MICRO-FRONTENDS AND MODULE FEDERATION IN ANGULAR

by Hürkan UĞUR

What is Module Federation?

```
Module federation, that is introduced in Angular 12, is a feature of Webpack 5 that allows developers to
share code across multiple applications. With this feature, you can create microfrontends that are
independently developed and deployed, yet still, work together to form a single application.
```

What is a Standalone Component?

```
Standalone components are Angular components that are not bound to any particular NgModule and can be
used across multiple parts of an application without the need for explicit imports and exports within
NgModule declarations.
```

```
• Standalone Components are highly reusable and can be used across modules and projects.

• Standalone Components eliminate complex import/export relationships between modules, simplifying the
component hierarchy.

• Standalone Components work perfectly with lazy-loaded modules.

• Standalone Components can integrate with third-party libraries without modifying NgModule
declarations.

• Standalone Components reduce the complexity of managing module dependencies.

• Standalone Components encourage cleaner code separation.

• Standalone Components are more portable for future refactoring or migration.
```

Create New Apps

```
ng new host
```

```
ng new teams
```

```
ng new calendar
```

Upgrade Angular Version

```
ng update @angular/cli @angular/core --allow-dirty
```

Install PrimeNG

```
npm i primeng@latest
```

```
npm i primeicons
```

```
src/styles.css:
```

```css
@import "primeng/resources/themes/bootstrap4-light-blue/theme.css";
@import "primeng/resources/primeng.css";
@import "primeicons/primeicons.css";

* {
    padding: 0;
    margin: 0;
    box-sizing: border-box;
}

body {
    font-family: var(--font-family);
}
```

Migrate the Existing Angular Projects to Standalone

```
Run the command below and select "Convert all components, directives and pipes to standalone":
```

```
ng g @angular/core:standalone
```

```
Run the command below and select "Remove unnecessary NgModule classes":
```

```
ng g @angular/core:standalone
```

```
Run the command below and select "Bootstrap the project using standalone APIs":
```

```
ng g @angular/core:standalone
```

Bootstrapping the App Using a Standalone Component

1) Delete the AppComponent and create your own Standalone Components for all projects:

```
calendar:
```

```
ng g c calendar --standalone
```

```
teams:
```

```
ng g c teams --standalone
```

```
host:
```

```
ng g c host --standalone
```

2) Create route files for all projects (The route list will be provided during the Module Federation setup):

```
host/src/app/routes/host.routes.ts:
```

```
export const HOST_ROUTES: Routes = [];
```

```
calendar/src/app/routes/calendar.routes.ts:
```

```
export const CALENDAR_ROUTES: Routes = [];
```

```
teams/src/app/routes/teams.routes.ts:
```

```
export const TEAMS_ROUTES: Routes = [];
```

3) Update the index.html to specify the application's entry point for all projects:

```
host/src/index.html:
```

```
<body>
  <app-host></app-host>
</body>
```

```
calendar/src/index.html:
```

```html
<body>
  <app-calendar></app-calendar>
</body>
```

```
teams/src/index.html:
```

```html
<body>
  <app-teams></app-teams>
</body>
```

4) Update the main.ts to migrate to BootstrapApplication:

```
host/src/main.ts:
```

```ts
import { importProvidersFrom } from "@angular/core";
import { bootstrapApplication, BrowserModule } from "@angular/platform-browser";
import { provideRouter, RouterModule, withComponentInputBinding } from "@angular/router";

bootstrapApplication(HostComponent, {
  providers: [
    importProvidersFrom(
      BrowserModule,
      RouterModule,
    ),
    provideRouter(
      HOST_ROUTES,
      withComponentInputBinding(), //--> For Input Binding (Requires Angular Version 16+)
    )
  ],
})
```

```
calendar/src/main.ts:
```

```ts
import { importProvidersFrom } from "@angular/core";
import { bootstrapApplication, BrowserModule } from "@angular/platform-browser";
import { provideRouter, RouterModule, withComponentInputBinding } from "@angular/router";

bootstrapApplication(CalendarComponent, {
  providers: [
    importProvidersFrom(
      BrowserModule,
      RouterModule,
    ),
    provideRouter(
      CALENDAR_ROUTES,
      withComponentInputBinding(), //--> For Input Binding (Requires Angular Version 16+)
    )
  ],
})
```

```
teams/src/main.ts:
```

```
import { importProvidersFrom } from "@angular/core";
import { bootstrapApplication, BrowserModule } from "@angular/platform-browser";
import { provideRouter, RouterModule, withComponentInputBinding } from "@angular/router";

bootstrapApplication(TeamsComponent, {
  providers: [
    importProvidersFrom(
      BrowserModule,
      RouterModule,
    ),
    provideRouter(
      TEAMS_ROUTES,
      withComponentInputBinding(), //--> For Input Binding (Requires Angular Version 16+)
    )
  ],
})
```

Module Federation Setup

1) Install dependencies for Module Federation:

```
npm i @angular-architects/module-federation
```

2) Migrate microfrontend apps to Module Federation:

```
ng add @angular-architects/module-federation --project calendar --port 4201 --type remote
```

```
ng add @angular-architects/module-federation --project teams --port 4202 --type remote
```

3) Migrate the host app to Module Federation:

```
ng add @angular-architects/module-federation --project host --port 4200 --type dynamic-host
```

4) As a result of the above commands:

```
Main.ts files will have been moved to the bootstrap.ts. Check all projects and fix if AppComponent is
used instead of Standalone Components.
```

```
Index.html files may have changed. Check all projects and fix if AppComponent is used instead of
```

```
Standalone Components.
```

```
In the host project, you can manage microfrontends and their ports through the
host/src/assets/mf.manifest.json file.
```

5) Declare the microfrontend end points in the Host App:

```
host/src/assets/mf.manifest.json:
```

```
{
  "calendar": "http://localhost:4201/remoteEntry.js",
  "teams": "http://localhost:4202/remoteEntry.js",
}
```

6) Expose route files of the microfrontends via their webpack.config.js:

```
calendar/webpack.config.js:
```

```
exposes: {
  './routes': './src/app/routes/calendar.routes.ts'
},
```

```
teams/webpack.config.js:
```

```
exposes: {
  './routes': './src/app/routes/teams.routes.ts'
},
```

7) Fill in the contents of the Route files in each project:

```
calendar/src/app/routes/calendar.routes.ts:
```

```
export const CALENDAR_ROUTES: Routes = [
  { path: '', redirectTo: 'menu', pathMatch: "full" },
  {
    path: 'menu',
    loadComponent: () => import('../calendar.component')
    .then(c => c.CalendarComponent)
  },
  { path: '**', redirectTo: 'menu' },
];
```

```
teams/src/app/routes/teams.routes.ts:
```

```
export const TEAMS_ROUTES: Routes = [
  { path: '', redirectTo: 'menu', pathMatch: "full" },
  {
    path: 'menu',
    loadComponent: () => import('../teams.component')
    .then(c => c.TeamsComponent)
  },
  { path: '**', redirectTo: 'menu' },
];
```

```
host/src/app/routes/host.routes.ts:
```

```
import { loadRemoteModule } from '@angular-architects/module-federation';

export const HOST_ROUTES: Routes = [
  { path: '', redirectTo: "calendar", pathMatch: "full" },
  {
    path: 'calendar',
    loadChildren: () => loadRemoteModule({
      type: 'manifest',
      remoteName: 'calendar', //---> The name that is written in assets/mf.manifest.json
      exposedModule: './routes'
    }).then(r => r.CALENDAR_ROUTES),
  },
  {
    path: 'teams',
    loadChildren: () => loadRemoteModule({
      type: 'manifest',
      remoteName: 'teams', //---> The name that is written in assets/mf.manifest.json
      exposedModule: './routes'
    }).then(r => r.TEAMS_ROUTES),
  },
  { path: "**", redirectTo: 'calendar', pathMatch: "full" }
];
```

Run the Apps

1) Run the Microfrontend Apps:

```
ng serve
```

2) Run the Host App:

```
ng serve
```