Angular 16 Zero-Dependency Microfrontend Guide

This guide creates a pure Angular 16 microfrontend with **zero additional dependencies** - just vanilla Angular that can be loaded by any microfrontend orchestrator (Single SPA, Module Federation, or custom).

Prerequisites

- Angular CLI 16+
- Node.js 16+
- No additional dependencies needed!

Initial Setup

1. Create New Angular Project

```bash
ng new my-zero-dep-microfrontend --routing=true --style=scss
cd my-zero-dep-microfrontend
```

2. Install Only Style Isolation Dependencies (Optional)

```bash
# Only for style isolation (completely optional)
npm install -D postcss postcss-prefixwrap autoprefixer
```

Note: Even PostCSS is optional - you can do manual style prefixing instead.

Project Configuration

3. Configure angular.json

Replace the build configuration in `angular.json`:

```json
```

```
{
  "projects": {
    "my-zero-dep-microfrontend": {
      "architect": {
        "build": {
          "builder": "@angular-devkit/build-angular:browser",
          "options": {
            "outputPath": "dist",
            "index": "src/index.html",
            "main": "src/main.microfrontend.ts",
            "polyfills": [
              "zone.js"
            ],
            "tsConfig": "tsconfig.app.json",
            "assets": [],
            "styles": ["src/styles.scss"],
            "scripts": [],
            "optimization": true,
            "outputHashing": "none",
            "extractLicenses": false,
            "namedChunks": false,
            "vendorChunk": false,
            "commonChunk": false,
            "buildOptimizer": true,
            "aot": true
          }
        }
      }
    }
  }
}
```

4. Create Zero-Dependency Microfrontend Entry

Create `src/main.microfrontend.ts`:

typescript

```
import { enableProdMode, NgZone, ApplicationRef } from '@angular/core';
import { platformBrowserDynamic } from '@angular/platform-browser-dynamic';
import { Router } from '@angular/router';

import { AppModule } from './app/app.module';
import { environment } from './environments/environment';

if (environment.production) {
  enableProdMode();
}

// Zero-dependency microfrontend implementation
let ngZone: NgZone;
let appRef: ApplicationRef;
let router: Router;
let containerElement: HTMLElement;

// Standard microfrontend lifecycle - no external dependencies
export async function bootstrap(props?: any): Promise<void> {
  console.log('Zero-dependency microfrontend bootstrap', props);
  return Promise.resolve();
}

export async function mount(props?: any): Promise<any> {
  console.log('Zero-dependency microfrontend mount', props);

  // Create or find container
  containerElement = document.getElementById('zero-dep-microfrontend');
  if (!containerElement) {
    containerElement = document.createElement('div');
    containerElement.id = 'zero-dep-microfrontend';

    // If props specify where to mount, use that. Otherwise append to body
    const mountPoint = props?.domElement || document.body;
    mountPoint.appendChild(containerElement);
  }

  // Create app-root element
  const appRoot = document.createElement('app-root');
  containerElement.appendChild(appRoot);

  try {
    // Bootstrap Angular
    const moduleRef = await platformBrowserDynamic().bootstrapModule(AppModule);

    ngZone = moduleRef.injector.get(NgZone);
    appRef = moduleRef.injector.get(ApplicationRef);
    router = moduleRef.injector.get(Router);

    // Handle initial navigation
    if (props?.basePath && router) {
```

```
      ngZone.run(() => {
        router.navigateByUrl(props.basePath);
      });
    }

    return moduleRef;
  } catch (error) {
    console.error('Mount error:', error);
    throw error;
  }
}

export async function unmount(props?: any): Promise<void> {
  console.log('Zero-dependency microfrontend unmount', props);

  if (appRef) {
    appRef.destroy();
  }

  if (containerElement?.parentNode) {
    containerElement.parentNode.removeChild(containerElement);
  }

  // Cleanup
  ngZone = null!;
  appRef = null!;
  router = null!;
  containerElement = null!;

  return Promise.resolve();
}

export async function update(props?: any): Promise<void> {
  if (router && props?.basePath) {
    ngZone.run(() => {
      router.navigateByUrl(props.basePath);
    });
  }
  return Promise.resolve();
}

// For standalone usage (not as microfrontend)
if (!(window as any).__MICROFRONTEND_MODE__) {
  mount().catch(err => console.error('Standalone bootstrap failed:', err));
}
```

Style Isolation (Zero Dependencies)

5. Manual Style Scoping (No PostCSS needed)

Update `src/styles.scss` with manual scoping:

scss

```scss
// Manual style scoping - no dependencies required
.zero-dep-microfrontend {

  // Import 3rd party libraries inside the scoped container
  @import '~bootstrap/dist/css/bootstrap.min.css';
  @import '~@angular/material/prebuilt-themes/indigo-pink.css';
  @import '~primeng/resources/themes/lara-light-blue/theme.css';
  @import '~primeng/resources/primeng.min.css';

  // Reset and base styles
  * {
    box-sizing: border-box;
  }

  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;
  line-height: 1.5;

  // Scope 3rd party component styles
  .mat-button, .mat-raised-button, .mat-fab {
    font-family: inherit;
  }

  .btn {
    font-family: inherit;
  }

  .p-button, .p-inputtext {
    font-family: inherit;
  }
}
```

6. Alternative: PostCSS Auto-Scoping (Optional)

If you installed PostCSS, create `postcss.config.js`:

javascript

```javascript
module.exports = {
  plugins: [
    require('autoprefixer'),
    require('postcss-prefixwrap')('.zero-dep-microfrontend')
  ]
};
```

7. Update App Component

Modify `src/app/app.component.ts`:

typescript

```
import { Component, ViewEncapsulation } from '@angular/core';

@Component({
  selector: 'app-root',
  template: `
    <div class="zero-dep-microfrontend">
      <header>
        <h2>Zero Dependency Microfrontend</h2>
        <nav>
          <a routerLink="/">Home</a> |
          <a routerLink="/about">About</a> |
          <a routerLink="/products">Products</a>
        </nav>
      </header>
      <main>
        <router-outlet></router-outlet>
      </main>
    </div>
  `,
  styleUrls: ['./app.component.scss'],
  encapsulation: ViewEncapsulation.Emulated
})
export class AppComponent {
  title = 'zero-dep-microfrontend';
}
```

Build Configuration

8. Build Scripts

Update `package.json`:

```
json

{
  "scripts": {
    "build:microfrontend": "ng build --configuration=production",
    "serve:dist": "npx http-server dist -p 4200 --cors",
    "dev": "ng serve",
    "dev:microfrontend": "ng serve --port 4201"
  }
}
```

9. Build Command

```
bash

npm run build:microfrontend
```

10. Final Output

Generates exactly 2 files:

- `dist/main.js` (pure Angular with lifecycle methods)
- `dist/styles.css` (scoped styles)

Usage with Any Orchestrator

11. Single SPA Integration

```javascript
// In Single SPA shell
registerApplication({
  name: 'zero-dep-mf',
  app: () => System.import('https://cdn.com/zero-dep-mf/main.js'),
  activeWhen: '/zero-dep'
});
```

12. Module Federation Integration

```javascript
// In Module Federation shell
const ZeroDepMF = React.lazy(() => import('zeroDepMF/App'));

// Wrap in React component
function ZeroDepWrapper() {
  const ref = useRef();

  useEffect(() => {
    import('zeroDepMF/App').then(app => {
      app.mount({ domElement: ref.current });
    });

    return () => import('zeroDepMF/App').then(app => app.unmount());
  }, []);

  return <div ref={ref}></div>;
}
```

13. Custom Orchestrator Integration

```javascript
```

```
// Any custom framework
class MicrofrontendLoader {
  async loadApp(url, mountPoint) {
    const app = await System.import(url);
    await app.bootstrap();
    await app.mount({ domElement: mountPoint });
    return app;
  }

  async unloadApp(app) {
    await app.unmount();
  }
}

// Usage
const loader = new MicrofrontendLoader();
const app = await loader.loadApp('https://cdn.com/app/main.js', document.getElementById('app'));
```

14. Vanilla JavaScript Integration

html

```
<!DOCTYPE html>
<html>
<head>
    <link rel="stylesheet" href="https://cdn.com/zero-dep-mf/styles.css">
</head>
<body>
    <div id="my-app-container"></div>

    <script type="module">
        import { bootstrap, mount, unmount } from 'https://cdn.com/zero-dep-mf/main.js';

        await bootstrap();
        await mount({
          domElement: document.getElementById('my-app-container'),
          basePath: '/'
        });

        // Later...
        // await unmount();
    </script>
</body>
</html>
```

Advanced Features (Still Zero Dependencies)

15. Communication System

typescript

```typescript
// services/communication.service.ts
import { Injectable } from '@angular/core';
import { Subject, fromEvent, Observable } from 'rxjs';
import { map } from 'rxjs/operators';

@Injectable({ providedIn: 'root' })
export class CommunicationService {
  private eventBus = new Subject<any>();

  // Internal events (within this microfrontend)
  emit(event: string, data: any) {
    this.eventBus.next({ event, data });
  }

  listen(event: string): Observable<any> {
    return this.eventBus.pipe(
      filter((item: any) => item.event === event),
      map((item: any) => item.data)
    );
  }

  // Global events (between microfrontends)
  emitGlobal(event: string, data: any) {
    window.dispatchEvent(new CustomEvent(`mf:${event}`, { detail: data }));
  }

  listenGlobal(event: string): Observable<any> {
    return fromEvent(window, `mf:${event}`).pipe(
      map((e: any) => e.detail)
    );
  }
}
```

16. State Management (Without External Libraries)

```typescript
```

```
// services/state.service.ts
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';

interface AppState {
  user: any;
  theme: string;
  data: any[];
}

@Injectable({ providedIn: 'root' })
export class StateService {
  private state = new BehaviorSubject<AppState>({
    user: null,
    theme: 'light',
    data: []
  });

  state$ = this.state.asObservable();

  updateState(partial: Partial<AppState>) {
    this.state.next({ ...this.state.value, ...partial });
  }

  getState(): AppState {
    return this.state.value;
  }
}
```

Testing

17. Test as Standalone App

bash

```
# Development server
npm run dev

# Build and serve
npm run build:microfrontend
npm run serve:dist
```

18. Test as Microfrontend

Create `test-shell.html`:

html

```html
<!DOCTYPE html>
<html>
<head>
  <title>Microfrontend Test Shell</title>
  <link rel="stylesheet" href="./styles.css">
</head>
<body>
  <h1>Test Shell</h1>
  <button onclick="loadMF()">Load Microfrontend</button>
  <button onclick="unloadMF()">Unload Microfrontend</button>
  <div id="mf-container"></div>

  <script type="module">
    let loadedApp = null;

    window.loadMF = async () => {
      const app = await import('./main.js');
      await app.bootstrap();
      loadedApp = await app.mount({
        domElement: document.getElementById('mf-container')
      });
    };

    window.unloadMF = async () => {
      if (loadedApp) {
        const app = await import('./main.js');
        await app.unmount();
        loadedApp = null;
      }
    };
  </script>
</body>
</html>
```

Benefits of Zero Dependencies

19. Advantages

1. **Minimal Bundle Size**: Only Angular core, no additional libraries
2. **Maximum Compatibility**: Works with any orchestrator
3. **No Lock-in**: Easy to migrate between frameworks
4. **Faster Loading**: Fewer dependencies to download
5. **Simpler Debugging**: No wrapper libraries to debug through
6. **Better Caching**: Fewer files to cache and manage
7. **Framework Agnostic**: Can be loaded by React, Vue, vanilla JS, etc.

20. Use Cases

Perfect for:

- **Multi-framework environments** where shell might be React/Vue
- **Legacy system integration** where you can't use modern bundlers
- **Maximum performance** requirements
- **Simple deployment** scenarios
- **CDN optimization** where every KB matters

Troubleshooting

Module Import Errors: Ensure your build outputs ES modules format compatible with SystemJS or native modules.

Style Conflicts: Always scope styles manually or with PostCSS, never rely on build-time isolation alone.

Memory Leaks: Always clean up in unmount - check for subscriptions, intervals, and event listeners.

Browser Compatibility: Test in target browsers since you're not using polyfills from additional libraries.

This zero-dependency approach gives you the smallest possible bundle while maintaining full microfrontend capabilities and maximum flexibility.