Documentation Project- Framework Design

Ionic Angular Framework

Enășoae Simona
Group 248-1

Selected project: a PoC app about how to use a given application framework.

A chat application
Features:

- Login using a username.
- Send messages to a specific Room.
- Display the list of Rooms.
- Display messages for a specific Room.
- Logout.

@ionic/angular combines the core Ionic experience with the tooling and APIs that are made for Angular. Ionic supports Angular 6.0.0 and up.

The Angular Router is one of the most important libraries in an Angular application.

Handling Redirects

In the application, router redirects were used. Redirects work the same way that a typical route object does, but just includes a few different keys.

```
const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'home', loadChildren: () => import('./home/home.module').then( m
 => m.HomePageModule)},
  {
    path: 'chat', loadChildren: () => import('./chat/chat.module').then( m
 => m.ChatPageModule)
  },
];
```

In the example, it is the index path of the app. Then if we load that, we redirect to the home route. The last key of pathMatch is required to tell the router how it should look up the path. Since we use ‘full’, we're telling the router that we should compare the full path.

We navigate in our app by using the router API.

```
 constructor(private router: Router  ) {}
```

```
            this.router.navigate(['chat']);
```

Local Storage
To keep the user logged in, I used **localStorage**. It is very easy to use it.

```
            localStorage.setItem('token', res.token);
```

```
            localStorage.getItem('token');
```

getItem - returns the current value associated with the given key, or null if the given key does not exist in the list associated with the object.
setItem - sets the value of the pair identified by key to value, creating a new key/value pair if none existed for key previously.

Structural Directives - ngIf and ngFor

A structural directive adds and removes elements in the DOM tree. **ngFor** adds DOM elements for each item from an array. A simple ngFor example looks like this:

```
<ion-list>
    <ion-item *ngFor="let item of rooms" (click)="showMessages(item)">
      {{ item }}
    </ion-item>
  </ion-list>
```

This example creates an ion-item component for each element in the array and adds it to the DOM. The code creates a local variable item that can be referenced inside the ngFor loop. The rooms array is a public instance variable in the TypeScript code.

The ***ngIf** directive is most commonly used to conditionally show an inline template, as seen in the following example. The default else template is blank.

```
<ion-button *ngIf="item.mustSend" (click)="sendMessAgain(item)"> Not send<
/ion-button>
```

NgModel

NgModel creates a FormControl instance from a domain model and binds it to a form control element**.** In the app, I used it to bind the message sent by the user.

```
 <ion-input [(ngModel)]="message" placeholder="Message"></ion-input>
```

BehaviorSubject

A **BehaviorSubject** is a type of observable (i.e. a stream of data that we can subscribe to like the observable returned from HTTP requests in Angular). A type of observable because it is a little different to a standard observable. We subscribe to a BehaviourSubject just like we would a normal observable, but the benefit of a BehaviourSubject for our purposes is that:

- It will always return a value, even if no data has been emitted from its stream yet
- When you subscribe to it, it will immediately return the last value that was emitted immediately (or the initial value if no data has been emitted yet)

We are going to use the BehaviorSubject to hold the data that we want to access throughout the application. This will allow us to:

- Just load data once, or only when we need to.
- Ensure that some valid value is always supplied to whatever is using it (even if a load has not finished yet).
- Instantly notify anything that is subscribed to the BehaviorSubject when the data changes.

Example:

```
rivate messagesSubject: BehaviorSubject<Message[]>;
```

When I want to update that data I just need to call the next method on the BehaviorSubject.

```
            this.messagesSubject.next(messages);
```

HttpClient

First I include another Angular module to make HTTP requests. This is already installed, I simply need to load and add it inside src/app/app.module.ts like this:

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

  imports: [BrowserModule, AppRoutingModule, HttpClientModule],
```

This is how I create an instance of HttpClientObject.

```
    constructor(private http: HttpClient) {
```

I used the htpp object to send a new message to the server using post method.

```
sendMessage(message: string, room: string) {
        return this.http.post<Message>(`${httpServerUrl}/message`, {text:
message, room: room}, this.httpOptionsToken())
            .pipe(tap(res => {
                . . .
            }, err => {
                . . .
            }));
    }
```