1. Building a Progressive Web App (PWA)

A **Progressive Web App** is a web application enhanced with **modern browser capabilities** to behave like a native application(mobile app) while remaining deployable as a website.

Core idea

Use web standards (HTML, CSS, JavaScript) + browser APIs to deliver:

- Offline capability
- Installabilit
- Background processing
- Native-like UX

A Blazor WebAssembly PWA:

- Runs **entirely in the browser**
- Executes **.NET code on WebAssembly**
- Uses **Service Workers** for offline support
- Uses **Web App Manifest** for installation metadata

2. Key Features of a PWA

1. Offline Support

- Cached assets and API responses
- Works with flaky or no network

2. Installable

- Appears as an app icon
- Launches in standalone window (no browser UI)

3. Responsive

- Works on mobile, tablet, desktop

4. Secure

- HTTPS enforced

5. Automatic Updates

- New version activates silently via service worker lifecycle

3. Who Uses PWAs in 2026?

Industry	**Use Case**
E-commerce	Offline browsing, fast reloads
SaaS(Software as a Service)	Installable dashboards
FinTech(Financial Technology)	Secure lightweight apps
Education	Cross-platform learning apps
Government	Low-bandwidth support

4. Why PWAs Are Preferred

- No app-store dependency
- Lower maintenance cost
- Instant updates
- Single codebase

5. Creating a Blazor WebAssembly Application as a PWA

Step 1: Create Project (PWA Enabled)

```
Bash

dotnet new blazorwasm -n BlazorPwaDemo --pwa
```

This **automatically generates**:

- wwwroot/manifest.json
- wwwroot/service-worker.js

Step 2: Understand the Project Structure

```
wwwroot/
  ├── manifest.json
  ├── service-worker.js
  ├── service-worker.published.js
  ├── index.html
```

Step 3: Web App Manifest (manifest.json)

```json
{
  "name": "Blazor PWA Demo",
  "short_name": "BlazorPWA",
  "start_url": "/",
  "display": "standalone",
  "background_color": "#ffffff",
  "theme_color": "#512BD4",
  "icons": [
    {
      "src": "icons/icon-192.png",
      "sizes": "192x192",
      "type": "image/png"
    }
  ]
}
```

Important properties

- display: standalone → removes browser chrome
- start_url → app entry point
- icons → required for install prompt

Step 4: Service Worker (service-worker.published.js)

```javascript
self.addEventListener('fetch', event => {
    event.respondWith(
        caches.match(event.request)
              .then(response => response || fetch(event.request))
    );
});
```

This enables:

- Cache-first strategy
- Offline fallback
- A **Service Worker** is a special JavaScript file that runs in the background of the browser, separate from the main web page.