**React Native / ReactJS**

1. **Có bao nhiêu công nghệ phát triển ứng dụng di động hiện tại?**
   * Các công nghệ phát triển ứng dụng di động hiện tại bao gồm:
     + Native Development: Swift (iOS), Kotlin (Android)
     + Cross-platform Development: React Native, Flutter, Xamarin
     + Hybrid Development: Ionic, Cordova
     + Progressive Web Apps (PWA)
2. **React Native là gì? Có bao nhiêu thread trong React Native? Giải thích cơ chế giao tiếp giữa các thread?**
   * React Native là một framework phát triển ứng dụng di động sử dụng JavaScript và React. Nó cho phép bạn xây dựng các ứng dụng mobile với trải nghiệm native trên cả iOS và Android.
   * Trong React Native có 3 thread chính:
     + **UI Thread (Main Thread):** Xử lý giao diện người dùng và tương tác với người dùng.
     + **JavaScript Thread:** Thực thi mã JavaScript và xử lý logic ứng dụng.
     + **Shadow Thread:** Tạo layout và tính toán kích thước các thành phần.
   * **Cơ chế giao tiếp giữa các thread:**
     + React Native sử dụng một cầu nối (bridge) để giao tiếp giữa JavaScript và các thành phần native. Các thông điệp được gửi qua cầu nối này bằng JSON, giúp JavaScript có thể gọi các API native và ngược lại.
3. **Nêu sự khác biệt giữa ReactJS và React Native? Ưu nhược điểm của React Native là gì?**
   * **Sự khác biệt:**
     + **ReactJS:** Được sử dụng để phát triển ứng dụng web.
     + **React Native:** Được sử dụng để phát triển ứng dụng di động native.
     + ReactJS sử dụng HTML và CSS, trong khi React Native sử dụng các thành phần native như View, Text và StyleSheet.
   * **Ưu điểm của React Native:**
     + Code reusability giữa iOS và Android.
     + Hiệu suất gần native.
     + Community và ecosystem rộng lớn.
   * **Nhược điểm của React Native:**
     + Hiệu suất không bằng native hoàn toàn.
     + Debugging có thể phức tạp.
     + Một số tính năng native có thể yêu cầu viết mã native.
4. **Virtual DOM là gì? Giải thích cơ chế hoạt động?**
   * **Virtual DOM** là một bản sao của DOM thật, được lưu trong bộ nhớ và được quản lý bởi React. Khi trạng thái của ứng dụng thay đổi, React sẽ cập nhật Virtual DOM thay vì cập nhật trực tiếp DOM thật.
   * **Cơ chế hoạt động:**
     + Khi trạng thái hoặc props thay đổi, React sẽ tạo một Virtual DOM mới.
     + So sánh Virtual DOM mới với Virtual DOM cũ (diffing).
     + React xác định các thay đổi và cập nhật DOM thật tương ứng (reconciliation).
5. **Khác biệt giữa state và props?**
   * **State:**
     + Quản lý dữ liệu nội bộ của component.
     + Có thể thay đổi trong suốt vòng đời của component.
     + Được khai báo bên trong component.
   * **Props:**
     + Được truyền từ component cha xuống component con.
     + Không thể thay đổi bởi component con (immutable).
     + Được sử dụng để giao tiếp giữa các component.
6. **Lifecycle của một Component?**
   * **Mounting:** Khi một component được tạo và chèn vào DOM.
     + constructor()
     + static getDerivedStateFromProps()
     + render()
     + componentDidMount()
   * **Updating:** Khi props hoặc state của component thay đổi.
     + static getDerivedStateFromProps()
     + shouldComponentUpdate()
     + render()
     + getSnapshotBeforeUpdate()
     + componentDidUpdate()
   * **Unmounting:** Khi component bị loại bỏ khỏi DOM.
     + componentWillUnmount()
7. **Liệt kê một số libraries dành cho state management.**
   * Redux
   * MobX
   * Recoil
   * Zustand
   * Context API (React's built-in)

**JavaScript**

1. **JavaScript là gì? Liệt kê các kiểu dữ liệu trong JS?**
   * JavaScript là một ngôn ngữ lập trình hướng đối tượng, thường được sử dụng để phát triển web.
   * Các kiểu dữ liệu trong JavaScript:
     + Primitive: String, Number, Boolean, Null, Undefined, Symbol, BigInt
     + Non-Primitive: Object (bao gồm Array, Function, Date, etc.)
2. **JavaScript ứng dụng ở những đâu?**
   * Phát triển web (client-side và server-side với Node.js).
   * Phát triển ứng dụng di động (React Native, Ionic).
   * Phát triển game.
   * Điều khiển phần cứng (IoT).
3. **Khai báo biến trong JS, nêu sự khác biệt giữa var, let, const?**
   * **var:**
     + Có phạm vi hàm hoặc global.
     + Có thể bị redeclared và reinitialized.
   * **let:**
     + Có phạm vi block (block scope).
     + Không thể bị redeclared nhưng có thể reinitialized.
   * **const:**
     + Có phạm vi block.
     + Không thể bị redeclared và reinitialized.
     + Phải được khởi tạo khi khai báo.
4. **Scope là gì?**
   * **Scope** (phạm vi) xác định vùng của mã nơi các biến có thể được truy cập. JavaScript có các loại scope:
     + **Global scope:** Biến được khai báo ngoài tất cả các hàm hoặc khối.
     + **Function scope:** Biến được khai báo bên trong hàm.
     + **Block scope:** Biến được khai báo trong khối (sử dụng let hoặc const).
5. **Hoisting là gì?**
   * **Hoisting** là cơ chế của JavaScript trong đó các khai báo biến và hàm được di chuyển lên đầu phạm vi trước khi mã được thực thi.
6. **Closure là gì?**
   * **Closure** là một hàm kết hợp với môi trường từ thời điểm nó được tạo ra. Closure cho phép một hàm truy cập và ghi nhớ phạm vi lexical tại thời điểm nó được định nghĩa.
7. **Event loop là gì? Giải thích cơ chế?**
   * **Event loop** là cơ chế giúp JavaScript thực thi non-blocking bằng cách quản lý các callback và các sự kiện.
   * **Cơ chế:**
     + Call stack: Chứa các hàm được gọi và thực thi.
     + Web APIs: Xử lý các tác vụ như setTimeout, DOM events.
     + Task queue: Chứa các callback đã sẵn sàng để thực thi.
     + Event loop kiểm tra call stack có trống không và task queue có callback không. Nếu có, nó sẽ di chuyển callback từ task queue vào call stack để thực thi.

**TypeScript**

1. **TypeScript là gì? Trình bày hiểu biết về OOP?**
   * TypeScript là một ngôn ngữ lập trình mạnh mẽ dựa trên JavaScript, bổ sung các tính năng như kiểu tĩnh và lập trình hướng đối tượng (OOP).
   * **OOP (Object-Oriented Programming)** là một mô hình lập trình dựa trên các đối tượng, bao gồm 4 nguyên lý chính:
     + **Encapsulation:** Đóng gói dữ liệu và các phương thức lại với nhau trong một đối tượng.
     + **Inheritance:** Tạo ra các lớp mới dựa trên các lớp hiện có.
     + **Polymorphism:** Đa hình, cho phép một hành động có thể được thực hiện theo nhiều cách.
     + **Abstraction:** Ẩn chi tiết triển khai và chỉ hiển thị các chức năng chính.
2. **Các kiểu dữ liệu trong TypeScript?**
   * String
   * Number
   * Boolean
   * Array
   * Tuple
   * Enum
   * Any
   * Void
   * Null
   * Undefined
   * Never
   * Object
3. **Khác biệt giữa interface và type?**
   * **Interface:**
     + Được sử dụng để định nghĩa cấu trúc của một đối tượng.
     + Có thể mở rộng bằng cách sử dụng từ khóa extends.
     + Có thể khai báo lại và mở rộng.
   * **Type:**
     + Có thể định nghĩa kiểu dữ liệu phức tạp bằng các kết hợp (union) và giao nhau (intersection).
     + Không thể khai báo lại.
     + Thường được sử dụng cho các kiểu dữ liệu cơ bản và phức tạp.
4. **Generics function là gì?**
   * **Generics** cho phép tạo các hàm, lớp, hoặc interface có thể hoạt động với nhiều kiểu dữ liệu khác nhau mà không mất đi tính an toàn kiểu (type safety).

**Git**

1. **Nghiên cứu về tạo git repo trên GitHub và các lệnh cơ bản pull/push.**
   * **Tạo Git repo trên GitHub:**
     + Đăng nhập vào GitHub.
     + Click vào nút "New repository".
     + Nhập tên repo và các thông tin khác, sau đó click "Create repository".
   * **Các lệnh cơ bản:**
     + git clone <repository-url>: Sao chép một repository từ GitHub về máy tính cục bộ.
     + git pull: Lấy và hợp nhất các thay đổi từ repository từ xa về branch hiện tại.
     + git add <file>: Thêm file vào staging area.
     + git commit -m "message": Commit các thay đổi trong staging area với một thông điệp.
     + git push: Đẩy các commit từ máy tính cục bộ lên repository từ xa.