Litera.ai

PWA & Mobile Patient Experience

Steps to make the patient portal installable, mobile-optimized, and ready for pilot patients

1. Why a PWA Instead of a Native App

A Progressive Web App is the right choice for the pilot for the following reasons:

- Patients receive a link via email on their phone. They are already in a mobile browser.
- No App Store approval process (Apple takes 1-4 weeks for health apps, requires a $99/year developer account).
- No APK sideloading or TestFlight for Android/iOS testing.
- Patients with limited English proficiency are unlikely to search an App Store.
- A PWA can be installed to the home screen and looks/feels like a native app.
- The entire care team can test immediately by opening a URL.

Post-pilot, if app store distribution becomes a requirement, you can wrap the PWA in a WebView using Capacitor (1-2 days of work) or build a React Native app that calls the same backend API.

2. Create the Web App Manifest

The manifest.json file tells the browser that your web app can be installed. It defines the app name, icons, colors, and display behavior.

Replit AI Prompt — Create Manifest

Create a file at client/public/manifest.json with the following configuration: name: 'Litera.ai — My Care Plan', short_name: 'Litera', description: 'View your care plan and check in with your care team', start_url: '/', display: 'standalone', background_color: '#ffffff', theme_color: '#1a3c6e', orientation: 'portrait'. Add an icons array with at least two entries: a 192x192 and a 512x512 PNG icon. For now, create simple placeholder icons using a canvas-based script or use the existing Litera.ai heart logo. Place the icon files in client/public/icons/. Then in client/index.html, add: <link rel='manifest' href='/manifest.json'> and <meta name='theme-color' content='#1a3c6e'> in the <head> section.

3. Service Worker for Offline Support

A service worker caches the app shell so the patient portal loads even on a slow or intermittent connection. For a pilot, a basic cache-first strategy is sufficient.

Replit AI Prompt — Service Worker

Add a service worker to the Vite build. Install vite-plugin-pwa: npm install vite-plugin-pwa --save-dev. In vite.config.ts, import VitePWA from 'vite-plugin-pwa' and add it to the plugins array with this configuration: VitePWA({ registerType: 'autoUpdate', manifest: false (we already have manifest.json), workbox: { globPatterns: ['**/*.{js,css,html,ico,png,svg,woff2}'], runtimeCaching: [{ urlPattern: /^\/api\/.*/i, handler: 'NetworkFirst', options: { cacheName: 'api-cache', expiration: { maxEntries: 50, maxAgeSeconds: 300 } } }] } }). This caches the app shell and applies network-first caching for API calls (so patients see the latest data when online, but can still see cached content when offline).

4. Mobile Viewport Optimization

The patient portal must be usable on a 375px-wide screen (iPhone SE, the smallest common phone). The following areas need attention:

4.1 Viewport Meta Tag

Verify that client/index.html includes this meta tag in the <head>:

```
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0, user-scalable=no">
```

The user-scalable=no prevents accidental zooming on form inputs, which is disorienting for elderly patients.

4.2 Patient Portal Mobile Audit

Replit AI Prompt — Mobile Optimization

Audit and fix client/src/pages/patient-portal.tsx for mobile responsiveness. Test at 375px width. Specifically: 1) The year-of-birth verification form should be centered, with the input and button full-width on mobile. The input should use inputMode='numeric' and pattern='[0-9]*' so mobile phones show the numeric keyboard. 2) The care plan content should use single-column layout on mobile with adequate padding (px-4). 3) Medication cards should stack vertically, not side-by-side. 4) The check-in traffic light buttons (green/yellow/red) should be large touch targets — at minimum 48x48px, ideally full-width stacked buttons on mobile. 5) All text should be at least 16px to prevent auto-zoom on iOS Safari. 6) Add a sticky header with the patient name and a language indicator. 7) Add touch-friendly spacing between all interactive elements (minimum 8px gap).

5. Add-to-Home-Screen Prompt

After a patient successfully verifies their year of birth, show a prompt suggesting they install the app to their home screen. This makes future access easier (they tap an icon instead of finding an email link).

Replit AI Prompt — Install Prompt

In client/src/pages/patient-portal.tsx, after successful YOB verification, add an install prompt component. Use the beforeinstallprompt browser event: 1) Create a state variable to capture the deferred prompt event. 2) Listen for the 'beforeinstallprompt' event in a useEffect, call e.preventDefault() and save the event. 3) After verification succeeds, show a card/banner that says (in the patient's translated language if possible, otherwise in English): 'Add to Home Screen for easy access to your care plan' with an 'Install' button and a 'Not now' dismiss option. 4) When the user taps 'Install', call deferredPrompt.prompt() and await the choice. 5) On iOS Safari, the beforeinstallprompt event does not fire. Detect iOS and instead show instructions: 'Tap the share button, then tap Add to Home Screen'. Include a small screenshot or icon showing the share button.

6. SMS Check-In Reminders (Optional)

Email may not reach all patients reliably (spam filters, infrequent email checking). For the pilot, adding SMS via Twilio provides a second channel.

6.1 Twilio Setup

1. Create a Twilio account at twilio.com. The free trial includes $15 in credit.
2. Get a phone number from the Twilio console (Phone Numbers > Buy a Number). Cost: $1/month.
3. Note your Account SID and Auth Token from the Twilio dashboard.
4. Add three new Replit secrets: TWILIO_ACCOUNT_SID, TWILIO_AUTH_TOKEN, TWILIO_PHONE_NUMBER.

Replit AI Prompt — SMS Check-In

Install the Twilio SDK: npm install twilio. Create a new file server/services/twilio.ts that exports a sendCheckInSms(phoneNumber, patientName, portalLink) function. It should use the Twilio REST client to send an SMS with a message like: 'Hi {patientName}, your care team at Litera.ai would like to check in. How are you feeling? Tap here to respond: {portalLink}'. Initialize the Twilio client with process.env.TWILIO_ACCOUNT_SID and process.env.TWILIO_AUTH_TOKEN. Use process.env.TWILIO_PHONE_NUMBER as the 'from' number. In server/services/scheduler.ts (or wherever check-in sending logic lives), after sending the email, also call sendCheckInSms if the patient has a phone number on file. Wrap the SMS call in a try-catch so email delivery is not affected if SMS fails.

7. PWA Testing Checklist

After completing the steps above, verify each item:

- **Manifest loads:** Open Chrome DevTools > Application > Manifest. All fields should appear with no errors.
- **Service worker registers:** DevTools > Application > Service Workers. Status should be 'activated and is running'.
- **Install prompt works:** On Android Chrome, the install banner or custom prompt appears. On iOS Safari, manual install instructions appear.
- **Offline shell loads:** Enable airplane mode. Navigate to the app URL. The app shell should load (API calls will fail gracefully).
- **Mobile layout:** Use DevTools device toolbar to test at 375px (iPhone SE), 390px (iPhone 14), and 414px (iPhone 14 Plus).
- **Touch targets:** All buttons are at least 48x48px. No elements overlap. Adequate spacing between interactive elements.
- **Numeric keyboard:** On a real phone, tapping the year-of-birth input shows the numeric keyboard, not the full keyboard.
- **Check-in buttons:** Green/yellow/red buttons are large, clearly labeled, and show a confirmation after tapping.
- **Email link works:** Send a real care plan email. Open the link on your phone. Verify the full flow works.
- **Lighthouse audit:** Run Lighthouse (DevTools > Lighthouse > check Progressive Web App). Target a PWA score of 90+.