10

Making HTTP Requests

HttpClientModule, HttpClient, Sending and Receiving data, Interceptors

HttpClientModule

- Angular has HttpClientModule that allows to perform and manage HTTP requests
- HttpClientModule provides the **HttpClient** service for client-server communication
  - Based on RxJS Observables
- Import **HttpClientModule**

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

@NgModule({
  imports: [BrowserModule, HttpClientModule],
  …
})
export class AppModule { }
```

Making requests

- HttpClient has methods to make Http requests
  - `VERB<TResponse>(url: string, body: any, options?: RequestOptions)`
  - Verb: get, post, put, delete, patch, head
  - Body not available for get

```
let data: Car = { id: 0, make: 'VW', model: 'T-Roc' };
return httpClient.post<Car>(this._carsUrl, data);
```

- Use RequestOptions to specify
  - Headers, query params, credentials, return types,...etc

```
const params = new HttpParams().set('q', 'tesla’);
return httpClient.get<Car[]>(this._carsUrl, {params})
```

Sending and Receiving data (Example)

```
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs/Observable';

@Injectable()
export class AuthService {
  constructor(private http: HttpClient) { }

  private _baseUrl = ‘localhost:3000/login';  // URL to web api

  login(email:string, password: string): Observable<User> {
    return this.http.post<User>(this._baseUrl, {email, password});
  }
}
```

Making requests

- HttpClient methods returns
  - `Observable<Tresponse>`
  - TResponse: response type

- Use {“responseType” : ‘blob’} if you want to return a file
  - ‘arraybuffer’, ‘text’ ,’json’ are other options
  - ‘json’ is the default

- If you need response properties, there are options availble
  - status, statusText, totalBytes, ...

Sending and Receiving data

The golden rule

Always separate client-server communication from your ***Components****, use* ***Services*** *instead*

Making requests

- Don't make requests in a component constructor
  - An async call in constructor is generally a bad idea

- Use events or lifecycle hooks

```
onSubmit() {
  this._authService.login(
      .subscribe({
         next:(value)=>{this.user = value},
         error: (error)=> {this.errorMessage = error}
         }
}
```

Service returns an Observable<User>

HTTP Interceptors

- Angular can intercept any HTTP request or response
  - Modify the request / response

- Great for repetitive tasks
  - Logging
  - Add authentication headers
  - Progress checking
  - Client-side caching

HTTP Interceptors

- An HTTP interceptor must implement **HttpInterceptor**
  - Implement the method `intercept(request, next)`
  - `request`: represnets the Http request object
  - `next:` http handler represents the next interceptor

- Interceptors has to be registered in DI
  - Using the **HTTP_INTERCEPTORS** injection token
  - We can define multiple interceptors

HTTP Interceptors (Authentication)

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {
   constructor(private auth: AuthService) {}

      intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

      // Get the auth header from the service
      const authHeader = this.auth.getAuthorizationHeader();

      // Clone the request to add the new header
      const authReq = req.clone({headers: req.headers.set('Authorization', authHeader)});

      // Pass on the cloned request instead of the original request
      return next.handle(authReq);
   }
}
```

HTTP Interceptors (Logging)

```
@Injectable()
export class LoggingInterceptor implements HttpInterceptor {
   constructor(private log: LoggingService) {}

   intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

      // Use RxJS operators to chain
      return next.handle(req).pipe(
         map((event: HttpEvent<any>) => {
            if (event instanceof HttpResponse) {
               log.info(new Date(), event)
            }
         return event;
      }));
  }
}
```

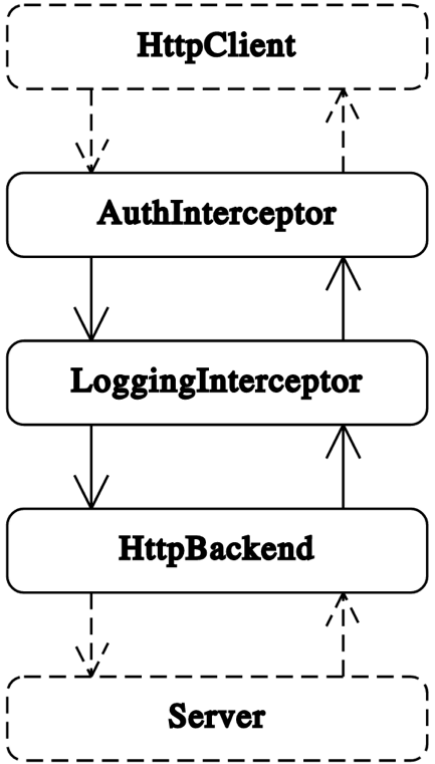
Providing HTTP Interceptors

```
providers: [
      {
            provide: HTTP_INTERCEPTORS,
            useClass: AuthInterceptor,
            multi: true
      },
      {
            provide: HTTP_INTERCEPTORS,
            useClass: LoggingInterceptor,
            multi: true
      }

]
```

Interception order

- The order of interception respect **the order of providers**
- The last interceptor is **ALWAYS** HttpBackend
- HttpBackend is responsible for dispatching the HTTP requests



LAB 10

Make HTTP requests

Useful links

CodeCademy

- **HTML**: www.codecademy.com/learn/learn-html
- **CSS** www.codecademy.com/learn/learn-css
- **Javascript**: www.codecademy.com/learn/introduction-to-javascript
- **Typescript**: www.codecademy.com/learn/learn-typescript

Pluralsight

- www.pluralsight.com/paths/building-websites-with-html-css-and-javascript