JS questions:

https://github.com/leonardomso/33-js-concepts

1. What is variable hoisting
2. What is memory heap and stack
3. What is a null pointer?
4. Difference between setTimeout and setImmediate?
5. What are some of the ES6 features in JS
    a. arrow functions
    b. class syntaxan
    c.
    d. destructuring
    e. spread operators
    f. let and const
    g. promises
    h. map and reduce
6. how arrow function is different from normal function
7. usage of this. on arrow function
8. what is promise
9. what is the need of promises
10. what if we dont care if promise resolve or reject, how to handle it
11. how to call second api call on success of first api call
12. what is promise chaining
13. what are the closures
14. what is the use of closures
15. How to cancel and axios ongoing call?
16. What are interceptors in axios
17. What are mixins in SASS
18. Is JS multithreaded
19. Diff between for and map
20. What is the usage of !! before any variable
21. How is memory managed in JS
    a. their memory is managed by the Garbage Collector – a background process that constantly reviews objects and modules and deallocates memory from the ones that are not referenced directly or indirectly from root objects.

React Native questions:

1. do we use closures in react?

2. which version of react u using
3. what is virtual dom
4. what is reconcilation method
5. react lifecycle
6. What is the difference between DOM and VirtualDOM
7. What are the different ways of state management used in React (local, cross component, app-wide)
8. What are the limitations of Context API

- *Syntax is complex*
- *Not suitable for rapidly changing data*
- *Complex to setup/management*
- *Nested markup tags etc*
- *Makes component reuse more difficult*

9. What is prop drilling
22. What is the difference between hooks useCallback and useMemo
23. What is the difference between Component and PureComponent
24. What is the difference between Component and functional component
25. usage of useref
26. what is forwardref
27. What is usage of useimperativehandle
28. give names of some performance hooks
29. What is the rule of hooks
    a. Only use them in functional component
    b. Dont call them in nested functions
    c. Dont call them in any block statements rather only at the top level
30. what is the replacement of usecallback and usememo in class components
31. what is a portal
32. How do we apply validation on props
33. What are the errorboundaries
34. Explaining the core principles of Redux and its flow
35. is redux state immutable
36. how are you maintaining immutable state
37. Which well-known hooks are used while working with redux
38. Explain the concept of pure functions and their significance in redux reducers
39. Describe the difference between the main axis and the cross axis in Flexbox.
40. How does Flexbox handle the arrangement and alignment of elements within a container?
41. Describe memory leak and how to fix it
42. What are the demerits of using PureComponent?
43. why we use flatlist
44. Difference between flatlist and scrollview
45. what is the diff btw button and pressable
46. diff between touchableopacity and pressable
47. How Different is React-native from ReactJS ?

- *Web vs mobile*
- *Npm libs are different*
- *Different view layer*
- *Build process is different*
- *Storage mechanisms are different*
- *Styling is different*
- *Machines are diff*
- *Learning curve is diff*

48. What is Flexbox and describe any elaborate on its most used properties?
49. Describe advantages of using React Native over native?
50. What are threads in General ? and explain Different Threads in ReactNative with Use of Each ?
51. Are default props available in React Native and if yes for what are they used and how are they used ?
52. What is the difference between reactjs and react native
53. What is setNativeProps?
54. What is Yoga
55. What is the diff in useEffect and useLayoutEffect
56. When to use Redux vs Context vs useReducer
57. Usage of onLayout
58. How to fix issues in 3rd party libs
59. How would you integrate a React Native module into an existing iOS application?
60. Describe a situation where you had to debug a complex React Native issue on iOS. How did you approach it?
61. Can you explain the process of bridging native iOS modules to React Native? What challenges have you faced while doing this?
62. How do you ensure a smooth animation experience in a React Native app?
63. What is the difference between React Native's useEffect hook and the componentDidMount lifecycle method in native iOS?
64. How would you optimize the performance of a React Native app on iOS, especially when dealing with large data sets or animations?
65. Describe how you would manage navigation in a React Native app. Which libraries would you use, and why?
66. Have you ever dealt with issues related to memory management or performance bottlenecks in a React Native app on iOS? How did you resolve them?
67. Explain how React Native interacts with the native iOS layer. What role does the JavaScript bridge play?
68. What is the difference between Expo and bare React Native CLI, and when would you recommend each?
69. How do you handle platform-specific code in a React Native project, particularly when there are iOS-specific requirements?
70. Describe a scenario where you needed to implement a custom native UI component for React Native. What steps did you take?

71. How do you test React Native applications on iOS devices? What tools or frameworks do you use?
72. How would you implement push notifications in a React Native app for iOS? What libraries or tools would you use?
73. Can you explain how Flexbox works in React Native? How does it differ from AutoLayout in iOS development?

Torchering questions

1. anything that you have done to improve your work or team performance
2. what is your approach to solving a slow application, your approach to debugging and rectifying the problem
3. what is your approach with refactoring? design, duplication?
4. what code smells you usually look after?
5. how you support your team
6. how do you fix technical bottlenecks
7. have u taken any react native project to production
8. have u been through any release process
9. what is the release process in android and ios
10. what architectural patterns have you used in react native
    - Higher Order Component
    - Provider Design Pattern
    - Compound Component
11. what are your practices in writing a clean, maintanable and efficient code
12. how confident are you in learning new technologies, which was the last new thing you learned and in how much time

Third party tools:

1. React Navigation
2. Redux or MobX
3. Axios
4. AsyncStorage
5. React Native Elements
6. React Native Paper

7. React Native Gesture Handler
8. React Native Vector Icons
9. React Native Firebase
10. Lottie React Native
11. Mmkv
12. WatermelonDB
13. Realm
14. Googlemaps
15.
16. Victory charts
17. Stripe React Native - https://github.com/stripe/stripe-react-native
18. Zendesk
19. Datadog RUM (`@datadog/mobile-react-native`)

React State management libs:
Redux
MobX
Zustand: https://github.com/pmndrs/zustand
Legend State: https://github.com/LegendApp/legend-state

Raw questions from interviews:

What is clean architecture?
Divide the system into layers to isolate logic and responsibilities.
Higher-level modules should not depend on lower-level modules; both should depend on abstractions.
Business logic should be at the core and independent of external frameworks or databases.
The domain layer should remain pure and independent of the infrastructure and interface.
The architecture promotes writing testable code by isolating business logic from external systems like databases and UI.
The application should not be tightly coupled with frameworks, allowing for easier refactoring or migration.

- It is a layered Architecture like an onion . I have worked on it using protocols and repository layers
Has 3 layers
presentaion layer
domain layer
and data layer
- prentaoin layer is like MVVM
where we wrtei UI and UI logic

and domain layer- have businns logic
which is domain specific
and data layer- has repsitory and network layer
- VIPER best fits int this role for clean architecture
but I have used MVVM with repository and coroditnaor
to fit in clean architecture
- The architecture separates the application into distinct layers, each responsible for a specific concern
- So in these your core idea of app/project doesn't get modify but your outer layers like, UI layer, analytics layer gets modified frequently
- In Clean Architecture focus is more on having more seprate layers and following SOLID principle

- The business logic can be tested without the UI, database, web server or any other external element.

- It is a layered Architecture like an onion . I have worked on it using protocols and repository layers
- So in these your core idea of app/project doesn't get modify but your outer layers like, UI layer, analytics layer gets modified frequently
- The business logic can be tested without the UI, database, web server or any other external element.
- It is a pattern that promotes separation of concerns and testability and makes your code more scalable.
- This architecture is more closer towrds following SOLID
- The advantages include improved code organization, easier testing, independence from frameworks, scalability, maintainability, adaptability, code reusability, and future-proofing.

What is your branching strategy?
What is code quality?

- following SOLID
- looslely coupled code
- now hard codeing
- no big functions
- no complex logic
- look for proper dependency injection
- coding style is followed
- good test code coverage
- have no code smells

What is the purpose of custom hooks?
What is the difference between types and interfaces in typescript
What is strictNullChecks and notImplicitAny
How RTK query signature looks like and how it works behind the scenes

What is the difference between promises and async-await, which one is performant
What is JWT token
What does JWT token consist of
Where do you configure strictNullCheck
What are code smells in react native

- Large Components
- Excessive Nesting
- Complex State Logic
- Overuse of setState
- Large Render Methods
- Overuse of forceUpdate
- Excessive Use of Any Types
- Inconsistent Coding Style
- Unnecessary Inline Styles
- Unmanaged Side Effects
- Unused Variables or Imports
- Overuse of Redux or Context

How to do image optimization in react native

- Use the Right Image Format
- Resize Images
- Compress Images
- WebP Format
- React Native Asset Optimization
- Use Image Caching Libraries
- Lazy Loading
- Use SVGs for Scalable Icons
- Image Caching
- CDN for Image Delivery

How many threads in react native
How to implement responsive web design

- Fluid Grids
- Flexible Images and Media
- Media Queries
- Breakpoints
- Mobile-First Approach

How do u test the sagas
What happens in react when virtual dom is not there

Without the virtual DOM, React would lose many of its key performance benefits, such as batching updates, minimizing DOM manipulation, and efficiently updating only the parts of the DOM that have changed. As a result, the user experience might suffer, especially in applications with complex UIs or frequent updates. Additionally, without the virtual DOM, React's reconciliation algorithm, which efficiently updates the UI based on changes to the application state, would not be as effective. Overall, the absence of the virtual DOM would likely lead to poorer performance and a less responsive user interface.

Descriptive Questions:

MVVM vs Flux:

- Data Binding: MVVM uses two-way data binding, while Flux uses unidirectional data flow.
- Components: MVVM includes View, ViewModel, and Model. Flux includes Actions, Dispatcher, Stores, and Views.
- Communication: In MVVM, the ViewModel directly communicates with the View. In Flux, communication flows through a central dispatcher, ensuring unidirectional data flow.
- Origin and Use Cases: MVVM originated in the Microsoft ecosystem and is common in desktop and mobile applications. Flux originated with React for web applications.

What are the different optimization techniques used in React

Use FlatList or VirtualizedList: Utilize FlatList or VirtualizedList for long lists, as they are optimized for performance and memory usage.

Avoid using inline styles: Use StyleSheet for styling components, as it is more efficient than inline styles.

Reduce Re-renders: Use shouldComponentUpdate, PureComponent, or React.memo to prevent unnecessary re-renders of components.

Use Native Components: Use native components for specific functionalities that require better performance, such as animations, maps, or image processing.

Optimize Images: Optimize images by using tools like Image Resizer, or use a library like React Native Fast Image for better handling of images.

Minimize Touchable Opacity: Instead of using Touchable Opacity for every component, consider using specific touchable components, like TouchableHighlight or TouchableWithoutFeedback, where appropriate.

Memory Management: Keep an eye on memory usage, avoid memory leaks, and use the debugging tools provided by React Native and third-party libraries to identify and resolve memory-related issues.

Code Splitting: Divide the code into smaller chunks and load only the necessary components or modules at runtime to reduce the initial load time.

Use Interaction Managers: Use InteractionManager to execute heavy tasks when the app is idle and ensure a smooth user experience.

Profiler and Performance Tools: Utilize the React Native Performance Monitor and other profiling tools to identify performance bottlenecks and optimize app performance.

React Native Hermes: Consider using Hermes, an open-source JavaScript engine optimized for running React Native apps on Android, to improve the app's start-up time and overall performance.

Update to the latest version of React Native: Always keep your React Native version up to date to benefit from the latest performance improvements and bug fixes.

Avoid unnecessary re-renders: Use techniques such as memoization and state management tools like Redux or Context API to minimize re-renders.

Others:

Use usememo, memo and usecallback
Remove console logs
Use code splitting and lazy loading
Modifying lifecycle events life shouldcomponentupdate
Selecting purecomponent implication where appropriate

What are the possible security measures could be taken in the app?

MobSF
npm audit
Ssl
Device not jailbroken
Encrypted bundle
ssl pinning
encryption at rest
encryotion in transit
authentication & authorization
third party api upgrades and exploits
biometric security
multi factor authentication
save keys and environment variables
secure coding
hiding logs & error details in production
meeting compliances (GDPR, HIPAA, PCI DSS)
Avoid Hardcoding Sensitive Data
Sanitize User Input
Regular Security Audits and Testing
Server-Side Security
    Implement Rate Limiting
    Replay attacks
Code analysis using Veracode in CI/CD

Compliances:
CWE - common weakness enumeration
GDPR - general data protection regulation
NIAP - national information assurance partnerships

Code Obfuscation
    Proguard for android
    iXGuard for iOS

Bank grade security:

Strong Authentication
Secure Password Policies
OAuth and OpenID Connect
End-to-End Encryption
Key Management
TLS/SSL
Certificate Pinning
Secure Code Practices
Obfuscation
Integrity Checks
Root/Jailbreak Detection
Secure Storage
Network Anomaly Detection

Minimize Data Collection
User Consent
Data Anonymization
Security Audits
Penetration Testing
Monitoring and Logging
Incident Response Plan
Security Awareness
In-App Notifications
Logout on inactivity
Geolocaiton checks
Device specific checks

Difference between accesstoken and refreshtoken

Access tokens have limited lifetimes. If your application needs access to a Google API beyond the lifetime of a single access token, it can obtain a refresh token. A refresh token allows your application to obtain new access tokens.

Save refresh tokens in secure long-term storage and continue to use them as long as they remain valid. Limits apply to the number of refresh tokens that are issued per client-user combination, and per user across all clients, and these limits are different. If your application requests enough refresh tokens to go over one of the limits, older refresh tokens stop working.

whats the new thing in react 18

- Concurrent Rendering
- New Hooks or Features
- Improved Error Handling and Diagnostics
- Optimizations and Performance Improvements
- Improved Server-Side Rendering (SSR)

what is virtual dom

The Virtual DOM (VDOM) is a concept and technique used in web development, particularly in frameworks like React, to improve the performance and efficiency of updating the user interface (UI). It is an abstraction of the actual DOM (Document Object Model) present in the browser.

what is reconcilation method

Reconciliation: When there are updates to the state or props of a React component, React calculates the difference between the previous Virtual DOM and the new Virtual DOM. This process is called "reconciliation" or "diffing."

HOC

Higher-Order Components (HOCs) are an advanced technique in React for reusing component logic. An HOC is a function that takes a component and returns a new component with enhanced behavior or props. This pattern is useful for cross-cutting concerns such as logging, error handling, and state management.

Deep linking

Deep linking refers to the process of using a hyperlink that directs users to a specific piece of content within a mobile application rather than simply launching the app. It allows users to access specific pages or features directly, which can enhance user experience, streamline navigation, and improve the app's usability.

what are the closures

- A function is defined within another function (nested function).
- The inner function references variables or parameters from the outer function.
- Closures allow functions to retain access to variables from their outer (enclosing) scope even after the outer function has finished executing.

do we use closures in react?

- In React (and React Native), closures are often used for maintaining state, handling event handlers, and managing the flow of data between components.

what is promise

Promises are a feature in JavaScript introduced with ECMAScript 6 (ES6) that provide a cleaner and more structured way to handle asynchronous operations.

A Promise is an object representing the eventual completion (or failure) of an asynchronous operation and its resulting value. It has three possible states:

- pending
- success
- failure

Handling errors in RN:

errors:

native side
js side

errors in user forms
try catch
error boundaries
send it to sentry
react-native-exception-handler

Software Development Methodologies:

There are various software development methodologies, each with its own approach to organizing, planning, and executing the process of developing software. Here are some commonly used methodologies:

- Waterfall Model: A linear and sequential approach where development is divided into phases, and each phase must be completed before the next one begins.
- Agile: An iterative and flexible approach that emphasizes collaboration, customer feedback, and the ability to respond to changes quickly. It often involves breaking the project into smaller, manageable tasks.
- Scrum: A subset of Agile, Scrum is an iterative and incremental framework that uses time-boxed iterations called sprints, typically lasting two to four weeks, to deliver a potentially shippable product increment.
- Kanban: A visual framework that focuses on continuous delivery and encourages incremental changes to the system. Work items are represented visually on a board, and the team collaborates to move items through various stages.
- Lean Development: Derived from manufacturing principles, Lean Development aims to eliminate waste, improve efficiency, and deliver value to the customer. It emphasizes continuous improvement and reducing unnecessary work.
- DevOps: More of a culture and set of practices than a strict methodology, DevOps aims to bridge the gap between development and operations teams, promoting collaboration, automation, and continuous delivery.
- Feature-Driven Development (FDD): A model that organizes software development around making progress on features. It is iterative and involves creating a feature list, planning by feature, and designing and building features in a short time frame.
- Extreme Programming (XP): An Agile methodology that focuses on improving software quality and responsiveness to changing customer requirements through frequent releases in short development cycles, paired programming, and continuous testing.
- Rapid Application Development (RAD): Emphasizes quick development and iteration, often with user feedback gathered during the development process. RAD typically involves prototyping and iterative testing.
- Incremental Model: Similar to the Waterfall model, but with cycles of development and refinement. Each iteration adds new functionality, building on the previous version.
- Spiral Model: Combines elements of the Waterfall model and prototyping in an iterative manner. It includes risk analysis and allows for changes during the development process.
- Minimum Viable Product (MVP): A development technique where a new product or website is developed with sufficient features to satisfy early adopters. The final, complete set of features is only designed and developed after considering feedback from initial users.

These methodologies can be adapted and combined based on the project's specific requirements and the organization's preferences.

Api error codes

Behavioral Questions:

1. Give me an example of a time you had a conflict with a team member. How did you handle it?
2. Tell me about a time you made a mistake at work. How did you resolve the problem, and what did you learn from your mistake?
3. Describe an occasion when you had to manage your time to complete a task. How did you do it?
4. Describe an occasion when you failed at a task. What did you learn from it?
5. Tell me about a time you took the initiative in your career. What was your motivation for doing so?
6. Describe a time when you used your leadership skills to motivate your team or colleagues.
7. Describe a time when you were responsible for a task you didn't receive training on and were unsure how to complete. How did you handle it?
8. Share an example of a career goal you had. What steps did you take to achieve it?
9. Give an example of a time when you had to make a difficult decision. How did you handle it?
10. Describe your process for solving problems. What steps do you take to resolve important issues at work?
11. Why r u looking for a new role?
12. what gets u most excited about work
13. whats most exciting about this role
14. how many times that u missed a deadline
15. where do u want to be in the next 5 years
16. how do u prefer to communicate with the co workers
17. how do u give critics and how do u recieve
18. can u share an example when u collaborated your colleages with diverse background to achieve a common goal
19. what management style make u to do the best work
20. what is the last time u took the risk professionally
21. Whats the best and worst team building exercise u have done
22. How do u build rapport with your team
23. How do commit to a positive team environment

OWASP

Broken Access Control

Cryptographic Failures

Injection
Insecure Design
Security Misconfiguration
Vulnerable and Outdated Components
Identification and Authentication Failures
Software and Data Integrity Failures
Security Logging and Monitoring Failures
Server-Side Request Forgery

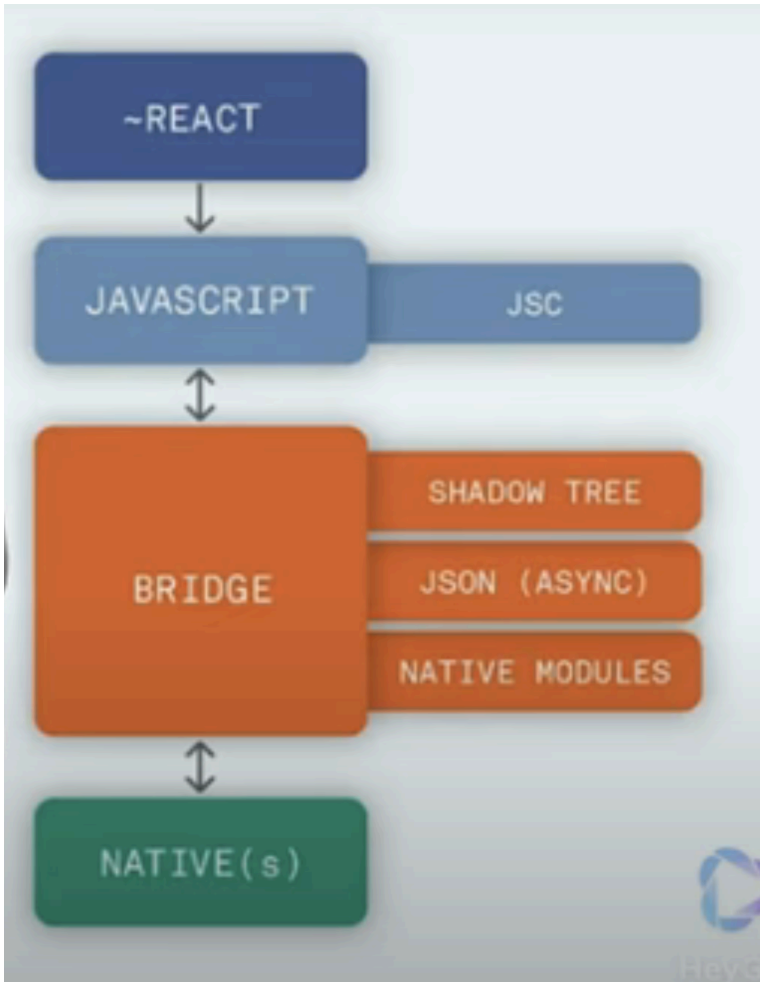
Benefits of fabric native modules:

Type Safety
Shared C++ Core
Better Host Platform Interoperability
Improved Performance
Consistency
Faster Setup
Less serialization of data between JS and host platform

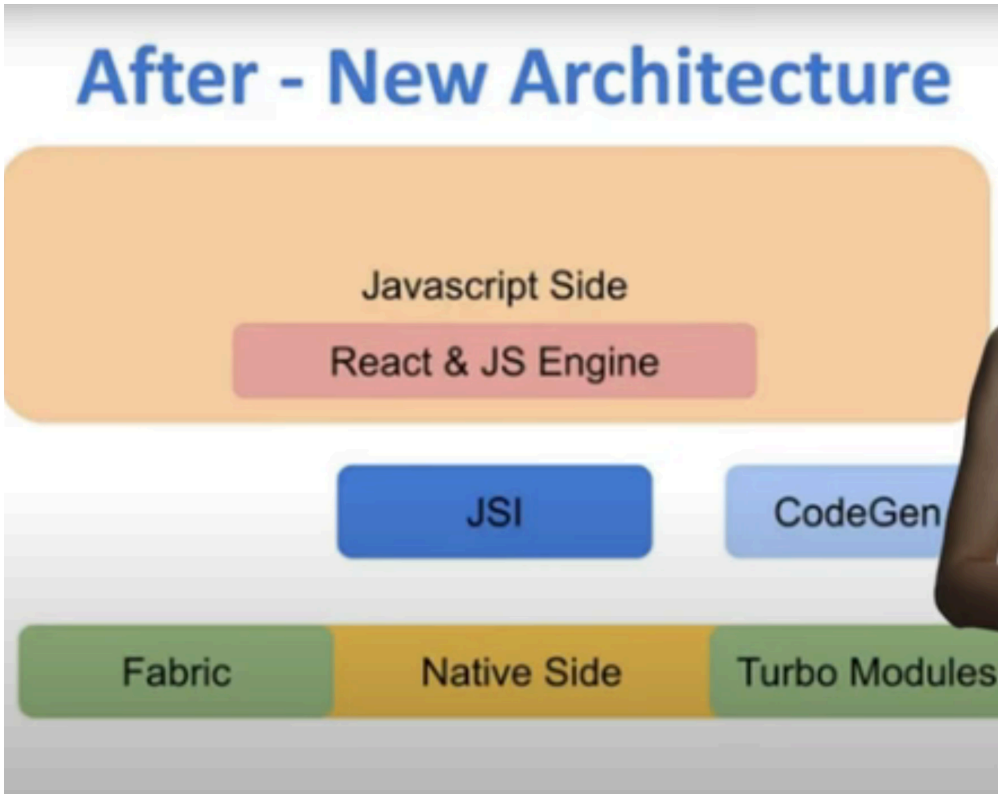
Old Architecture

React Native previously used a **bridge** to communicate between JavaScript and native modules. The Modules are written in C++, Objective C, Java, or kotlin to access native features like cameras, sensors, etc. Unfortunately, the **Bridge** has some limitations.

One main limitation is that each time one layer communicates with another, it involves serializing (converting JS Object to JSON String) and deserializing (converting JSON String back to JS Object) data. Since the conversion takes time, this process adds a performance issue to the communication flow.



New Architecture



How many Node.js libraries are there?

Multiple Node.js libraries are created to serve a purpose and have specific functionality. There are more than 83 Node.js libraries available that helps developers to create more robust web applications.

What is the difference between the Node.js library and the framework?

A library helps with the complexity and focuses more on the reuse of code, whereas a framework reduces complexities for developers. A framework has abstract away behaviour, logic and even architectural patterns that can be used in making a new project. One of the most important technical difference between library and framework are called the inversion of control. In a library you are in charge of the application flow. You can choose where and when you can call the library. But when using a framework, it is in charge of the workflow.

What are NPM libraries?

NPM libraries are the world's; largest software registry and contain over 800,000 codes. All open-sourced developers use npm to share software details. Multiple organizations use npm to manage their private development. There are more than 1.3 million packages available in the main npm registry.

Is react a node library?

No. Both Node.js and React.js are different. Node.js is mainly used for the backend development with the JavaScript technology where as React.js is used for front end development.

React is a JavaScript library that is used to help in the creation of user interfaces. This was authored by Facebook and Instagram engineers. It was aimed to help solve the problems involved when developing a complex user interface with databases that keep changing over time.

Day to day tasks as Lead

1. Team Coordination and Management:
    - Start the day with a team stand-up meeting to discuss progress, blockers, and goals for the day.
    - Assign tasks and provide guidance to team members based on project priorities and individual strengths.
    - Address any team concerns or conflicts and work towards resolutions.
2. Technical Planning and Architecture:
    - Review the project requirements and plan the technical approach to meet them using React Native.
    - Collaborate with other leads or stakeholders to define project milestones, timelines, and technical requirements.
    - Design the overall architecture of the React Native application, including data flow, state management, and component structure.
3. Coding and Development:
    - Spend time coding and implementing features, especially those that require complex logic or optimization.
    - Review code submitted by team members, providing feedback and ensuring adherence to best practices and project standards.
    - Troubleshoot and debug issues as they arise, both independently and by assisting team members.
4. Testing and Quality Assurance:
    - Oversee the testing strategy for the React Native application, including unit tests, integration tests, and end-to-end testing.
    - Conduct code reviews to ensure code quality and identify potential areas for improvement.
    - Work closely with QA engineers to address any reported bugs or issues and ensure a smooth testing process.
5. Documentation and Knowledge Sharing:
    - Document important design decisions, architectural patterns, and code implementations for future reference.

- Share knowledge and best practices with team members through code reviews, tech talks, or documentation sessions.
- Stay up-to-date with the latest React Native developments and share insights with the team to improve efficiency and maintainability.

6. Client Communication:
    - Communicate regularly with clients or project stakeholders to provide updates on progress, discuss requirements, and gather feedback.
    - Manage client expectations regarding project timelines, scope changes, and deliverables.
    - Address any client concerns or requests in a timely and professional manner.
7. Continuous Improvement:
    - Reflect on the day's accomplishments and challenges during team retrospectives.
    - Identify areas for process improvement or optimization within the development workflow.
    - Proactively seek opportunities to enhance the team's skills and knowledge through training, workshops, or external resources.

What is Flux Design Pattern?

The Flux pattern is an architectural pattern used primarily in building user interfaces, particularly with frameworks like React. It was created by Facebook to address some of the challenges they faced while managing data flow in large-scale React applications.
At its core, Flux is based on a unidirectional data flow, which means that data has a single source of truth and flows in one direction through the application. This helps in managing state in complex applications and reduces the risk of data inconsistency and hard-to-debug issues.

What are Web Hooks?

Webhooks are user-defined HTTP callbacks that enable real-time communication between web applications. They allow one application to send data to another application in near real-time as soon as a particular event occurs. Webhooks are commonly used for automation, integration, and notification purposes in web development.

What is consent management platform?

A Consent Management Platform (CMP) is a tool or system used by organizations to manage user consent preferences regarding the collection and processing of their personal data.

App Rejection reasons

App Crashes or Bugs
Incomplete Information or Misleading Claims
Violation of App Store Guidelines
Poor User Experience
Privacy Concerns
Copyright or Trademark Infringement
Incomplete Functionality
Performance Issues
Non-compliance with App Review Guidelines Updates
Repetitive or Similar Apps

Testing strategy:

Unit Testing with Jest
Component Testing with React Testing Library
Integration Testing
Snapshot Testing
Redux Testing using redux-mock-store
Async Testing using jest-fetch-mock
End-to-End Testing with Detox
UI Regression testing using chromatic and storybook
UI regression testing using react native owl

Agile:
Customer satisfaction through early and continuous software delivery
Welcome changing requirements, even late in development
Deliver working software frequently, with a preference for shorter timescales
Collaboration between business people and developers throughout the project
Build projects around motivated individuals, giving them the environment and support they need, and trust them to get the job done
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation
Working software is the primary measure of progress
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely
Continuous attention to technical excellence and good design enhances agility
Simplicity--the art of maximizing the amount of work not done--is essential
The best architectures, requirements, and designs emerge from self-organizing teams

At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly

AB Testing:
A/B testing, also known as split testing, is a method used in marketing, product management, and user experience design to compare two versions of a webpage or app to determine which one performs better. The goal is to identify changes that improve a given metric or outcome, such as conversion rates, click-through rates, or user engagement.

Create Variations (A and B)
Randomly Assign Visitors/Users
Collect Data
Statistical Analysis
Draw Conclusions

flatlist optimization

- Key Extractor - Ensure each item in your FlatList has a unique key by implementing the keyExtractor prop. This helps React Native optimize re-rendering of items.
- Avoid Anonymous Functions, rather define outside render
- Memoize renderitem using usecallback to avoid reevaluation
- initialNumToRender - Set initialNumToRender to an optimal number to reduce the initial load time.
- windowSize - Adjust windowSize to control how many items are rendered outside the visible area. A larger window size can reduce flickering but uses more memory.
- Set removeClippedSubviews to true to unmount items outside the viewport.
- If items have a fixed height/width, implement getItemLayout to avoid calculating positions dynamically.
- Implement pagination (infinite scrolling) to load data in chunks rather than all at once.
- Ensure that FlatList props like extraData are stable and only change when necessary.

Button vs Pressable:

Purpose:

Button: Designed for creating standard buttons with minimal configuration.
Pressable: Designed for creating customizable interactive components with detailed touch feedback.
Styling:

Button: Limited styling options (mainly color).
Pressable: Extensive styling options, allowing custom styles for different states.
Functionality:

Button: Primarily supports a single onPress event.
Pressable: Supports multiple interaction states and events like onPressIn, onPressOut, onLongPress.
Usage:

Button: Quick and easy for standard buttons.
Pressable: Better suited for custom buttons or other interactive components requiring advanced touch handling and styling.

TouchableOpacity vs Pressable:

Purpose:

TouchableOpacity: Simple component that provides an opacity change on press for basic visual feedback.
Pressable: Flexible component designed for advanced touch interactions and custom behaviors.
Styling:

TouchableOpacity: Supports basic styling, with primary feedback being opacity change.
Pressable: Supports extensive and conditional styling based on interaction states.
Interaction States:

TouchableOpacity: Mainly focuses on the onPress event.
Pressable: Supports multiple interaction states (onPressIn, onPressOut, onLongPress, etc.).
Use Case:

TouchableOpacity: Good for simple touch feedback with opacity change.
Pressable: Better suited for custom and complex interactive components with detailed control over touch interactions.

useEffect vs useLayoutEffect:

Execution Timing: useEffect runs after the render and paint, while useLayoutEffect runs after render but before paint.

Performance Impact: useEffect is non-blocking and doesn't impact the visual rendering, while useLayoutEffect is blocking and can impact the initial rendering of the component.
Use Cases: useEffect is used for non-UI related effects, while useLayoutEffect is used for effects that need to manipulate or measure the DOM before the user sees it.

Error codes (HTTP Status codes):

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

How React works

Component-Based Architecture: React is built around the concept of reusable components. Components are self-contained units that encapsulate a part of the UI and its behavior. These components can be composed together to build complex UIs.

Virtual DOM (Document Object Model): React uses a virtual DOM to efficiently manage updates to the UI. The virtual DOM is a lightweight copy of the actual DOM. When changes are made to the UI, React first updates the virtual DOM rather than the real DOM, which is a costly operation. React then compares the virtual DOM with the real DOM to determine the most efficient way to update the actual DOM, minimizing the number of DOM manipulations needed.

Reconciliation: React's reconciliation process is responsible for determining the minimal set of changes needed to update the UI. It compares the previous virtual DOM with the new one and computes the differences (called "diffing"). By only updating the parts of the DOM that have changed, React ensures optimal performance.

Declarative Syntax: React encourages a declarative programming style, where you describe how your UI should look based on the application state, rather than imperatively manipulating the DOM to achieve a desired result. This makes the code easier to understand, debug, and maintain.

State Management: React allows you to manage the state of your application efficiently. Components can have their own state, which determines how they render and behave. When the state of a component changes, React automatically re-renders the component and its children, updating the UI accordingly.

JSX (JavaScript XML): JSX is a syntax extension for JavaScript that allows you to write HTML-like code within JavaScript. JSX makes it easier to write React components by providing a familiar syntax for describing the UI.

Lifecycle Methods: React components have lifecycle methods that allow you to hook into various stages of a component's life,

How many package managers to we have in NodeJS

1. NPM
2. YARN
3. PNPM

Promise is an object representing intermediate state of operation which is guaranteed to complete its execution at some point in future.	Async/Await is a syntactic sugar for promises, a wrapper making the code execute more synchronously.
Promise has 3 states – resolved, rejected and pending.	It does not have any states. It returns a promise either resolved or rejected.

Error handling is done using .then() and .catch() methods.	Error handling is done using .try() and .catch() methods.

Promise chains can become difficult to understand sometimes.	Using Async/Await makes it easier to read and understand the flow of the program as compared to promise chains.

1. When should you use each type of state management in React Native?

Answer:
The choice of state management in React Native depends on the complexity and requirements of the application:

- **React's built-in `useState` and `useReducer`:** Best for local or component-specific state. Suitable for lightweight applications or isolated UI logic.
- **Context API:** Useful for global states that don't require frequent updates, like themes, language settings, or user authentication.
- **Redux/MobX:** Ideal for large applications needing predictable state management, especially when there's a need to share state across deeply nested components or complex state transformations.
- **Recoil/Jotai/Zustand:** Modern libraries with a simpler setup than Redux, often chosen for projects where developers seek the flexibility of React's new paradigms without the boilerplate.

2. What performance optimization strategies do you use in React Native?

Answer:
Some common strategies to optimize performance in React Native include:

- **Memoization:** Using `React.memo`, `useMemo`, and `useCallback` to prevent unnecessary re-renders of components or functions.

- **Native components:** Leverage native modules/components for performance-heavy tasks, like video playback or advanced animations, as they perform faster than JavaScript-based components.
- **Optimizing images:** Compress images, use the correct image format, and consider using caching libraries to reduce image load time.
- **FlatLists:** Prefer `FlatList` or `SectionList` for rendering large lists instead of `ScrollView` to improve memory handling.
- **Memory management:** Release resources when no longer needed, such as removing event listeners in `componentWillUnmount` or using cleanup functions in `useEffect`.
- **Code splitting and lazy loading:** Split code and load components only when needed to reduce the initial load time.
- **Interaction Manager:** Use `InteractionManager` to delay heavy tasks until interactions are complete, improving the perceived smoothness of the app.
- **Use Hermes engine:** Hermes is an optimized JavaScript engine for React Native, reducing memory usage and improving startup time on Android.
- **Regular upgrades:** Keep React Native and its libraries updated to benefit from performance improvements in newer versions.

3. How would you synchronize data between multiple devices in real-time?

Answer:
To achieve real-time synchronization between devices in React Native, several options are available:

- **Firebase Firestore:** A managed, real-time NoSQL database by Google, allowing easy integration for syncing data across devices. Firestore offers built-in support for real-time listeners.
- **WebSockets (e.g., using `socket.io`):** Ideal for applications requiring low-latency data transfer and two-way communication, commonly used for real-time chat and collaborative tools.
- **PubNub:** A managed service that simplifies real-time data streaming and message broadcasting across multiple clients. It's well-suited for chat, live updates, and location tracking.

4. What is a race condition, and how does it affect React Native applications?

Answer:
A **race condition** occurs when multiple threads or processes access shared resources, such as variables or data structures, at the same time without proper synchronization. This leads to unexpected outcomes, especially when:

- Two or more threads read and write to the same variable without synchronization.
- Critical code blocks are not protected by mutexes, semaphores, or locks, causing unpredictable application behavior.

Solution:
To avoid race conditions, use **synchronization primitives** like locks, mutexes, or semaphores to ensure that only one thread can access the critical section at a time. In JavaScript, consider using techniques like promises or async/await to ensure sequential operations and avoid conflicting changes to shared resources.

1. When should you use Expo versus a bare React Native setup?

Answer:
Choosing between Expo and bare React Native depends on several factors:

- **Bundle size:** Expo apps may have larger bundle sizes because they include all Expo modules by default, which could be unnecessary for smaller apps. Bare React Native lets you add only what you need.
- **Native capabilities:** Expo provides limited access to native code, so if your app needs deep integration with native modules (e.g., complex animations or custom functionalities), bare React Native may be more suitable.
- **Deployment:** Expo simplifies the deployment process with `expo build` and `expo publish`, making it ideal for projects without complex native requirements. However, bare React Native provides more control over custom deployment configurations.
- **npm audit with Expo:** Expo abstracts certain libraries and dependencies, making it challenging to audit specific packages. With a bare setup, you have full control over package management and can easily run `npm audit` for any vulnerabilities.

2. How important is it to have zero vulnerabilities in your application?

Answer:
Maintaining zero vulnerabilities is ideal for security, but it's often a balance between risk and feasibility. Minor vulnerabilities that pose minimal risk may not always need immediate resolution, especially in non-critical areas of the app. However, for vulnerabilities that could expose user data or system integrity, it’s critical to prioritize fixes to ensure app security and user trust.

3. What are some basic principles of app security in React Native?

Answer:
Key principles include:

- **Secure Storage:** Store sensitive information like tokens securely, using secure storage libraries instead of async storage.
- **Network Security:** Use HTTPS for API requests and secure WebSocket connections. Implement certificate pinning to prevent man-in-the-middle attacks.
- **Code Obfuscation:** Use tools like ProGuard or R8 (for Android) and equivalent measures for iOS to obfuscate code and prevent reverse engineering.
- **Data Encryption:** Encrypt sensitive data, both in transit and at rest.
- **Authentication & Authorization:** Implement strong authentication methods and properly handle permissions for user roles and data access.

4. How do you monitor for new vulnerabilities in your project?

Answer:
There are several methods to stay updated on potential vulnerabilities:

- **npm audit:** Run `npm audit` regularly to identify security issues in dependencies.
- **GitHub Dependabot alerts:** GitHub provides automated security alerts for vulnerabilities in project dependencies.
- **Vulnerability databases:** Keep track of popular databases like the National Vulnerability Database (NVD) and OWASP for the latest security threats.

5. How do you decide when to move code to a native module, and how do you approach this?

Answer:
Code should move to a native module when performance-critical features or functionality are better handled natively, like complex animations, access to device sensors, or system-level APIs unavailable through JavaScript.
Approach:

- Create a bridge in React Native to allow communication between JavaScript and native code.
- Write the required logic in Java (for Android) or Objective-C/Swift (for iOS) and expose it to the JavaScript layer using React Native's native module setup.

6. Are you comfortable working with native code?

Answer:
Yes, I am comfortable working with native code, especially in situations where native performance or specific device features are required. Familiarity with Java/Kotlin for Android and Objective-C/Swift for iOS helps integrate custom functionality directly with the React Native codebase.

7. What challenges have you faced with CocoaPods and Gradle?

Answer:
Some common challenges include:

- **Dependency conflicts:** Conflicting versions of libraries can lead to build failures. Regularly updating dependencies and checking compatibility helps mitigate this.
- **Slow build times:** Both CocoaPods and Gradle can be slow, especially in CI/CD pipelines. Optimizations like using Gradle Daemon and caching dependencies can help.
- **Platform-specific issues:** Occasionally, a library works well on one platform but causes issues on another, requiring conditional configurations in `Podfile` or `build.gradle`.

8. How do you test your app on various devices?

Answer:
Testing across different devices involves:

- **Simulators/Emulators:** Useful for rapid testing across various screen sizes and OS versions.
- **Physical Devices:** Essential for testing actual performance, responsiveness, and compatibility.
- **Device Farms (e.g., Firebase Test Lab, BrowserStack):** Useful for automating tests on multiple real devices remotely, ensuring broad device coverage.

9. Are there tools to test how layouts look in an automated way?

Answer:
Yes, tools like **Detox** and **Appium** are often used for end-to-end UI testing, automating layout checks and interaction flows across different devices and orientations. Snapshot testing with **Jest** and tools like **Storybook** also help in visually verifying component appearance.

10. How do you detect crashes on users' devices?

Answer:
Crash monitoring tools like **Sentry**, **Firebase Crashlytics**, and **Bugsnag** are widely used to capture crash logs, stack traces, and insights into why an app crashed on a user’s device. These tools provide alerts and detailed reports to aid in debugging.

11. What other state management libraries have you used?

Answer:
Aside from Redux and Context API, I have experience with:

- **Recoil:** Offers a simple way to manage state with minimal boilerplate, with an atom-based approach for atomized state management.

- **Jotai/Zustand:** Lightweight, modern libraries that simplify state management with a smaller footprint, particularly useful for quick setup and specific state management scenarios.