[-- What is Accessibility? 2](#_Toc1859)

[-- What is SOLID principles? 2](#_Toc1137)

[-- Typescript Utility Types 3](#_Toc8891)

[-- Socket 5](#_Toc12751)

[--Portals 5](#_Toc1129)

[-- This keyword 5](#_Toc5132)

[-- Hoisting 6](#_Toc31233)

[--Scope 6](#_Toc18426)

[--Closure 7](#_Toc25394)

[Blocking resource in CSS 7](#_Toc13326)

[Context api vs Redux 8](#_Toc1053)

[Long polling vs Standard polling 8](#_Toc3452)

[Full duplex communication, Web sockets, Web socket handshake 8](#_Toc109)

[Explain pagination, name web pages that uses pagination 9](#_Toc18270)

[Lazy loading - explain and how to implement; name web pages as an example of lazy loading 10](#_Toc27757)

[package.log explain (.log vs package..) 11](#_Toc17416)

[Devdependencies vs dependencies 11](#_Toc2957)

[Minification uglification, mapping (source maps) 12](#_Toc8686)

[Monolith vs microservices 12](#_Toc20522)

[Advantage of micro services and microfrontends 13](#_Toc2250)

[Orchestration of micro services VS choreography of micro services 13](#_Toc3745)

[Oauth 2.0 14](#_Toc16611)

[Oauth (JSON webtoken) 14](#_Toc9717)

[Distributed tracing (time and correlationId) 15](#_Toc28072)

[How to implement component will unmount in a functional component? 18](#_Toc12935)

[What is an example for using component will unmount? 19](#_Toc19959)

[How to handle errors in React 19](#_Toc17331)

[HOC (higher order component) - explain, how/why to use 19](#_Toc14280)

[React DOM - explain, how/why to use 20](#_Toc4956)

[Api interceptor/middleware - example 20](#_Toc29211)

[PWA - Progressive Web Application - explain 21](#_Toc30363)

[What is the library you use for unit testing? 21](#_Toc26450)

[If the story is not clear how do you approach that? 23](#_Toc14646)

[How do you get familiar with a new code base? 23](#_Toc15200)

[What is SPA, and why is it important/good? 24](#_Toc24973)

[Difference between localstorage and cookie. 24](#_Toc27767)

[Can you create ErrorBoundary as functional component? 24](#_Toc2326)

[Responsive layout vs Adaptive layout (mobile) 24](#_Toc6679)

[What is PropTypes? 25](#_Toc5542)

[What is web pack? 25](#_Toc11698)

[What to test by unit tests on react component 25](#_Toc16920)

[What is render blocking code? 26](#_Toc14647)

[What is Prototype (in JS)? 26](#_Toc6417)

[Three different ways to check that the object is undefined. 27](#_Toc17229)

[What does npm build do? 27](#_Toc23277)

[How to use local npm package in application? 28](#_Toc13824)

[when we rerender component 28](#_Toc20512)

[Stripe API for payment 28](#_Toc3112)

[What is Pivot Table in Database? 29](#_Toc21387)

[Performance tool in React 29](#_Toc31454)

[Typescript any, null, never 29](#_Toc31899)

[typescript generics 29](#_Toc32694)

[JWT 30](#_Toc20259)

[Http LifeCycle 30](#_Toc16784)

[integrate testing 30](#_Toc18264)

[Capturing and Bubbling 30](#_Toc24904)

[Deep copy and shallow copy 31](#_Toc2644)

[http only and secure cookie 31](#_Toc137)

[CSRF attack and token 31](#_Toc2059)

[bearer token 32](#_Toc14478)

[normalization and document database 32](#_Toc414)

[Dependency tree in react 32](#_Toc18327)

[Strict Mode in React 33](#_Toc16224)

## -- What is Accessibility?

Web accessibility or eAccessibility, is the inclusive practice of ensuring there are no barriers that prevent interaction with, or access to, websites on the World Wide Web by people with physical disabilities, situational disabilities, and socio-economic restrictions on bandwidth and speed.

Four Major Categories of Accessibility

The Web Content Accessbility Guidelines(WCAG) are organized by four main principles, which state that content must be POUR: Perceivable, Operable, Understandable, and Robust

Features of Accessible Websites

Good use of HTML headings

Accessible with keyboard

Accessible images

Accessible menus.

Accessible forms

Accessible tables.

Effective use of color.

Meaningful link text

## -- What is SOLID principles?

<https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/>

1. Single responsibility Principle
2. Open-closed Principle
3. Liskov Substitution Priniciple
4. Interface Segregation Principle
5. Dependency Inversion Principle

Single-Responsibility Principle

Single-responsibility Principle states:

A class should have one and only one reason to change, meaning that a class should have only one job.

Open-Closed Principle

Objects or entities should be open for extension but closed for modification

Liskov Substitution Principle

Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.

This means that every subclass or derived class should be substitutable for their base or parent class.

Interface Segregation Principle

A client should never be forced to implement an interface that it doesn’t use, or clients shouldn’t be forced to depend on methods they do not use.

Dependency Inversion Principle

Entities must depend on abstractions, not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions.

## -- Typescript Utility Types

<https://www.typescriptlang.org/docs/handbook/utility-types.html>

Partial <Type>

Constructs a type with all properties of Type set to optional. This utility will return a type that represents all subsets of a given type.

Required <Type>

Constructs a type consisting of all properties of Type set to required. The opposite of Partial

Readonly <Type>

Constructs a type with all properties of Type set to readonly, meaning the properties of the constructed type cannot be reassigned.

Record<Keys, Type>

Constructs an object type whose property keys are Keys and whose property values are Type. This utility can be used to map the properties of a type to another type.

Pick<Type, Keys>

Constructs a type by picking the set of properties Keys(string literal or union of string literals) from Type.

Omit<Type, Keys>

Constructs a type by picking all properties from Type and then removing Keys (string literal or union of string literals)

Exclude<UnionType, ExcludedMembers>

Constructs a type by excluding from UnionType all union members that are assignable to ExcludedMembers.

Extract<Type, Union>

Constructs a type by extracting from Type all union members that are assignable to Union.

NonNullable<Type>

Constructs a type by excluding null and undefined from Type.

Parameters<Type>

Constructs a tuple type from the types used in the parameters of a function type Type.

ConstructorParameters<Type>

Constructs a tuple or array type from the types of a constructor function type. It produces a tuple type with all the parameter types (or the type never if Type is not a function)

ReturnType<Type>

Constructs a type consisting of the return type of function Type.

InstanceType <Type>

Constructs a type consisting of the instance type of a constructor function in Type.

ThisParameterType <Type>

Extracts the type of the this parameter for a function type or unknown if the function type has no this parameter

OmitThisParameter <Type>

Removes the this parameter from Type. If Type has no explicitly declared this parameter, the result is simply Type. Otherwise, a new function type with no this parameter is created from Type. Generics are erased and only the last overload signature is propagated into the new function type.

ThisType <Type>

This utility does not return a transformed type. Instead, it serves as a marker for a contextual this type. Note that the noImplicitThis flag must be enabled to use this utility

Intrinsic String Manipulation Types

Uppercase <StringType>

Lowercase <StringType>

Capitalize <StringType>

Uncapitalize <StringType>

## -- Socket

WebSocket is a computer communication protocol, providing full-duplex communication channels over a single TCP connection.

Unlike HTTP, where you have to constantly request updates, with websockets, updates are sent immediately when they are available. WebSockets keeps a single, persistent connection open while eliminating latency problems that arise with HTTP request/response-based methods

HTTPS - Hypertext Transfer Protocol Secure is an extension of the Hypertext Transfer Protocol

https is a combination of the Hypertext Transfer Protocol with the Secure Socket Layer(SSL)/ Transport Layer Security(TLS) protocol. TLS is an authentication and security protocol widely implemented in browsers and Web servers

## --Portals

Portals provide a first-class way to render children into a DOM node that exists outside the DOM hierarchy of the parent component.

A typical use case for portals is when a parent component has an overflow: hidden or z-index style, but you need the child to visually “break out” of its container. For example, dialogs, hovercards, and tooltips.

For MUI’s react portal

The portal component renders its children into a new “subtree” outside of current DOM hierarchy.

## -- This keyword

A function’s this keyword behaves a little differently in JavaScript compared to other languages. It also has some differences between strict mode and non-strict mode.

In most cases, the value of this is determined by how a function is called. It can’t be set by assignment during execution and it maybe different each time the function is called. ES5 introduced the bind() method to set the value of a function’s this regardless of thow it’s called, and ES2015 introduced arrow functions which don’t provide their own this binding(it retains the this value of the enclosing lexical context)

# -- Hoisting

<https://developer.mozilla.org/en-US/docs/Glossary/Hoisting>

JavaScript **Hoisting** refers to the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of the code.

Hoisting allows functions to be safely used in code before they are declared.

Variable and class declarations are also hoisted, so they too can be referenced before they are declared. Note that doing so can lead to unexpected errors, and is not generally recommended.

One of the advantages of hoisting is that it lets you use a function before you declare it in your code.

[var hoisting](https://developer.mozilla.org/en-US/docs/Glossary/Hoisting" \l "var_hoisting" \o "Permalink to var hoisting)

Here we declare then initialize the value of a var after using it. The default initialization of the var is undefined.

If we forget the declaration altogether (and only initialize the value) the variable isn't hoisted. Trying to read the variable before it is initialized results in ReferenceError exception.

[let and const hoisting](https://developer.mozilla.org/en-US/docs/Glossary/Hoisting" \l "let_and_const_hoisting" \o "Permalink to let and const hoisting)

Variables declared with let and const are also hoisted but, unlike var, are not initialized with a default value. An exception will be thrown if a variable declared with let or const is read before it is initialized.

[class hoisting](https://developer.mozilla.org/en-US/docs/Glossary/Hoisting" \l "class_hoisting" \o "Permalink to class hoisting)

Classes defined using a [class declaration](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes" \l "class_declarations) are hoisted, which means that JavaScript has a reference to the class. However the class is not initialized by default, so any code that uses it before the line in which it is initialized is executed will throw a ReferenceError.

# --Scope

<https://developer.mozilla.org/en-US/docs/Glossary/Scope>

The current context of execution. The context in which [values](https://developer.mozilla.org/en-US/docs/Glossary/Value) and **expressions** are "visible" or can be referenced. If a [variable](https://developer.mozilla.org/en-US/docs/Glossary/Variable) or other expression is not "in the current scope," then it is unavailable for use. Scopes can also be layered in a hierarchy, so that child scopes have access to parent scopes, but not vice versa.

A [function](https://developer.mozilla.org/en-US/docs/Glossary/Function) serves as a **closure** in [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript), and thus creates a scope, so that (for example) a variable defined exclusively within the function cannot be accessed from outside the function or within other functions.

**Scope chains establish the scope for a given function**. Each function defined has its own nested scope, and any function defined within another function has a local scope which is linked to the outer function — this link is called the chain.

Scope in JavaScript refers to the current context of code, which determines the accessibility of variables to JavaScript. The two types of scope are local and global:

# --Closure

<https://developer.mozilla.org/en-US/docs/Web/JavaScript/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.

# Blocking resource in CSS

By default, CSS is treated as a render blocking resource, which means that **the browser won't render any processed content until the CSSOM is constructed**. Make sure to keep your CSS lean, deliver it as quickly as possible, and use media types and queries to unblock rendering.

1. Identify your render blocking resources.
2. Don't use CSS imports.
3. Load conditional CSS with media attributes.
4. Defer non-critical CSS.
5. Use the defer and async attributes to eliminate render-blocking JavaScript.
6. Find and remove unused CSS and JavaScript.
7. Split code into smaller bundles.
8. Minify CSS and JavaScript files.

# Context api vs Redux

<https://dev.to/ruppysuppy/redux-vs-context-api-when-to-use-them-4k3p>

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

1. Create the ****Context****
2. Create a ****Provider**** for the ****Context****
3. Consume the data in the ****Context****

****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****, but since we are comparing it to ****Context API****, so we will stick to ****React-based Applications****.

# Long polling vs Standard polling

This variation of the traditional polling technique allows the server to push information to a client whenever the data is available. **With Long-Polling, the client requests information from the server exactly as in regular polling, but with the expectation that the server may not respond immediately**.

# Full duplex communication, Web sockets, Web socket handshake

The term full-duplex describes **simultaneous data transmission and receptions over one channel**. A full-duplex device is capable of bi-directional network data transmissions at the same time. Half-duplex devices can only transmit in one direction at one time.

WebSockets are one of many different tools for building web applications that **provide instant, real-time updates and communication**. The WebSocket Protocol establishes full-duplex, bidirectional communication between a client and server.

A WebSocket is **a standard bidirectional TCP socket between the client and the server**. The socket starts out as a HTTP connection and then "Upgrades" to a TCP socket after a HTTP handshake. After the handshake, either side can send data.

**Web Socket is a standard protocol for two-way data transfer between client and server**. The Web Socket protocol is built over TCP. Web sockets are mainly used to push messages to a client in real time updates. HTTP is a communication protocol of the world wide web.

First, the server must listen for incoming socket connections using a standard TCP socket. Depending on your platform, this may be handled for you automatically. For example, let's assume that your server is listening on example.com, port 8000, and your socket server responds to [GET](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET) requests at example.com/chat

The handshake is the "Web" in WebSockets. It's the bridge from HTTP to WebSockets. In the handshake, details of the connection are negotiated, and either party can back out before completion if the terms are unfavorable. The server must be careful to understand everything the client asks for, otherwise security issues can occur.

<https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers>

# Explain pagination, name web pages that uses pagination

*The [pagination](https://www.interaction-design.org/literature/topics/pagination" \o "What is Pagination?) design pattern is widely used in websites that open up a lot of content to users. From search engines such as Google and Firefox, to e-commerce sites such as Amazon and Asos—the use of pagination is very, very broad. And although it seems straightforward, implementing pagination in the right way has some very specific points that require your attention. That’s why we are about to show you everything you need to take into consideration when implementing this powerful pattern. Done in the right way, it can keep your users on the ‘same page’ as you but ready to devour reams more of inviting content.*

Pagination is used in some form **in almost every web application to divide returned data and display it on multiple pages within one web page**. Pagination also includes the logic of preparing and displaying the links to the various pages. Pagination can be handled client-side or server-side.

# Lazy loading - explain and how to implement; name web pages as an example of lazy loading

<https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading>

**Lazy loading** is a strategy to identify resources as non-blocking (non-critical) and load these only when needed. It's a way to shorten the length of the [critical rendering path](https://developer.mozilla.org/en-US/docs/Web/Performance/Critical_rendering_path), which translates into reduced page load times.

Lazy loading can occur on different moments in the application, but it typically happens on some user interactions such as scrolling and navigation.

Lazy loading can be applied to multiple resources and through multiple strategies.

**Code splitting** JavaScript, CSS and HTML can be split into smaller chunks. This enables sending the minimal code required to provide value upfront, improving page-load times. The rest can be loaded on demand.

* Entry point splitting: separates code by entry point(s) in the app
* Dynamic splitting: separates code where [dynamic import()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/import) statements are used

[JavaScript](https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading" \l "javascript" \o "Permalink to JavaScript)

**Script type module** Any script tag with type="module" is treated as a [JavaScript module](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Modules) and is deferred by default.

[CSS](https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading" \l "css" \o "Permalink to CSS)

By default, CSS is treated as a [render blocking](https://developer.mozilla.org/en-US/docs/Web/Performance/Critical_rendering_path) resource, so the browser won't render any processed content until the [CSSOM](https://developer.mozilla.org/en-US/docs/Web/API/CSS_Object_Model) is constructed. CSS must be thin, delivered as quickly as possible, and the usage media types and queries are advised to unblock rendering.

[Fonts](https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading" \l "fonts" \o "Permalink to Fonts)

By default, font requests are delayed until the render tree is constructed, which can result in delayed text rendering.

It is possible to override the default behavior and preload web font resources using <link rel="preload">, the [CSS font-display property](https://developer.mozilla.org/en-US/docs/Web/CSS/@font-face/font-display), and the [Font Loading API](https://developer.mozilla.org/en-US/docs/Web/API/CSS_Font_Loading_API).

See also: [Element Link](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/link)

[Images and iframes](https://developer.mozilla.org/en-US/docs/Web/Performance/Lazy_loading" \l "images_and_iframes" \o "Permalink to Images and iframes)

Very often, webpages contain many images that contribute to data-usage and how fast a page can load. Most of those images are off-screen ([non-critical](https://developer.mozilla.org/en-US/docs/Web/Performance/Critical_rendering_path)), requiring user interaction (an example being scroll) in order to view them.

# package.log explain (.log vs package..)

Package log **implements a simple logging package**. It defines a type, Logger, with methods for formatting output. It also has a predefined 'standard' Logger accessible through helper functions Print[f|ln], Fatal[f|ln], and Panic[f|ln], which are easier to use than creating a Logger manually.

# Devdependencies vs dependencies

A dependency is a library that a project needs to function effectively. DevDependencies are the packages a developer needs during development

The difference between these two, is that devDependencies are modules which are only required during development, while dependencies are modules which are also required at runtime. To save a dependency as a devDependency on installation we need to do an npm install --save-dev, instead of just an npm install --save.

<https://www.geeksforgeeks.org/difference-between-dependencies-devdependencies-and-peerdependencies/#:~:text=A%20dependency%20is%20a%20library,a%20developer%20needs%20during%20development.>

# Minification uglification, mapping (source maps)

Minification is **the process of minimizing code and markup in your web pages and script files**. It's one of the main methods used to reduce load times and bandwidth usage on websites. Minification dramatically improves site speed and accessibility, directly translating into a better user experience.

Minification is just removing unnecesary whitespace and redundant / optional tokens like curlys and semicolons, and can be reversed by using a linter.

Uglify JS is **a JavaScript library for minifying JavaScript files**. To 'uglify' a JavaScript file is to minify it using Uglify. Uglification improves performance while reducing readability.

A source map is **a file that maps from the transformed source to the original source**, enabling the browser to reconstruct the original source and present the reconstructed original in the debugger. To enable the debugger to work with a source map, you must: generate the source map.

# Monolith vs microservices

**A monolithic application is simply deployed on a set of identical servers behind a load balancer.** **In contrast, a microservice application typically consists of a large number of services**.

**Better performance is a crucial advantage of monolithic applications over microservices**. A microservice-based application may have to make 100 different API calls to 100 other microservices to load one UI screen.

In contrast to the microservices architecture, **monolithic applications are much easier to debug and test**. Since a monolithic app is a single indivisible unit, you can run end-to-end testing much faster. Simple to deploy

**Large Monolithic code-base (often spaghetti code) puts immense cognitive complexity on the developer's head**. As a result, the development velocity is poor. Granular scaling (i.e., scaling part of the application) is not possible. Polyglot programming or polyglot database is challenging.

# Advantage of micro services and microfrontends

https://www.redhat.com/en/blog/5-benefits-using-micro-frontends-build-process-driven-applications#:~:text=Micro%20frontends%20allow%20you%20to%20democratize%20user%20experience&text=Breaking%20up%20a%20frontend%20monolith,both%20frontend%20and%20backend%20capability.

Breaking up a frontend monolith provides enterprises an opportunity to more fully embrace a robust microservice-like philosophy. This approach allows cross-functional teams to work autonomously to deliver both frontend and backend capability

Micro frontends allow you to take a more iterative approach to updates

Micro frontends allow you to democratize user experience

Micro frontend codebases can be smaller, more manageable

Micro frontends promote reusability across process and case management

Micro frontends help you rapidly develop different views of your processes relevant to each person’s role

Micro frontends help you rapidly develop different views of your processes relevant to each person’s role

# Orchestration of micro services VS choreography of micro services

Microservices usually communicate using http (REST) or messaging/events. Orchestration is often associated with orchestration platforms that **allow you to create a scripted interaction among services to automate workflows**.

In choreography, **microservices works in parallel, unlike orchestration**. The entire system work on event-based architecture, where a service collects data from a message bus and perform the business logic and in return submit data to another message bus.

**The choreography describes the interactions between multiple services, where as orchestration represents control from one party's perspective**. This means that a choreography differs from an orchestration with respect to where the logic that controls the interactions between the services involved should reside.

Orchestration entails actively controlling all elements and interactions like a conductor directs the musicians of an orchestra, while choreography entails establishing a pattern or routine that microservices follow as the music plays, without requiring supervision and instructions.

There is no centralized logic. **Choreography possibilities may go farther beyond orchestration as it is more aligned with the real world**. My opinion is that we do not need to distinguish much between these two, as we need to focus on the business logic. Where a single point of logic does the job, we do orchestration.

# Oauth 2.0

The OAuth (open authorization) protocol was developed by the Internet Engineering Task Force and **enables secure delegated access**. It lets an application access a resource that is controlled by someone else (end user). This kind of access requires Tokens, which represent delegated right of access.

OAuth 2.0 is the industry-standard protocol for authorization. OAuth 2.0 focuses on client developer simplicity while providing specific authorization flows for web applications, desktop applications, mobile phones, and living room devices. This specification and its extensions are being developed within the [IETF OAuth Working Group](https://www.ietf.org/mailman/listinfo/oauth).

# Oauth (JSON webtoken)

JSON Web Token (JWT, [RFC 7519](https://tools.ietf.org/html/rfc7519)) is a way to encode claims in a JSON document that is then signed.

JWTs can be used as OAuth 2.0 [Bearer Tokens](https://oauth.net/2/bearer-tokens/) to encode all relevant parts of an access token into the access token itself instead of having to store them in a database.

Basically, JWT is a token format. **OAuth is an authorization protocol that can use JWT as a token**. OAuth uses server-side and client-side storage. If you want to do real logout you must go with OAuth2.

JWT and OAuth2 are entirely different and serve different purposes, but **they are compatible and can be used together**. The OAuth2 protocol does not specify the format of the tokens, therefore JWTs can be incorporated into the usage of OAuth2.

**If you want to do real logout you must go with OAuth2**. Authentication with JWT token can not logout actually. Because you don't have an Authentication Server that keeps track of tokens. If you want to provide an API to 3rd party clients, you must use OAuth2 also.

# Distributed tracing (time and correlationId)

Distributed tracing, sometimes called distributed request tracing, is **a method to monitor applications built on a microservices architecture**. IT and DevOps teams use distributed tracing to follow the course of a request or transaction as it travels through the application that is being monitored.

How distributed tracing works?

Applications may be built as monoliths or microservices. A monolithic application is developed as a single functional unit. In microservice architecture, an application is broken down into modular services, each of which handles a core function of the application and is often managed by a dedicated team.

Microservices are used to build many modern applications because they make it easier to test and deploy quick updates and prevent a single point of failure. But it can be challenging to troubleshoot microservices because they often run on a complex, distributed backend, and requests may involve sequences of multiple service calls. By using end-to-end distributed tracing, developers can visualize the full journey of a request—from frontend to backend—and pinpoint any performance failures or bottlenecks that occurred along the way.

End-to-end distributed tracing platforms begin collecting data the moment that a request is initiated, such as when a user submits a form on a website. This triggers the creation of a unique trace ID and an initial span—called the parent span—in the tracing platform. A trace represents the entire execution path of the request, and each span in the trace represents a single unit of work during that journey, such as an API call or database query. Whenever the request enters a service, a top-level child span is created. If the request made multiple commands or queries within the same service, the top-level child span may act as a parent to additional child spans nested beneath it. The distributed tracing platform encodes each child span with the original trace ID and a unique span ID, duration and error data, and relevant metadata, such as customer ID or location.

Finally, all of the spans are visualized in a [flame graph](https://www.datadoghq.com/knowledge-center/distributed-tracing/flame-graph), with the parent span on top and child spans nested below in order of occurrence. Since each span is timed, engineers can see how long the request spent in each service or database, and prioritize their troubleshooting efforts accordingly. Developers can also use the flame graph to determine which calls exhibited errors.

**Benefits and Challenges of Distributed Tracing**

According to a survey conducted by O’Reilly in 2020, [61 percent of enterprises](https://www.oreilly.com/radar/microservices-adoption-in-2020/" \t "https://www.datadoghq.com/knowledge-center/distributed-tracing/_blank) use microservice architecture. As that number grows, so does the need for distributed tracing and improved observability. Frontend engineers, backend engineers, and site reliability engineers use distributed tracing to achieve the following benefits:

****Reduce MTTD and MTTR****

If a customer reports that a feature in an application is slow or broken, the support team can review distributed traces to determine if this is a backend issue. Engineers can then analyze the traces generated by the affected service to quickly troubleshoot the problem. If you use an end-to-end distributed tracing tool, you would also be able to investigate frontend performance issues from the same platform.

****Understand service relationships****

By viewing distributed traces, developers can understand cause-and-effect relationships between services and optimize their performance. For example, viewing a span generated by a database call may reveal that adding a new database entry causes latency in an upstream service.

****Measure specific user actions****

Distributed tracing helps measure the time it takes to complete key user actions, such as purchasing an item. Traces can help identify backend bottlenecks and errors that are harming the user experience.

****Improve collaboration and productivity****

In microservice architectures, different teams may own the services that are involved in completing a request. Distributed tracing makes it clear where an error occurred and which team is responsible for fixing it.

****Maintain Service Level Agreements (SLAs)****

Most organizations have [SLAs](https://www.cio.com/article/2438284/outsourcing-sla-definitions-and-solutions.html" \t "https://www.datadoghq.com/knowledge-center/distributed-tracing/_blank), which are contracts with customers or other internal teams to meet performance goals. Distributed tracing tools aggregate performance data from specific services, so teams can readily evaluate if they’re in compliance with SLAs.

Despite these advantages, there are some challenges associated with the implementation of distributed tracing:

****Manual instrumentation****

Some distributed tracing platforms require you to manually instrument or modify your code to start tracing requests. Manual instrumentation consumes valuable engineering time and can introduce bugs in your application, but the need for it is often determined by the language or framework that you want to instrument. Standardizing which parts of your code to instrument may also result in missing traces.

****Head-based sampling****

Traditional tracing platforms tend to randomly sample traces just as each request begins. This approach results in missing and incomplete traces. With head-based sampling, businesses cannot always capture traces that are most relevant to them, such as high-value transactions or requests from enterprise customers. In contrast, some modern platforms can ingest all of your traces and rely on tail-based decisions, allowing you to capture complete traces that are tagged with business-relevant attributes, such as customer ID or region.

****Backend coverage only****

Unless you use an end-to-end distributed tracing platform, a trace ID is generated for a request only when it reaches the first backend service. You won’t have visibility into the corresponding user session on the frontend. This makes it harder to determine the root cause of a problematic request and whether a frontend or backend team should fix the issue.

# How to implement component will unmount in a functional component?

**How to manage componentWillUnmount with useEffect**

If you add a return function inside the useEffect function, it is triggered when a component unmounts from the DOM. This looks like:

import React, { useEffect } from 'react';

const ComponentExample => () => {

useEffect(() => {

return () => {

// Anything in here is fired on component unmount.

}

}, [])

}

# What is an example for using component will unmount?

The componentWillUnmount() method **allows us to execute the React code when the component gets destroyed or unmounted from the DOM (Document Object Model)**. This method is called during the Unmounting phase of the React Life-cycle i.e before the component gets unmounted.

**On clicking the 'Delete User' button, the User component will get unmounted from the DOM tree and before destroying the User component, the componentWillUnmount method will be called**.

# How to handle errors in React

Error handling with Error Boundaries — For class components. **Error boundaries are the most straightforward and effective way to handle errors that occur within your React components**. You can create an error boundary component by including the life cycle method componentDidCatch(error, info) if you use class component.

# HOC (higher order component) - explain, how/why to use

A higher-order component (HOC) is an advanced technique in React for reusing component logic. HOCs are not part of the React API, per se. They are a pattern that emerges from React’s compositional nature.

Concretely, ****a higher-order component is a function that takes a component and returns a new component.****

Whereas a component transforms props into UI, a higher-order component transforms a component into another component.

HOCs are common in third-party React libraries, such as Redux’s [connect](https://github.com/reduxjs/react-redux/blob/master/docs/api/connect.md" \l "connect" \t "https://reactjs.org/docs/_blank) and Relay’s [createFragmentContainer](https://relay.dev/docs/v10.1.3/fragment-container/" \l "createfragmentcontainer" \t "https://reactjs.org/docs/_blank).

# React DOM - explain, how/why to use

The react-dom package provides DOM-specific methods that can be used at the top level of your app and as an escape hatch to get outside the React model if you need to.

**createPortal()**

Creates a portal. Portals provide a way to [render children into a DOM node that exists outside the hierarchy of the DOM component](https://reactjs.org/docs/portals.html).

**flushSync()**

Force React to flush any updates inside the provided callback synchronously. This ensures that the DOM is updated immediately.

# Api interceptor/middleware - example

Api interceptor

Interceptor is **an API gateway server built for accepting API requests from the client applications and routing them to the appropriate backend services**. May it be a single service or multiple services to be called in a single API call, interceptor provides you with the necessary tools to control your API request flow.

Postman Interceptor is a Chrome extension that acts as a browser companion to the Postman Desktop app. Interceptor **enables you to capture network requests and cookies directly from a Chrome browser**. Once Interceptor is running in Chrome, you can start a debug session, which is a time-bound session of traffic capture.

# PWA - Progressive Web Application - explain

**Progressive Web Apps** (PWAs) are web apps that use [service workers](https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API), [manifests](https://developer.mozilla.org/en-US/docs/Web/Manifest), and other web-platform features in combination with [progressive enhancement](https://developer.mozilla.org/en-US/docs/Glossary/Progressive_Enhancement) to give users an experience on pair with native apps.

PWAs are web apps developed using a number of specific technologies and standard patterns to allow them to take advantage of both web and native app features. For example, web apps are more discoverable than native apps; it's a lot easier and faster to visit a website than to install an application, and you can also share web apps by sending a link.

On the other hand, native apps are better integrated with the operating system and therefore offer a more seamless experience for the users. You can install a native app so that it works offline, and users love tapping their icons to easily access their favorite apps, rather than navigating to it using a browser.

PWAs give us the ability to create web apps that can enjoy these same advantages.

It's not a brand new concept—such ideas have been revisited many times on the web platform with various approaches in the past. Progressive Enhancement and responsive design already allow us to build mobile friendly websites.

PWAs, however, provide all this and more without losing any of the existing features that make the web great.

# What is the library you use for unit testing?

JavaScript Unit Testing is **a method where JavaScript test code is written for a web page or web application module**. It is then combined with HTML as an inline event handler and executed in the browser to test if all functionalities are working as desired. These unit tests are then organized in the test suite.

**Jest** was the most popular JavaScript unit testing framework in 2020. For web apps that are based on React, Jest is the preferred framework. Apart from React, Jest supports unit testing of Angular, VueJS, NodeJS, and others.

import Enzyme, { shallow, mount } from 'enzyme';

import Adapter from '@wojtekmaj/enzyme-adapter-react-17';

import AlbumCard from './AlbumCard';

import faker from 'faker';

Enzyme.configure({ adapter: new Adapter() });

describe('AlbumCard', () => {

    const mockCallBackEdit = jest.fn();

    const mockCallBackDelete = jest.fn();

    const header = faker.internet.userName();

    const content = faker.internet.userName();

    const image = faker.internet.url();

    it('Renders AlbumCard without crashing', () => {

        shallow(<AlbumCard

            header = {header}

            content = {content}

            image = {image}

            albumIndex = {0}

            onEditAlbum = {mockCallBackEdit}

            onDeleteAlbum = {mockCallBackDelete}

        />);

    });

    it('Renders image, header, content and simulate click', () => {

        const wrapper = mount(<AlbumCard

            header = {header}

            content = {content}

            image = {image}

            albumIndex = {0}

            onEditAlbum = {mockCallBackEdit}

            onDeleteAlbum = {mockCallBackDelete}

        />);;

        expect(wrapper.find(`.MuiCardMedia-root`)).toHaveLength(1);

        expect(wrapper.find(`h2`).text()).toEqual(header);

        expect(wrapper.find(`p`).text()).toEqual(content);

        const editButton = wrapper.find(`button`).first();

        editButton.simulate('click');

        expect(mockCallBackEdit).toHaveBeenCalledTimes(1);

        const deleteButton = wrapper.find(`button`).at(1);

        deleteButton.simulate('click');

        expect(mockCallBackDelete).toHaveBeenCalledTimes(1);

    });

});

# If the story is not clear how do you approach that?

What is React Storybook? Storybook is **a development environment tool that is used as a playground for UI components**. It allows us, the developers, to create and test components in isolation. It runs outside of the app, too, so project dependencies won't affect the behaviour of components.

A story is a component with a set of arguments that define how the component should render. “Args” are Storybook’s mechanism for defining those arguments in a single JavaScript object. Args can be used to dynamically change props, slots, styles, inputs, etc. It allows Storybook and its addons to live edit components. You **do not** need to modify your underlying component code to use args.

When an arg’s value changes, the component re-renders, allowing you to interact with components in Storybook’s UI via addons that affect args.

Define Stories

Use the **named** exports of a CSF file to define your component’s stories. We recommend you use UpperCamelCase for your story exports. Here’s how to render Button in the “primary” state and export a story called Primary.

# How do you get familiar with a new code base?

1. Start from the top.
2. You do not need to understand the whole codebase.
3. Use the finished product yourself.
4. Run the code locally.
5. Ask questions.
6. Pair program.
7. Read and write documentation.
8. Take notes.

# What is SPA, and why is it important/good?

SPA stands for **Single Page Application**. It is a very common way of programming websites these days. The idea is that the website loads all the HTML/JS the first time you visit. When you then navigate, the browser will only rerender the content without refreshing the website.

**SPAs are reactive web applications that give a native look and feel without page loads**. SPAs owe this to AJAX/Client-Side Rendering typically provided by a client-side framework such as react/vue/angular. Many in the industry refer to more traditional web applications as Multi-Page Applications (MPAs).

# Difference between localstorage and cookie.

**Cookies are mainly for reading server-side, whereas local storage can only be read by the client-side** . Apart from saving data, a big technical difference is the size of data you can store, and as I mentioned earlier localStorage gives you more to work with.

# Can you create ErrorBoundary as functional component?

there's no way to turn a functional component into an error boundary

# Responsive layout vs Adaptive layout (mobile)

The responsive design will reconfigure all design elements whether it's viewed on a desktop, laptop, tablet, or mobile phone. With adaptive design, different fixed layouts are created that adapt to the users screen size.

# What is PropTypes?

PropTypes is **React's internal mechanism for adding type checking to components**. React components use a special property named propTypes to set up type checking. When props are passed to a React component, they are checked against the type definitions configured in the propTypes property.

PropTypes **exports a range of validators that can be used to make sure the data you receive is valid**. In this example, we're using PropTypes. string . When an invalid value is provided for a prop, a warning will be shown in the JavaScript console.

# What is web pack?

Webpack is a tool that **lets you compile JavaScript modules**, also known as module bundler. Given a large number of files, it generates a single file (or a few files) that run your app. It can perform many operations: helps you bundle your resources.

# What to test by unit tests on react component

****Rendering component trees**** in a simplified test environment and asserting on their output.

****Running a complete app**** in a realistic browser environment (also known as “end-to-end” tests).

# What is render blocking code?

Simply put, JavaScript is a piece of code that might be present on your website to enable some functions of your theme or plugins. And “Render Blocking” means that **these JavaScript codes are either blocking, or slowing down how your website is displayed, or rendered, by your browser**

By default, CSS is treated as a render blocking resource, which means that **the browser won't render any processed content until the CSSOM is constructed**. Make sure to keep your CSS lean, deliver it as quickly as possible, and use media types and queries to unblock rendering.

Tricks to eliminate render blocking resources

Identify your render blocking resources.

Don't add CSS with the @import rule.

Use the media attribute for conditional CSS.

Defer non-critical CSS.

Use the defer and async attributes to eliminate render-blocking JavaScript.

Find and remove unused CSS and JavaScript.

How to mock functions calls while testing with jest?

<https://jestjs.io/docs/mock-functions>

.mock, and test

# What is Prototype (in JS)?

The prototype is **an object that is associated with every functions and objects by default in JavaScript**, where function's prototype property is accessible and modifiable and object's prototype property (aka attribute) is not visible. Every function includes prototype object by default.

Prototypes are the mechanism by which JavaScript objects inherit features from one another. In this article, we explain what a prototype is, how prototype chains work, and how a prototype for an object can be set.

# Three different ways to check that the object is undefined.

-hasOwnProperty() method

const hero = {

name: 'Batman'

};

hero.hasOwnProperty('name'); // => true

hero.hasOwnProperty('realName'); // => false

-in operator

const hero = {

name: 'Batman'

};

'name' in hero; // => true

'realName' in hero; // => false

-Comparing with undefined

const hero = {

name: 'Batman'

};

hero.name !== undefined; // => true

hero.realName !== undefined; // => false

Using typof

# What does npm build do?

npm run build **creates a build directory with a production build of your app**. Set up your favorite HTTP server so that a visitor to your site is served index. html , and requests to static paths like /static/js/main.

# How to use local npm package in application?

run npm i -g local-install.

run npx install-local --save <local-path> inside the target repository to install the local dependency.

# when we rerender component

There's a checkbox well hidden in the React DevTools settings that allows you to visually highlight the components that rerendered. To enable it, go to "Profiler" >> click the "Cog wheel" on the right side of the top bar >> "General" tab >> Check the "Highlight updates when components render." checkbox.

React components automatically re-render **whenever there is a change in their state or props**. A simple update of the state, from anywhere in the code, causes all the User Interface (UI) elements to be re-rendered automatically.

(props, state, key)

# Stripe API for payment

The Stripe API is organized around [REST](http://en.wikipedia.org/wiki/Representational_State_Transfer" \t "https://stripe.com/docs/_blank). Our API has predictable resource-oriented URLs, accepts [form-encoded](https://en.wikipedia.org/wiki/POST_(HTTP)" \l "Use_for_submitting_web_forms" \t "https://stripe.com/docs/_blank) request bodies, returns [JSON-encoded](http://www.json.org/" \t "https://stripe.com/docs/_blank) responses, and uses standard HTTP response codes, authentication, and verbs.

You can use the Stripe API in test mode, which doesn't affect your live data or interact with the banking networks. The API key you use to [authenticate](https://stripe.com/docs/api" \l "authentication) the request determines whether the request is live mode or test mode.

Payouts, refunds, tokens

<https://api.stripe.com/v1/payouts>

<https://api.stripe.com/v1/refunds>

<https://api.stripe.com/v1/tokens>

# What is Pivot Table in Database?

A pivot table is **a statistics tool that summarizes and reorganizes selected columns and rows of data in a spreadsheet or database table to obtain a desired report**. The tool does not actually change the spreadsheet or database itself, it simply “pivots” or turns the data to view it from different perspectives.

# Performance tool in React

**In the React Developer Tools tab, there will be a tab called "Profiler"**. Click the record button to start recording performance data and, after using your app, click the "Stop" button.

# Typescript any, null, never

any: It is **a built-in data type in TypeScript which helps in describing the type of variable which we are unsure of while writing the code**. Though the data type 'any' is useful it shouldn't be used unnecessarily.

Null refers to **a value that is either empty or doesn't exist**. null means no value. To make a variable null we must assign null value to it as by default in typescript unassigned values are termed undefined.

TypeScript introduced a new type never , which **indicates the values that will never occur**. The never type is used when you are sure that something is never going to occur. For example, you write a function which will not return to its end point or always throws an exception.

# typescript generics

TypeScript Generics is **a tool which provides a way to create reusable components**. It creates a component that can work with a variety of data types rather than a single data type. It allows users to consume these components and use their own types.

# JWT

JWT, or JSON Web Token, is **an open standard used to share security information between two parties — a client and a server**. Each JWT contains encoded JSON objects, including a set of claims. JWTs are signed using a cryptographic algorithm to ensure that the claims cannot be altered after the token is issued.

# Http LifeCycle

Every Http request first contacts a DNS server which resolves the request URL domain to a IP address. After fetching the Webserver IP address request is forwarded to it(via PUT request). A webserver like apache handles this request and forwards this to application which has to handle this.

# integrate testing

Integration testing -- also known as integration and testing (I&T) -- is **a type of software testing in which the different units, modules or components of a software application are tested as a combined entity**. However, these modules may be coded by different programmers.

**Cypress is an open-source test runner built for the modern web** it makes integration testing easier and faster than before, with the help of cypress you can test any kind of web application. Cypress directly interacts with DOM which makes it independent of the web framework which the application is using.

# Capturing and Bubbling

Capturing phase – **the event goes down to the element.** **Target phase – the event reached the target element.** **Bubbling phase – the event bubbles up from the element**.

# Deep copy and shallow copy

**Deep copy stores copies of the object's value.** **Shallow Copy reflects changes made to the new/copied object in the original object**. Deep copy doesn't reflect changes made to the new/copied object in the original object. Shallow Copy stores the copy of the original object and points the references to the objects.

# http only and secure cookie

According to the Microsoft Developer Network, HttpOnly is **an additional flag included in a Set-Cookie HTTP response header**. Using the HttpOnly flag when generating a cookie helps mitigate the risk of client side script accessing the protected cookie (if the browser supports it).

HttpOnly and secure flags can be used to make the cookies more secure. **When a secure flag is used, then the cookie will only be sent over HTTPS, which is HTTP over SSL/TLS**.(secure sockets layer/transport layer security)

# CSRF attack and token

Cross-Site Request Forgery (CSRF) is **an attack that forces authenticated users to submit a request to a Web application against which they are currently authenticated**. CSRF attacks exploit the trust a Web application has in an authenticated user.

An attacker can use CSRF to obtain the victim's private data via a special form of the attack, known as login CSRF. **The attacker forces a non-authenticated user to log in to an account the attacker controls**. If the victim does not realize this, they may add personal data—such as credit card information—to the account.

In a successful CSRF attack, **the attacker causes the victim user to carry out an action unintentionally**. For example, this might be to change the email address on their account, to change their password, or to make a funds transfer.

A CSRF token is **a secure random token (e.g., synchronizer token or challenge token) that is used to prevent CSRF attacks**. The token needs to be unique per user session and should be of large random value to make it difficult to guess. A CSRF secure application assigns a unique CSRF token for every user session.

# bearer token

Bearer Tokens are **the predominant type of access token used with OAuth 2.0**. A Bearer Token is an opaque string, not intended to have any meaning to clients using it. Some servers will issue tokens that are a short string of hexadecimal characters, while others may use structured tokens such as JSON Web Tokens.

# normalization and document database

Normalization is **the process of organizing data in a database**. This includes creating tables and establishing relationships between those tables according to rules designed both to protect the data and to make the database more flexible by eliminating redundancy and inconsistent dependency.

With relational database it is not a problem

But for document-oriented database it can be a problem

In MongoDB, you probably don't want to store data this way. It requires a lot of queries to get the data you want. To have a user's informations, you'll need to do 3 trips to the database. One for users, one for accountsPref, another for articles.

You don't want three trips to the database to get your informations. We could store the accounts preferences of each user as an embedded document.

The advantage of this is that you need one less query to get the information. The downside is that it takes up more space and is more difficult to keep in sync. For example, we decide that the light style should be renamed day. We would have to update every single document where the user.accountsPref.style was light.

You can use an hybrid of referencing and embedding. You could keep the subdocument, you add the reference of your account preferences but you only put the most frequently used fields in the subdocument. If you know that only the style field is frequently used by your app

This can be a nice approach because your requirements may change over time. If you want to include more or less info on the page, you can always add or remove fields from the embedded document.

# Dependency tree in react

A dependency tree for a sentence is **a directed acyclic graph with words as nodes and relations as edges**. Each word in the sentence either modifies another word or is modified by a word. The root of the tree is the only entry that is modified but does not modify anything else.

**Use the npm list to show the installed packages in the current project as a dependency tree**. Use npm list --depth=n to show the dependency tree with a specified depth. Use npm list --prod to show packages in the dependencies .

# Strict Mode in React

StrictMode is **a tool for highlighting potential problems in an application**. Like Fragment , StrictMode does not render any visible UI. It activates additional checks and warnings for its descendants. Note: Strict mode checks are run in development mode only; they do not impact the production build.