IN4MATX 133: User Interface Software

Lecture 14: Separation in Angular

Professor Daniel A. Epstein
TA Jamshir Goorabian
TA Simion Padurean

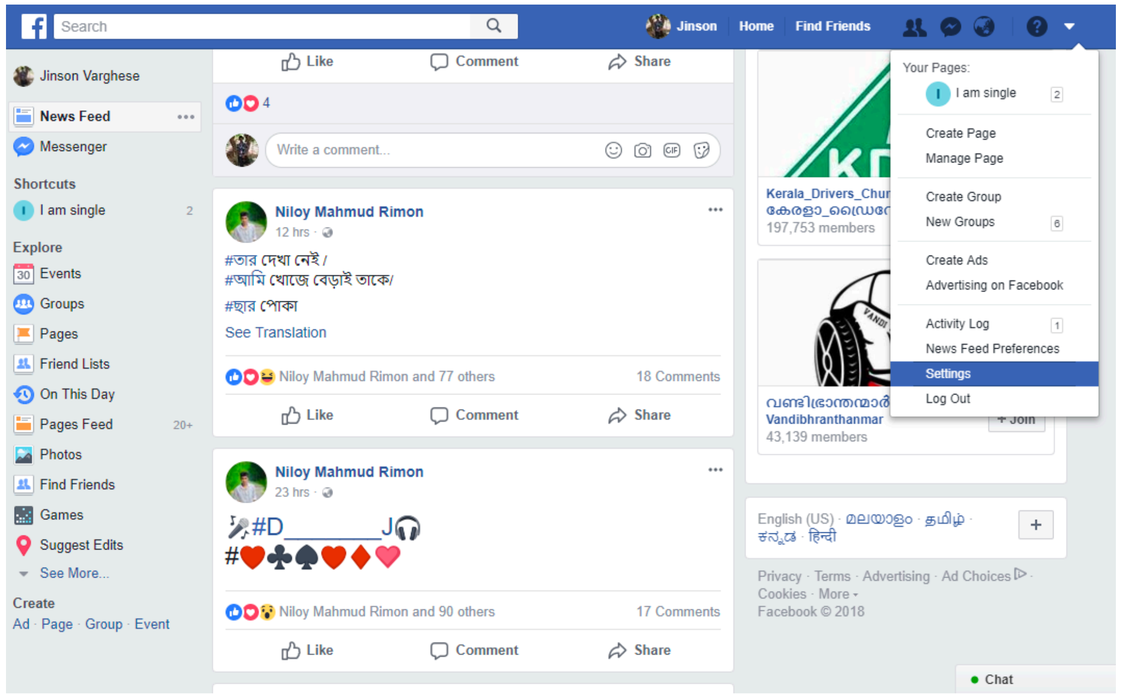
Today's goals

By the end of today, you should be able to…

- Differentiate and explain the roles of Angular components, modules, and services
- Implement a service in Angular
- Navigate Angular's file structure to get started

A “large” client interface

- Hundreds of pages and ways to navigate between pages
- Repeated UI components (status updates)
- Different content, links, etc. displayed for each person



A “large” client interface

- Loading lots of libraries can be slow and expensive
- So Angular supports sectioning parts of projects into distinct modules

Angular modules

- Segment code into a library, similar to a JavaScript library
- A component only imports the modules it needs

Angular modules

- By default, each Angular app has one module, `app.module.ts`
- But an app can create multiple modules to section off code
- `ng generate module [name]`
- Modules can *import* other modules
- Modules also *declare* which components they use
  - When you create a new component (`ng generate component [name]`), it automatically gets added to the declarations for the root module

Angular modules

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { HelloComponent } from './hello/hello.component';
import { DayComponent } from './day/day.component';

@NgModule({
  declarations: [
    AppComponent,
    HelloComponent,
    DayComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Components used

Modules to import

The “root” component of the module

Angular modules

- `BrowserModule` is included by default
  - Required to run any app in the browser
- When creating an Angular project, can specify whether a *Routing* module should be created
  - Routing: defines what URIs to send to what endpoints
  - For Angular, defines what URIs to send to what components

Angular routing

app-routing.module.ts

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { ArtistPageComponent } from './pages/artist-page/artist-page.component';
import { TrackPageComponent } from './pages/track-page/track-page.component';
import { AlbumPageComponent } from './pages/album-page/album-page.component';
import { HomePageComponent } from './pages/home-page/home-page.component';

const routes: Routes = [
  { path: 'artist/:id', component: ArtistPageComponent},
  { path: 'track/:id', component: TrackPageComponent},
  { path: 'album/:id', component: AlbumPageComponent},
  { path: '', component: HomePageComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Listens for any endpoint `artist/:id`

`id` can be retrieved in `album-page.component.ts`

Retrieving route in a component

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-album-page',
  templateUrl: './album-page.component.html',
  styleUrls: ['./album-page.component.css']
})
export class AlbumPageComponent implements OnInit {

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    var albumId = this.route.snapshot.paramMap.get('id');
  }

}
```

"Injecting a service"

Retrieve the id from the URI

Angular services

- Anything not associated with a specific view should be turned into a *service*
  - e.g., getting data from an API, parsing URIs for routing information
- Helps keep components lightweight
- Services can then be *injected* into a component (importing)
- To inject, import the service and retrieve it as a parameter in the constructor
- `ng generate service [name]`

Angular services

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';

@Component({
  selector: 'app-album-page',
  templateUrl: './album-page.component.html',
  styleUrls: ['./album-page.component.css']
})
export class AlbumPageComponent implements OnInit {

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
    var albumId = this.route.snapshot.paramMap.get('id');
  }
}
```

Importing a service

Injecting it

Service can be referenced later

Angular services

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpHeaders } from '@angular/common/http';

@Injectable({
  providedIn: 'root'
})
export class SpotifyService {
 baseUrl:string = 'http://localhost:8888';

  constructor(private http:HttpClient) { }

  private sendRequestToExpress(endpoint:string) {
  }
}
```

Defined as injectable

Services can inject other services!

What module(s) can use this service

HttpClient injected

Import a custom service

```
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute } from '@angular/router';
import { SpotifyService } from '../../services/spotify.service';

@Component({
  selector: 'app-album-page',
  templateUrl: './album-page.component.html',
  styleUrls: ['./album-page.component.css']
})
export class AlbumPageComponent implements OnInit {

  constructor(private route: ActivatedRoute,
private spotifyService:SpotifyService) { }
```

Import service via file structure

Inject it like any other service

Question 

Which of these is best implemented as a *module*, *service*, and *component*?

1. A library which communicates with Snapchat's database
2. The interface and packages needed to create and send a Snap
3. The interface and interaction for laying text over a Snap

A (1) Service, (2) Component, (3) Component

B (1) Service, (2) Module, (3) Component

C (1) Service, (2) Module, (3) Module

D (1) Module, (2) Service, (3) Component

E (1) Module, (2) Module, (3) Module

Angular classes

- Plain-old classes can also be made in Angular
  - Any processing or munging you need to do, for example

- `ng generate class [name]`

```
export class Dataparser {
  public constructor() {
    console.log('Hello, world!');
  }
}
```

Import a class

```
import { Component, OnInit, Input } from '@angular/core';
import { Dataparser } from '../dataparser';

@Component({
  selector: 'app-day',
  templateUrl: './day.component.html',
  styleUrls: ['./day.component.css']
})
export class DayComponent implements OnInit {
  @Input() today:string;

  days = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];

  constructor() {
    var data = new Dataparser();
  }

  ngOnInit() {
  }
}
```

Import class via file structure

Instantiate it like any other class

Import a library

- Since Angular is in TypeScript, it can use any JavaScript or TypeScript library
- Install as normal with npm: `npm install [packagename]`
  - If you want TypeScript typings, don't forget to install `@types/[packagename]`

Import a library

- Since Angular is in TypeScript, it can use any JavaScript or TypeScript library
- Install as normal with npm: `npm install [packagename]`
  - If you want TypeScript typings, don't forget to install `@types/[packagename]`

Import a library

```
import * as chroma from 'chroma-js';
```

Note: different syntax

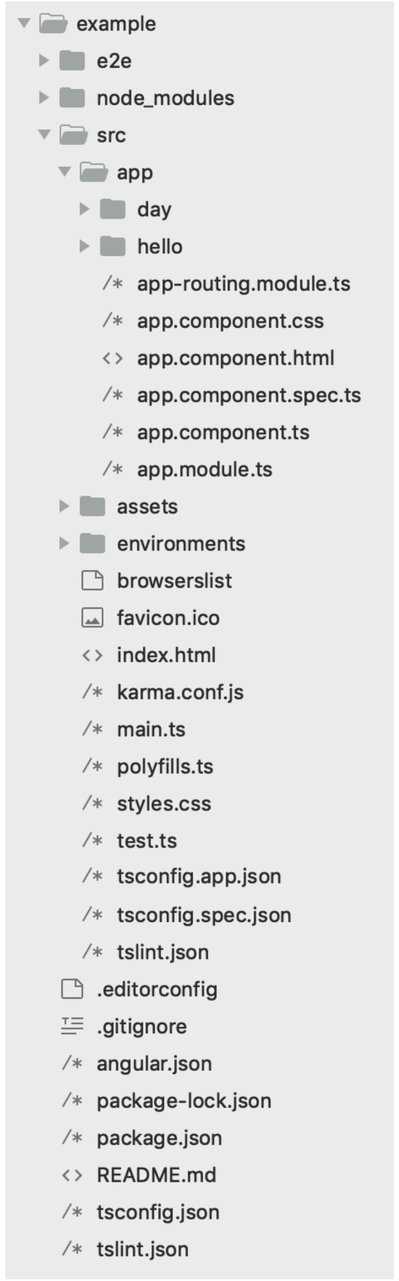
```
export class Dataparser {

  constructor() {
    console.log(chroma('royalblue')); //'#4169e1'
  }
}
```

Can now be referenced

Angular's file structure

- Angular projects generate a *lot* of files
  - There are about 75 in the starter code for A3
- Most are boilerplate



[Live walkthrough of Angular’s structure]

Today’s goals

By the end of today, you should be able to…

- Differentiate and explain the roles of Angular components, modules, and services
- Implement a service in Angular
- Navigate Angular’s file structure to get started

IN4MATX 133: User Interface Software

Lecture 14: Separation in Angular

Professor Daniel A. Epstein
TA Jamshir Goorabian
TA Simion Padurean