Angular Crash Course

Quickly build a frontend web app to connect users to your REST backend

Chris Stone

What is Angular?

- Cross-platform framework for writing UI apps on web, mobile, or desktop
- Write in TypeScript; run in JavaScript
- Manages **templates**, **routing**, and **application state**
- Component-based. Apps are built up of custom HTML-like elements, e.g. `<my-widget property="foo"></my-widget>`, each with backing JavaScript logic

Terminology: Component

A component is a custom HTML-like element defined by code + template

```
@Component({
  selector: 'my-widget',
  templateUrl: './widget.component.html',
})
export class WidgetComponent {
  title = 'hello world';
  users = ['user1', 'user2'];
}
```

Use the template in any HTML by including its selector in some HTML:

```
<my-widget></my-widget>
```

Terminology: Template

A template contains HTML-like markup that defines visual presentation

```
<h1>{{title}}</h1>
<ul>
  <li *ngFor="let user of users">name: {{user.name}}</li>
</ul>
<button type="button" (click)="addUser()">add user</button>
```

Just like HTML, with access to special variables (`title`, `users`) and directives, such as looping (`*ngFor`) and events (`click`)

Terminology: Service

A service is some dependency object that your component can use. Angular injects services into your component's constructor.

```
@Component({
  selector: 'my-widget',
  templateUrl: './widget.component.html',
})
export class WidgetComponent {
  title = 'hello world';
  users = ['user1', 'user2’];

  constructor(private http: HttpClient) { }
}
```

Angular provides many services out of the box, like `HttpClient`, shown here, or you can write your own.

Get started

- Requirements:
  - Node 8+
  - Command line
  - Python (or any HTTP backend will do)
- Follow along here or jump right to source code:
  - https://github.com/ctstone/angular-crash-course

A simple backend (server.py)

```python
from flask import (Flask, jsonify, request)
from flask_cors import (CORS)

app = Flask(__name__)
cors = CORS(app, origins = '*')
users = []

@app.route('/users', methods=['GET'])
def get_users():
  global users
  return jsonify(value = users)

@app.route('/users', methods=['POST'])
def add_user():
  global users
  users.append(request.get_json()['name'])
  return '', 201

if __name__ == '__main__’:
  app.run()
```

Start your backend

- Pip install flask flask-cors
  - CORS allows other sites to talk to this one via the user’s web browser
- Python server.py
- Browse to http://localhost:5000/users (should be empty array)
- curl -X POST -d '{"name": "user1"}' -H 'content-type: application/json' http://localhost:5000/users
- Refresh browser (should see new user name in the array)

Create Angular app

- npm install --global @angular/cli
  - Install the Angular scaffolding/build/serve command line tool
- ng new client --skip-tests
  - “client” is the name of our app. This can be anything
- cd client
- ng serve
  - Host the app on a development server
  - Watches for file changes and rebuilds
  - Available on http://localhost:4200

Add custom Angular component

- Open a code editor (e.g. vscode) at folder “client”
- Open a new command line in “users”
- ng generate component users
  - Scaffolds a new component class called “users”
- Edit src/app/app.component.html. Replace all content with:
  - <app-users></app-users>
- Changes are automatically loaded in browser on file save

Add the HttpClient module

- HttpClient is Angular's preferred HTTP abstraction layer
- It is defined in an external module that must be imported into our app module:
  - Edit src/app/app.module.ts
  - Import the TypeScript class:

    ```
    import { HttpClientModule } from '@angular/common/http';
    ```
  - Import the Angular module into our module definition:

    ```
    imports: [
      BrowserModule,
      HttpClientModule, // <-- add this
    ]
    ```

Add the HttpClient service

- Now we can use HttpClient in our module’s components
- Edit src/app/users/users.component.ts
- Import the TypeScript class

  ```
  import { HttpClient } from '@angular/common/http’;
  ```
- Add the dependency as a constructor parameter

  ```
  constructor(private http: HttpClient) { }
  ```

Use the HttpClient service

- Now we can use the HttpClient utility in our class
- Add a public member variable to your component:

```
export class UsersComponent implements OnInit {
  users: string[];
  ...
```

- Load users array from the backend when the component inits

```
ngOnInit() {
  this.http.get<any>('http://localhost:5000/users')
    .subscribe((resp) => this.users = resp.value);
}
```

Show the users

- All public members are accessible from the component’s template
- Edit src/app/users/users.component.html
- Set content to:

```
<h2>Users</h2>
<ul>
   <li *ngFor="let user of users">{{user}}</li>
</ul>
<p *ngIf="!users?.length">no users</p>
```

Add a user

- Edit src/app/users.component.ts
- Add new class method:

```
addUser() {
  const name = window.prompt(`What is the user's name?`);
  this.http.post('http://localhost:5000/users', {name})
    .subscribe(() => {
      this.ngOnInit(); // in practice, chain observables with
`flatMap` instead of nested `subscribe`
    });
}
```

- Edit src/app/users.component.html
- Add a button to invoke the method:

```
<button type="button" (click)="addUser()">add a user</button>
```

Next steps

- Add UI flourishes with Twitter’s Bootstrap framework (JS+CSS utils)
- Add navigable pages to your app with the @angular/routing module
- Add user interaction with the @angular/forms module
- Add friendly icons from Font Awesome
- Build for production (optimized, static html+js output) with `ng build --prod`
- Configure production web server (e.g. Apache, IIS, Nginx) to route all requests to index.html