Reusing Code between NativeScript and Angular Web Apps

Radu Nicoara
Consultant/Developer
Consultingwerk Software Services Ltd.

- Independent IT consulting organization
- Focusing on **OpenEdge** and related technology
- Located in Cologne, Germany, subsidiaries in UK and Romania
- Customers in Europe, North America, Australia and South Africa
- Vendor of developer tools and consulting services
- Specialized in GUI for .NET, Angular, OO, Software Architecture, Application Integration
- Experts in OpenEdge Application Modernization
Radu Nicoara

- Consultant / Developer
- Using OpenEdge for about 5 years
- Several years of experience in Angular, NativeScript and node.js
- Developed the SmartComponent Library for Angular and NativeScript
- Supports customers, offers consultation for web and mobile architecture and provides trainings and mentoring services
My Story: From Web Developer to Mobile

I used to be a happy web developer working on the SmartComponent Library for Angular package. I was comfortable thinking of Angular as a web framework, I was in my element. All was good in the world and you could almost see the unicorns flying through the sky.
Nothing Lasts Forever

One day, while I was happily coding and admiring the pretty unicorns above my head, our product manager came to me and said: Radu, the web frontend is awesome. But it would be really cool if we also had a mobile frontend. It should ideally run on Android and iOS, but we don’t really want to have to maintain two code bases.

Have fun with it!
My Reaction
Where to start?

I had no previous mobile development experience. The technology that we would use needed to resemble something I was familiar with. At this point, I started paying more attention to articles mentioning Angular as a “platform-agnostic” framework.
Angular Architecture

- Centered around component composition
- Application organised in Modules
  - Angular Modules != ES6 Modules
- Rendering decoupled from DOM
  - „virtual“ DOM for performance

```typescript
import { BrowserModule } from '@angular/platform-browser'; // AOT
import { BrowserModule } from '@angular/platform-browser-dynamic'; // JIT
import { ServerModule } from '@angular/platform-server'; // Server
```
Eureka!
Web application framework
Platform-agnostic framework
NativeScript
Angular
Going Mobile: Enter NativeScript!

- One code base for iOS and Android
- Supports Angular
- Use native iOS and Android libraries out of the box directly through TypeScript with zero-day support
- Supports Angular CLI productivity tools
- Native app performance (not native-like!)
What can be shared?

- Services
- Pipes
- Router configuration
- Components (if implemented with code sharing in mind)
Sharing Component Definitions

- Must be “dumb” components
- Must not directly manipulate the DOM
- Must not use platform specific APIs
- Component UI definitions cannot be shared
Sharing Component Definitions

```typescript
@Component({
  selector: "CustomerList",
  templateUrl: "./customer-list.component.html"
})
export class CustomerListComponent {
  public customers: Customer[] = [ ... ];
}

<ul class="list-group">
  <li class="list-group-item" *ngFor="let customer of customers">
    {{customer.name}}
  </li>
</ul>

<RadListView [items]="customers">
  <ng-template tkListItemTemplate let-item="item">
    {{customer.name}}
  </ng-template>
</RadListView>
```
The Challenge: structuring the project

- Which files can be shared and which are platform specific?
- Dependency management – different platforms require different dependencies
- Automate build processes: npm scripts, gulp, sh/batch scripts?
I HAVE NO IDEA
WHAT I'M DOING
The @nativescript/schematics Collection

- Provides out of the box support for code sharing-friendly project scaffolding using the Angular CLI
- Automagically provides build scripts, dependencies, etc.
- As simple as:

```bash
ng new --collection@nativescript/schematics --name=my-app --shared
```
Build Process

ng serve
ng build [--prod]

tns run [ios/android] --bundle
tns build [android/ios] --bundle
Building for the web

- Angular CLI ignores all NativeScript specific files
- NativeScript specific files have specific terminations which can be seen in tsconfig.json:

  ```json
  "exclude": [
    "**/*.ns.ts",
    "**/*.tns.ts",
    "**/*.android.ts",
    "**/*.ios.ts"
  ]
  ```
Code Separation

- We need to separate web from mobile code
- Platform specific code without conflicts (e.g. ngModule imports)
- Solution: naming convention
Naming Conventions

- .tns before the file extension indicates it is NativeScript specific
- Same file name without .tns termination indicates it is web specific
- Single file without .tns termination indicates that it is shared
- Most common use case is component code, where we need to separate UI definitions
Example Component File Structure

customers
  - customers.component.ts -> shared class definition
  - customers.component.html -> web specific UI definition
  - customers.component.tns.html -> mobile specific UI definition
  - customers.component.css -> web specific stylesheet
  - customers.component.tns.css -> mobile specific stylesheet

The @Component decorator points to the same templateUrl and styleUrls (without .tns extension). This is handled by the build process.
Code Separation: NgModules

- Web and mobile NgModules have different dependencies
- Most common differences:

  - CommonModule
  - HttpClientModule
  - FormsModule
  - RouterModule
  - NativeScriptCommonModule
  - NativeScriptHttpClientModule
  - NativeScriptFormsModule
  - NativeScriptRouterModule
Code Separation: NgModules

- Use naming conventions to maintain two versions of the @NgModule
- Each @NgModule uses different versions of its dependencies
- Angular’s dependency injection provides the right implementation to your service
Example: HttpClient

```typescript
// my-feature.module.ts
@NgModule({
    imports: [
        HttpClientModule
    ]
})
export class MyFeatureModule { }
```

```typescript
// my-feature.module.tns.ts
@NgModule({
    imports: [
        NativeScriptHttpClientModule
    ]
})
export class MyFeatureModule { }
```

```typescript
@Injectable()  // my-http.service.ts – shared NG service
export class MyHttpService {
    constructor(private http: HttpClient) { }
}
```
Questions