Day - 10

17.02.2025

API fetching

Import HttpClientModule

First, you need to import HttpClientModule in your Angular application. This is usually done in the app.module.ts file.

```
import { HttpClientModule } from '@angular/common/http';


@NgModule({
  declarations: [
    // your components
  ],
  imports: [
    BrowserModule,
    HttpClientModule, // Add HttpClientModule here
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Create a Service to Handle API Requests

It's a best practice to create a service to handle all HTTP requests. You can generate a service using the Angular CLI:

```
ng generate service api
```

Use the Service in a Component

Inject the ApiService into your component and call the methods to fetch data.

```
import { Component, OnInit } from '@angular/core';
import { ApiService } from './api.service';

@Component({
  selector: 'app-root',
  template: `
    <h1>Posts</h1>
    <ul>
      <li *ngFor="let post of posts">{{ post.title }}</li>
    </ul>
  `
})
export class AppComponent implements OnInit {
  posts: any[] = [];

  constructor(private apiService: ApiService) { }

  ngOnInit(): void {
    this.apiService.getPosts().subscribe(
      (data) => {
        this.posts = data;
      },
      (error) => {
        console.error('Error fetching posts:', error);
      }
    );
  }
}
```

Handle Errors

Always handle errors when making HTTP requests. You can use the error callback in the subscribe method or use RxJS operators like catchError.

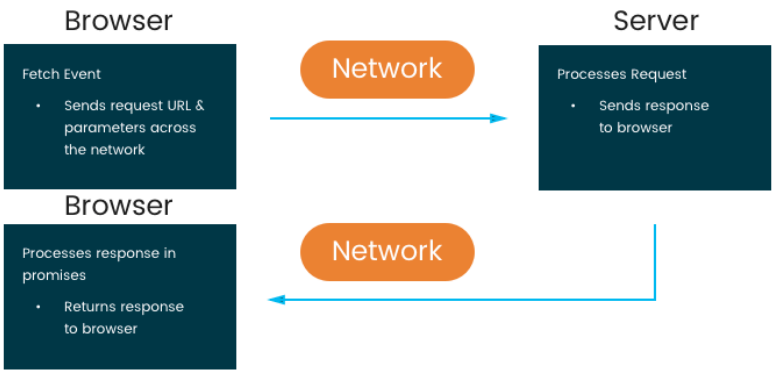
```
import { catchError } from 'rxjs/operators';
import { of } from 'rxjs';

this.apiService.getPosts().pipe(
  catchError((error) => {
    console.error('Error:', error);
    return of([]); // Return an empty array or default value
  })
).subscribe((data) => {
  this.posts = data;
});
```

Testing API Calls

You can test your API calls using tools like Postman or by writing unit tests in Angular.

Example unit test for the service:



Default Folder Structure

```
my-angular-app/
├── e2e/                                  # End-to-end tests (using Protractor by default)
├── node_modules/                         # Installed npm packages and dependencies
├── src/                                  # Application source code
│   ├── app/                              # Main application code
│   │   ├── components/                   # Reusable components
│   │   ├── services/                     # Services for business logic and API calls
│   │   ├── models/                       # Data models (optional)
│   │   ├── interfaces/                   # TypeScript interfaces (optional)
│   │   ├── guards/                       # Route guards (optional)
│   │   ├── pipes/                        # Custom pipes (optional)
│   │   ├── directives/                   # Custom directives (optional)
│   │   ├── app.component.ts              # Root component
│   │   ├── app.component.html            # Root component template
│   │   ├── app.component.css             # Root component styles
│   │   ├── app.module.ts                 # Root module
│   │   └── app-routing.module.ts         # Routing configuration
│   ├── assets/                           # Static assets (images, fonts, etc.)
│   ├── environments/                     # Environment-specific configuration files
│   │   ├── environment.ts                # Development environment
│   │   └── environment.prod.ts           # Production environment
│   ├── styles.css                        # Global styles
│   ├── main.ts                           # Application entry point
│   ├── polyfills.ts                      # Polyfills for browser compatibility
│   ├── test.ts                           # Unit test entry point
│   └── index.html                        # Main HTML file
├── .editorconfig                         # Editor configuration
├── .gitignore                            # Git ignore file
├── angular.json                          # Angular CLI configuration
├── package.json                          # npm dependencies and scripts
├── README.md                             # Project documentation
├── tsconfig.json                         # TypeScript configuration
└── tslint.json                           # TSLint configuration (for code linting)
```

`app/`	Contains the component files in which your app logic and data are defined. See details in App source folder below.
`assets/`	Contains image files and other asset files to be copied as-is when you build your application.
`environments/`	Contains build configuration options for particular target environments. By default there is an unnamed standard development environment and a production ("prod") environment. You can define additional target environment configurations.
`browserslist`	Configures sharing of target browsers and Node.js versions among various front-end tools. See Browserslist on GitHub for more information.
`favicon.ico`	An icon to use for this app in the bookmark bar.
`index.html`	The main HTML page that is served when someone visits your site. The CLI automatically adds all JavaScript and CSS files when building your app, so you typically don't need to add any `<script>` or `<link>` tags here manually.
`main.ts`	The main entry point for your app. Compiles the application with the JIT compiler and bootstraps the application's root module (AppModule) to run in the browser. You can also use the AOT compiler without changing any code by appending the `--aot` flag to the CLI `build` and `serve` commands.