HTTP

In order to start making HTTP calls from our Angular app we need to import the `angular/http` module and register for HTTP services. It supports both XHR and JSONP requests exposed through the `HttpModule` and `JsonpModule` respectively. In this section we will be focusing only on the `HttpModule`.

Setting up angular/http

In order to use the various HTTP services we need to include `HttpModule` in the imports for the root `NgModule`. This will allow us to access HTTP services from anywhere in the application.

```
...
import { AppComponent } from './app.component'
import { HttpModule } from '@angular/http';

@NgModule({
  imports: [
    BrowserModule,
    ReactiveFormsModule,
    FormsModule,
    HttpModule
  ],
  providers: [SearchService],
  declarations: [AppComponent],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

Making HTTP Requests

To make HTTP requests we will use the `Http` service. In this example we are creating a `SearchService` to interact with the Spotify API.

```
import { Http } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Observable';
import 'rxjs/add/operator/map';

@Injectable()
export class SearchService {

  constructor(private http: Http) {}

  search(term: string) {
    return this.http
      .get('https://api.spotify.com/v1/search?q=' + term + '&type=artist')
      .map(response => response.json());
  }
}
```

View Example

Here we are making an HTTP GET request which is exposed to us as an observable. You will notice the `.map` operator chained to `.get`. The `Http` service provides us with the raw response as a string. In order to consume the fetched data we have to convert it to JSON.

In addition to `Http.get()`, there are also `Http.post()`, `Http.put()`, `Http.delete()`, etc. They all return observables.

Catching Rejections

To catch rejections we use the subscriber's `error` and `complete` callbacks.

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

@Injectable()
export class AuthService {

  constructor(private http: Http) {}

  login(username, password) {
    const payload = {
      username: username,
      password: password
    };

    this.http.post(`${ BASE_URL }/auth/login`, payload)
      .map(response => response.json())
      .subscribe(
        authData => this.storeToken(authData.id_token),
        (err) => console.error(err),
        () => console.log('Authentication Complete')
      );
  }
}
```

Catch and Release

We also have the option of using the `.catch` operator. It allows us to catch errors on an existing stream, do something, and pass the exception onwards.

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

@Injectable()
export class SearchService {

  constructor(private http: Http) {}

  search(term: string) {
    return this.http.get('https://api.spotify.com/v1/dsds?q=' + term + '&type=artist')
      .map((response) => response.json())
      .catch((e) => {
        return Observable.throw(
          new Error(`${ e.status } ${ e.statusText }`)
        );
      });
  }
}
```

View Example

It also allows us to inspect the error and decide which route to take. For example, if we encounter a server error then use a cached version of the request otherwise re-throw.

```
@Injectable()
export class SearchService {

  ...

  search(term: string) {
    return this.http.get(`https://api.spotify.com/v1/dsds?q=${term}&type=artist`)
      .map(response => response.json())
      .catch(e => {
        if (e.status >==  500) {
          return cachedVersion();
        } else {
          return Observable.throw(
            new Error(`${ e.status } ${ e.statusText }`)
          );
        }
      });
  }
}
```

Cancel a Request

Cancelling an HTTP request is a common requirement. For example, you could have a queue of requests where a new request supersedes a pending request and that pending request needs to be cancelled.

To cancel a request we call the `unsubscribe` function of its subscription.

```
@Component({ /* ... */ })
export class AppComponent {
  /* ... */

  search() {
    const request = this.searchService.search(this.searchField.value)
      .subscribe(
        result => { this.result = result.artists.items; },
        err => { this.errorMessage = err.message; },
        () => { console.log('Completed'); }
      );

    request.unsubscribe();
  }
}
```

View Example

Retry

There are times when you might want to retry a failed request. For example, if the the user is offline you might want to retry a few times or indefinitely.

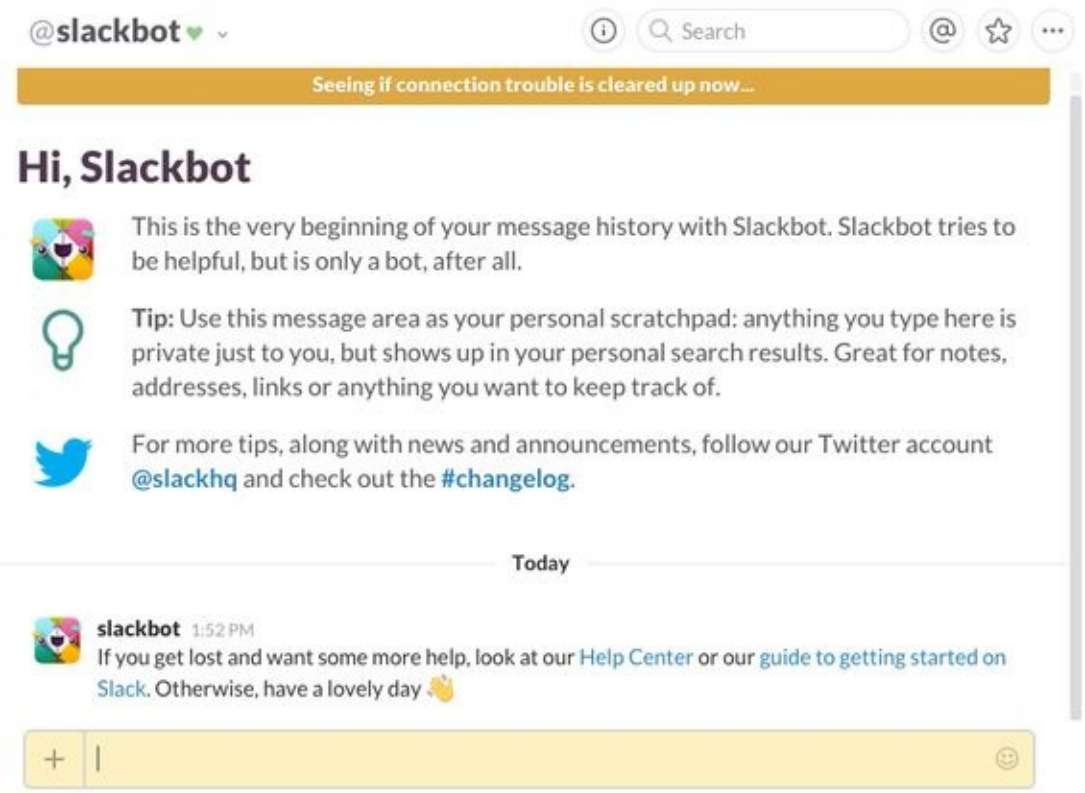

Figure: Retry example from Slack

Use the RxJS `retry` operator. It accepts a `retryCount` argument. If not provided, it will retry the sequence indefinitely.

Note that the error callback is not invoked during the retry phase. If the request fails it will be retried and only after all the retry attempts fail the stream throws an error.

```
import { Http } from '@angular/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs/Rx';

@Injectable()
export class SearchService {

  constructor(private http: Http) {}

  search(term: string) {
    let tryCount = 0;
    return this.http.get('https://api.spotify.com/v1/dsds?q=' + term + '&type=artist')
      .map(response => response.json())
      .retry(3);
  }
}
```

View Example

Requests as Promises

The observable returned by Angular http client can be converted it into a promise.

> We recommend using observables over promises. By converting to a promise you will be lose the ability to cancel a request and the ability to chain RxJS operators.

```
import { Http } from '@angular/http';
import { Injectable } from '@angular/core';
import 'rxjs/add/operator/map';
import 'rxjs/add/operator/toPromise';

@Injectable()
export class SearchService {

  constructor(private http: Http) {}

  search(term: string) {
    return this.http
      .get(`https://api.spotify.com/v1/search?q=${term}&type=artist`)
      .map((response) => response.json())
      .toPromise();
  }
}
```

We would then consume it as a regular promise in the component.

```
@Component({ /* ... */ })
export class AppComponent {
    /* ... */

    search() {
        this.searchService.search(this.searchField.value)
          .then((result) => {
        this.result = result.artists.items;
      })
      .catch((error) => console.error(error));
    }
}
```