HTTP CLIENT

🕮 HTTP CLIENT

The HttpClient in @angular/common/http offers a simplified client HTTP API for Angular applications that rests on the XMLHttpRequest interface exposed by browsers. Additional benefits of HttpClient include testability features, typed request and response objects, request and response interception, Observable apis, and streamlined error handling.

TBC

🕮 SETUP

- HttpClientModule

Service

- HttpClient

🕮 HTTP CLIENT

```
getConfig() {
  // now returns an Observable of Config
  return this.http.get<Config>(this.configUrl);
}
```

TBC

🕮 HTTP CLIENT

```
config: Config;

showConfig() {
  this.configService.getConfig()
    // clone the data object, using its known Config shape
    .subscribe((data: Config) => this.config = { ...data });
}
```

TBC

🕮 HTTP CLIENT RESPONSE

```
getConfigResponse(): Observable<HttpResponse<Config>> {
  return this.http.get<Config>(
    this.configUrl, { observe: 'response' });
}
```

TBC

🕮 HTTP CLIENT RESPONSE

```
showConfigResponse() {
  this.configService.getConfigResponse()
    // resp is of type `HttpResponse<Config>`
    .subscribe(resp => {
      // display its headers
      const keys = resp.headers.keys();
      this.headers = keys.map(key =>
        `${key}: ${resp.headers.get(key)}`);

      // access the body directly, which is typed as `Config`.
      this.config = { ... resp.body };
    });
}
```

TBC

🕮 HTTP CLIENT ERRORS

```
showConfig() {
  this.configService.getConfig()
    .subscribe(
      (data: Config) => this.config = { ...data }, // success path
      error => this.error = error // error path
    );
}
```

TBC

🕮 HTTP CLIENT ERRORS

```
private handleError(error: HttpErrorResponse) {
  if (error.error instanceof ErrorEvent) {
    // A client-side or network error occurred. Handle it accordingly.
    console.error('An error occurred:', error.error.message);
  } else {
    // The backend returned an unsuccessful response code.
    // The response body may contain clues as to what went wrong,
    console.error(
      `Backend returned code ${error.status}, ` +
      `body was: ${error.error}`);
  }
  // return an observable with a user-facing error message
  return throwError(
    'Something bad happened; please try again later.');
};
```

🕮 HTTP CLIENT ERRORS

```
getConfig() {
  return this.http.get<Config>(this.configUrl)
    .pipe(
      catchError(this.handleError)
    );
}
```

TBC

🕮 HTTP CLIENT ERRORS

```
getConfig() {
  return this.http.get<Config>(this.configUrl)
    .pipe(
      retry(3), // retry a failed request up to 3 times
      catchError(this.handleError) // then handle the error
    );
}
```

🕮 HEADERS

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


const httpOptions = {
  headers: new HttpHeaders({
    'Content-Type':  'application/json',
    'Authorization': 'my-auth-token'
  })
};
```

🕮 POST

```
/** POST: add a new hero to the database */
addHero (hero: Hero): Observable<Hero> {
  return this.http.post<Hero>(this.heroesUrl, hero, httpOptions)
    .pipe(
      catchError(this.handleError('addHero', hero))
    );
}
```

🕮 POST

```
this.heroesService
    .addHero(newHero)
    .subscribe(hero => this.heroes.push(hero));
```

🕮 DELETE

```
/** DELETE: delete the hero from the server */
deleteHero (id: number): Observable<{}> {
  const url = `${this.heroesUrl}/${id}`; // DELETE api/heroes/42
  return this.http.delete(url, httpOptions)
    .pipe(
      catchError(this.handleError('deleteHero'))
    );
}
```

🕮 DELETE

```
this.heroesService
    .deleteHero(hero.id)
    .subscribe();
```

TBC

🕮 PUT

```
/** PUT: update the hero on the server. Returns the updated hero upon success. */
updateHero (hero: Hero): Observable<Hero> {
  return this.http.put<Hero>(this.heroesUrl, hero, httpOptions)
    .pipe(
      catchError(this.handleError('updateHero', hero))
    );
}
```

TBC

🕮 INTERCEPTORS

```
@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(private auth: AuthService) {}

  intercept(req: HttpRequest<any>, next: HttpHandler) {
    // Get the auth token from the service.
    const authToken = this.auth.getAuthorizationToken();

    // Clone the request and replace the original headers with
    // cloned headers, updated with the authorization.
    const authReq = req.clone({
      headers: req.headers.set('Authorization', authToken)
    });

    // send cloned request with header to the next handler.
    return next.handle(authReq);
  }
}
```

```
providers: [
  {
    provide: HTTP_INTERCEPTORS,
    useClass: TokenInterceptor,
    multi: true
  }
]
```

HERE GOES!

TBC