RV College of Engineering®
Mysore Road, RV Vidyaniketan Post,
Bengaluru - 560059, Karnataka, India

Department of Master of Computer Applications (MCA)

Mobile Application Development (MCA221IA)

Hand Notes

Unit - 2	Topic: Introduction to PWAs, Tools to Measure Progressive WebApps
Editor: 1RV24MC071 - Nagaraja Mahabaleshwar Hegde	

List of Questions

02 Marks Questions:

Q1: What is a Progressive Web App (PWA)? Give some popular examples.
Answer:
A Progressive Web App (PWA) is a type of web application built using modern web technologies (like Service Workers, Web App Manifest, and HTTPS) and design patterns to deliver a user experience similar to native mobile or desktop applications. PWAs aim to be **reliable** (load instantly, even in uncertain network conditions), **fast** (respond quickly to user interactions), and **engaging** (feel like a natural app on the device, with features like push notifications). They can work offline, load quickly, and can be installed on a user's device directly from the browser.

Popular Examples:

- Twitter (formerly Twitter Lite)
- Starbucks
- Uber
- Pinterest
- Flipkart Lite
- Spotify

Q: What is the role of HTTPS in PWAs?
Answer:
HTTPS (Hypertext Transfer Protocol Secure) is crucial for PWAs because:

1. **Security:** It encrypts data between the user's browser and the server, preventing man-in-the-middle attacks and ensuring data integrity and privacy.
2. **Service Worker Requirement:** Modern browsers require web applications to be served over HTTPS to register and use Service Workers, which are fundamental for PWA features like offline support and push notifications.

Q3: List the advantages of PWAs.
Answer:
Advantages of PWAs include:

- **Offline Functionality:** Service workers enable access to content even without an internet connection.
- **Fast Loading:** Caching strategies make them load quickly after the first visit.
- **Push Notifications:** Ability to send re-engagement messages to users.
- **App-like Experience:** Smooth animations, navigation, and an immersive feel.
- **Installable:** Can be added to the home screen or app drawer without an app store.
- **Discoverability:** Searchable by search engines like any website.
- **Platform-Agnostic:** Works across different platforms and devices from a single codebase.
- **Automatic Updates:** Always serve the latest version without manual user updates.

Q4: What is an "App Shell" in PWA architecture?

Answer:

An App Shell is the minimal HTML, CSS, and JavaScript required to power the user interface of a Progressive Web App. It's the static framework or skeleton of the app that loads quickly and is cached by a Service Worker on the first visit, providing an instant, reliably performing UI while dynamic content loads.

Q5: How do PWAs achieve "installability"?

Answer:

PWAs achieve "installability" primarily through the **Web App Manifest** file and by meeting certain browser criteria (like being served over HTTPS and having a registered Service Worker). The manifest provides metadata (name, icons, start URL, display mode) that allows browsers to prompt users to "Add to Home Screen" or install the app, making it launchable like a native app.

Q6: What does "Progressive" in Progressive Web App mean?
Answer:
"Progressive" means that PWAs are built with progressive enhancement as a core tenet. They work for every user, regardless of browser choice, because they're built with web standards. They then "progressively" enhance with more advanced features (like offline support, push notifications, installability) for users with modern browsers that support those capabilities.

04/06 Marks Questions:

Q7: Explain key features of PWAs.
Answer:
Key features of Progressive Web Apps include:

- **Progressive Enhancement:** They work for any user on any browser, and progressively enhance with features for users on modern browsers.
- **Responsive:** They adapt to any form factor: desktop, mobile, tablet, or whatever comes next.
- **Connectivity Independent:** Service workers enable them to work offline or on low-quality networks.
- **App-like Interface:** Utilize an app shell model to provide app-style navigations and interactions, feeling more like a native app than a website.
- **Fresh:** Always up-to-date thanks to the service worker update process, ensuring users have the latest version.
- **Safe and Secure (HTTPS):** Served via HTTPS to prevent snooping and ensure content hasn't been tampered with. This is a prerequisite for service workers.
- **Discoverable:** Identifiable as "applications" thanks to W3C manifests and service worker registration scope, allowing search engines to find them.
- **Re-engageable:** Make re-engagement easy through features like push notifications, even when the browser is closed.
- **Installable:** Allow users to "keep" apps they find most useful on their home screen without the hassle of an app store, using the Web App Manifest.
- **Linkable:** Easily shareable via a URL and do not require complex installation.

Q8: Explain Lighthouse and how it is used to measure PWAs.
Answer:
Lighthouse is an open-source, automated tool developed by Google for improving the quality of web pages. It can be run against any web page, public or requiring authentication. It has audits for performance, accessibility, progressive web app features, SEO, and more.

How it is used to measure PWAs:

1. **Running Audits:** Lighthouse can be run from Chrome DevTools (Audits panel), from the command line (Node.js), or as a Node module.
2. **PWA Checklist:** It specifically checks for PWA criteria based on a baseline PWA checklist. This includes:
    - **Fast and Reliable:** Verifies if the page loads quickly, even on slow networks, and if it provides a custom offline page.
    - **Installable:** Checks for a Web App Manifest with required properties (like name, short_name, start_url, icons) and if the site is served over HTTPS.
    - **PWA Optimized:** Ensures the PWA is configured for a custom splash screen, sets a theme color, and has a viewport meta tag.

3. **Scoring and Reports:** After running the audits, Lighthouse generates a report with scores (0-100) for each category. For PWAs, it lists which criteria are met and which are not.

4. **Actionable Advice:** The report provides specific suggestions and links to documentation on how to fix the identified issues and improve the PWA's compliance, performance, and overall quality. This helps developers iteratively enhance their PWAs.

08 Marks Questions

Q9: Describe tools used to measure and test Progressive Web Apps.
Answer:
Several tools are available to help developers measure, test, and debug Progressive Web Apps to ensure they meet performance, reliability, and installability criteria:

1. **Google Lighthouse:**
    - **Description:** An open-source, automated tool integrated into Chrome DevTools (Audits/Lighthouse panel), available as a CLI tool, and a Node module.
    - **Usage:** Audits PWAs against a checklist covering performance, accessibility, best practices, SEO, and PWA-specific features.
    - **PWA Testing:** Checks for Service Worker registration, HTTPS usage, a valid Web App Manifest (with properties like name, short_name, start_url, icons, display), offline usability, and fast load times.
    - **Output:** Provides a score (0-100) for each category and a detailed report with actionable suggestions for improvements.
2. **Chrome DevTools:**
    - **Description:** A suite of web developer tools built directly into the Google Chrome browser.
    - **Usage for PWAs:**
        - **Application Panel:**
            - *Manifest:* Inspect the parsed Web App Manifest, trigger "Add to Home Screen" events.
            - *Service Workers:* View registered service workers, inspect their lifecycle (install, activate, running, redundant), test push notifications, background sync, unregister/update service workers, and simulate offline mode.
            - *Cache Storage:* Inspect and clear cached resources managed by Service Workers.
            - *Storage:* View and manage other storage like Local Storage, IndexedDB.
        -
        - **Network Panel:** Simulate various network conditions (slow 3G, offline), inspect requests and responses.
        - **Performance Panel:** Profile runtime performance to identify bottlenecks.
        - **Audits/Lighthouse Panel:** Direct access to Lighthouse audits.

3. **WebPageTest:**
    - **Description:** A powerful online tool for measuring web page performance from various locations around the world, using real browsers and different connection speeds.
    - **Usage:** Provides detailed performance metrics (load time, Speed Index, Time to First Byte), waterfall charts, and video recordings of page load.

    - **PWA Testing:** While not PWA-specific like Lighthouse, it's excellent for deep performance analysis, which is a core PWA tenet. It can help identify slow-loading resources or rendering issues that affect the user experience.
4. **Workbox:**
    - **Description:** A set of JavaScript libraries from Google that make it easier to write and manage Service Workers, particularly for caching strategies and offline support.
    - **Usage:** Not a measurement tool itself, but a development tool that simplifies PWA development, reducing boilerplate code for common Service Worker patterns like precaching, runtime caching, and routing.
    - **Testing Aid:** Using Workbox helps ensure robust and well-tested Service Worker logic, which indirectly contributes to a better PWA. It has its own logging and debugging features.
5. **PWABuilder:**
    - **Description:** An open-source tool from Microsoft that helps developers build quality PWAs and package them for app stores.
    - **Usage:** Users can enter their website URL, and PWABuilder analyzes it for PWA compliance, suggests improvements for the manifest, service worker, and security.
    - **PWA Testing:** It provides a "PWA Report Card" highlighting what's done well and what needs improvement to be a fully functional PWA. It can also help generate manifest files and basic service workers.

These tools provide a comprehensive suite for developing, debugging, testing, and optimizing PWAs to ensure they deliver a high-quality user experience

10 Marks Questions

Q10: Compare PWAs with native apps.
Answer:
Progressive Web Apps (PWAs) and Native Apps both aim to provide rich user experiences on mobile and desktop devices, but they differ significantly in their technology, distribution, and capabilities.

Feature	Progressive Web Apps (PWAs)	Native Apps
Technology Stack	HTML, CSS, JavaScript (Web technologies)	Platform-specific languages (e.g., Swift/Objective-C for iOS, Java/Kotlin for Android)
Installation	Directly from the browser ("Add to Home Screen"); no app store required initially.	Via app stores (e.g., Apple App Store, Google Play Store).

Discovery	Through search engines (SEO), social media, direct links.	Primarily through app store search, featured lists, ads.
Platform Reach	Cross-platform; works on any device with a modern browser.	OS-specific; separate versions needed for iOS, Android, etc.
Development Cost & Effort	Generally lower; single codebase can target multiple platforms.	Generally higher; often requires separate development teams and codebases for different OS.
Updates	Automatic and seamless; new content/features loaded when the PWA is opened (via Service Worker update process).	Manual or automatic updates via the app store; can be delayed by user or store review.
Offline Access	Yes, through Service Workers and caching strategies. Content can be pre-cached or cached on demand.	Yes, typically designed for offline use by storing data locally.
Performance	Can be very fast, especially with App Shell and caching. Performance relies heavily on web optimization techniques.	Generally very high performance due to direct access to device hardware and optimized code.
Hardware Access	Limited but growing through Web APIs (e.g., Geolocation, Camera (getUserMedia), Bluetooth, NFC - support varies by browser/OS).	Extensive access to device hardware and features (e.g., advanced camera controls, contacts, calendar, background tasks, device sensors).
Push Notifications	Yes, via Service Workers and Push API.	Yes, a standard feature.
App Store Approval	Not required for basic PWA functionality and direct installation. May be needed if distributing via stores like Google Play.	Mandatory; apps must pass review processes which can be time-consuming and restrictive.

Data Storage	Browser storage (Cache API, IndexedDB, LocalStorage) - typically smaller limits than native apps.	More generous local storage options on the device.
Security	Served over HTTPS; sandboxed by the browser. Permissions model for sensitive features.	OS-level security features; permissions requested at install-time or runtime. App code has more system access.
Monetization	Web-based payment APIs, ads. In-app purchases if packaged and distributed via stores.	Robust in-app purchase systems, ads, subscriptions.
Deep Linking	Standard web URLs, inherently linkable.	Requires specific setup (e.g., URI schemes, Universal Links, App Links).
Size	Generally very lightweight as only essential assets are initially loaded/cached.	Can be significantly larger due to bundled resources and binaries.