Lesson 7 - HTTP Client and APIs

February 23, 2025

What We'll Cover:

1. Setting Up HttpClient (Importing HttpClientModule)
2. Making GET, POST, PUT, DELETE Requests
3. Handling API Responses and Errors
4. Using async Pipe for Observables
5. Mocking a Backend with a JSON Server (Optional)

Setting Up HttpClient

- **Concept**: Angular’s tool for HTTP requests
- **Setup**: Import HttpClientModule
- **Example**:

```
import { HttpClientModule } from '@angular/common/http';
@Component({
  standalone: true,
  imports: [HttpClientModule],
  template: `...`
})
export class AppComponent {}
```

Service:

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


@Injectable({ providedIn: 'root' })
export class ApiService {
  constructor(private http: HttpClient) {}
}
```

Key Point: HttpClient returns Observables

Making HTTP Requests

- **Concept**: Interact with APIs using HTTP methods
- **Examples**:

```
@Injectable({ providedIn: 'root' })
export class ApiService {
  private url = 'https://api.example.com/tasks';
  constructor(private http: HttpClient) {}
  getTasks() { return this.http.get<string[]>(this.url); }
  addTask(task: string) { return this.http.post(this.url, { task }); }
  updateTask(id: number, task: string) { return this.http.put(`${this.url}/${id}`, { task }); }
  deleteTask(id: number) { return this.http.delete(`${this.url}/${id}`); }
}
```

Key Point: Methods return Observables—subscribe to get data

Handling Responses and Errors

- **Concept**: Process data or catch issues
- **Component Example**:

```
@Component({
  standalone: true,
  imports: [CommonModule, HttpClientModule],
  template: `<ul>@for (task of tasks; track task) { <li>{{ task }}</li> }</ul>`
})
export class TaskListComponent {
  tasks: string[] = [];
  constructor(private apiService: ApiService) {
        this.apiService.getTasks().subscribe({
        next: (data) => this.tasks = data,
        error: (err) => console.error('Error:', err)  }); }}
```

Error Handling: Use catchError in service (optional)

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

getTasks() {
  return this.http.get<string[]>(this.url).pipe(
      catchError(() => of([])) // Fallback to empty array
  );
}
```

Using async Pipe

- **Concept**: Simplify Observables in templates
- **Setup**:
    - Return Observable from service without subscribing
    - Use async in template
- **Example**:

```
@Component({
  standalone: true,
  imports: [CommonModule, HttpClientModule],
  template: `
      <ul>@for (task of tasks$ | async; track task) { <li>{{ task }}</li> }</ul>
  `})
export class TaskListComponent {
  tasks$ = this.apiService.getTasks();
  constructor(private apiService: ApiService) {}}
```

Key Point: No manual subscribe, auto-unsubscribes

Mocking a Backend (Optional)

- **Concept**: Test APIs without a real server
- **Tool**: JSON Server (npm package)
- **Steps**:

    1. Install: npm install -g json-server
    2. Create db.json:

{ "tasks": ["Task 1", "Task 2"] }

1. Run: json-server --watch db.json
2. Update service URL: http://localhost:3000/tasks

Benefit: Simulate GET/POST/PUT/DELETE

Putting It Together

- **Mini-Project**: Task app with API
- **Service**:

```
@Injectable({ providedIn: 'root' })
export class TaskService {
  private url = 'http://localhost:3000/tasks';
  constructor(private http: HttpClient) {}
  getTasks() { return this.http.get<string[]>(this.url); }
  addTask(task: string) { return this.http.post(this.url, { task }); }
}
```

Component:

```
@Component({
  standalone: true,
  imports: [CommonModule, FormsModule, HttpClientModule],
  template: `
        <input [(ngModel)]="newTask" (keyup.enter)="addTask()">
        <ul>@for (task of tasks$ | async; track task) { <li>{{ task }}</li> }</ul>
  `
})
export class TaskComponent {
  tasks$ = this.taskService.getTasks();
  newTask = '';
  constructor(private taskService: TaskService) {}
  addTask() {this.taskService.addTask(this.newTask).subscribe(() => {   this.tasks$ = this.taskService.getTasks();
this.newTask = '';    }); }}
```

Key Takeaways

- HttpClient connects to APIs with HttpClientModule
- Use GET/POST/PUT/DELETE for CRUD operations
- Handle responses with subscribe or async
- async pipe simplifies Observable usage
- Mock backends with JSON Server for testing