.

Vercel CLI

You can deploy your Projects directly from the command line using Vercel CLI. This method works whether your project is connected to Git or not.

1. **Install Vercel CLI**:

npm i -g vercel

2. **Initial Deployment**:

In your project's root directory, run:

vercel --prod

This links your local directory to your Vercel Project and creates a Production Deployment. A .vercel directory is added to store Project and Organization IDs.

Vercel CLI can also integrate with custom CI/CD workflows or third-party pipelines. Learn more about the different environments on Vercel.

Deploy Hooks

Deploy Hooks let you trigger deployments with a unique URL. You must have a connected Git repository to use this feature, but the deployment does not require a new commit.

1. From your Project settings, create a Deploy Hook
2. A unique URL is generated for each Project
3. Make an HTTP GET or POST request to this URL to trigger the deployment

Refer to the Deploy Hooks documentation for more information.

Vercel REST API

The Vercel REST API lets you create deployments by making an HTTP POST request to the deployment endpoint. In this workflow:

1. Generate a SHA for each file you want to deploy
2. Upload those files to Vercel
3. Send a request to create a new deployment with those file references

This method is especially useful for custom workflows, multi-tenant applications, or integrating with third-party services not officially supported by Vercel. For more details, see the API reference and How do I generate an SHA for uploading a file.

Accessing Deployments

Vercel provides three default environments—**Local**, **Preview**, and **Production**:

1. **Local Development**: developing and testing code changes on your local machine
2. **Preview**: deploying for further testing, QA, or collaboration without impacting your live site
3. **Production**: deploying the final changes to your user-facing site with the production domain

Using the Dashboard

Vercel's dashboard provides a centralized way to view, manage, and gain insights into your deployments.

Resources Tab and Deployment Summary

When you select a deployment from your **Project → Deployments** page, you can select the **Resources** tab to view and search:

- **Middleware**: Any configured matchers.
- **Static Assets**: Files (HTML, CSS, JS) and their sizes.
- **Functions**: The type, runtime, size, and regions.

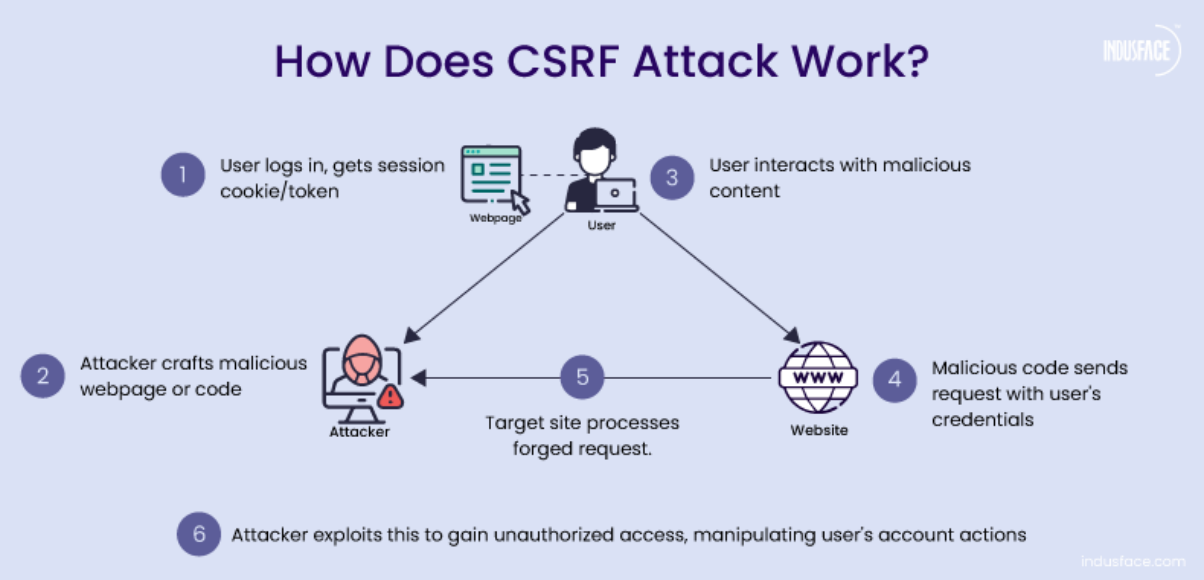
9 Discuss XSS and CSRF attacks. Explain prevention techniques for each.

Cross-Site Scripting (XSS) attacks inject malicious scripts into a trusted website, which are then executed by a victim's browser, while Cross-Site Request Forgery (CSRF) tricks a user's browser into sending unintended, malicious HTTP requests to a vulnerable site they are already authenticated with. To prevent XSS, use input validation, sanitization, output encoding, and Content Security Policy (CSP); for CSRF, implement anti-CSRF tokens, validate the Origin and Referer headers, use SameSite cookies, and enforce custom headers.

Cross-Site Scripting (XSS)

- **What it is:** The attacker injects malicious scripts (usually JavaScript) into a web page or application that is then executed by other users' browsers. The goal is to steal user data like cookies or session tokens, or to perform actions on the user's behalf.
- **How it works:** This often occurs when a website includes unsanitized user input in the output it generates.
- **Prevention techniques:**
    - **Input validation and sanitization:** Ensure user input is clean and does not contain malicious code before processing it.
    - **Output encoding:** Encode any data that is outputted to the browser so that it is treated as text, not code.
    - **Content Security Policy (CSP):** A server-side response header that allows you to define trusted sources for scripts, styles, and other resources, preventing the browser from loading and executing scripts from untrusted URLs.
    - **Subresource Integrity (SRI):** Use this to ensure that external scripts fetched by your site haven't been tampered with by verifying their integrity against a known hash.
    - **Use secure frameworks:** Some modern frameworks automatically handle many of these security measures, like escaping outputs.

Cross-Site Request Forgery (CSRF)



- **What it is:** Tricking a user's browser into making an unintended request to a web application where the user is authenticated.
- **How it works:** An attacker can craft a malicious link or form on a different site that, when clicked by the victim, sends a request to the target site (e.g., to change an email address or transfer funds) using the user's existing authenticated session.
- **Prevention techniques:**
    - **Anti-CSRF tokens:** Generate a unique, unpredictable token for each user session that is included in state-changing forms. The server validates this token upon receiving the request to ensure it originated from the trusted site.
    - **SameSite cookies:** Use the SameSite attribute with values like Strict or Lax to prevent the browser from sending cookies with cross-site requests, as explained in this Acunetix article and by Pynt.
    - **Custom request headers:** Check for a custom, non-standard header in the HTTP request to verify it's from your own site.
    - **Double-submit cookies:** Use a pattern where the CSRF token is sent as both a cookie and a form field, and the server verifies that both values match.

10. Explain the role of linting tools (ESLint) in ensuring code quality.

Linting tools like ESLint play a crucial role in ensuring code quality by performing static code analysis to identify and report on problematic patterns in source code. This process helps maintain consistency, prevent bugs, and improve the overall readability and maintainability of a codebase.

ESLint specifically contributes to code quality:

- **Enforcing Coding Standards and Style Guides:**

ESLint allows developers to define and enforce specific coding conventions and style rules, such as indentation, brace style, and variable naming conventions. This ensures consistency across a project, making the code easier to read and understand for all team members.

- **Detecting Potential Errors and Bugs:**

ESLint can identify common programming errors and potential bugs before runtime, such as undeclared variables, unreachable code, or incorrect use of operators (e.g., flagging the use of == instead of ===). Catching these issues early in the development cycle saves time and effort in debugging.

- **Promoting Best Practices:**

ESLint can be configured with rules that encourage the adoption of best practices in JavaScript/TypeScript development, such as avoiding global variables, optimizing performance, or using modern language features effectively.

- **Improving Code Readability and Maintainability:**

By enforcing consistent formatting and identifying complex or poorly structured code, ESLint helps create a cleaner, more organized codebase. This significantly improves readability and makes the code easier to maintain and refactor in the long run.

- **Facilitating Collaboration:**

When all developers on a team adhere to the same coding standards enforced by ESLint, it reduces friction during code reviews and integration, leading to smoother collaboration and higher quality output.

- **Customization and Extensibility:**

ESLint's pluggable architecture allows developers to create custom rules and integrate community-driven plugins and configurations. This flexibility enables tailoring ESLint to meet specific project requirements and coding styles.