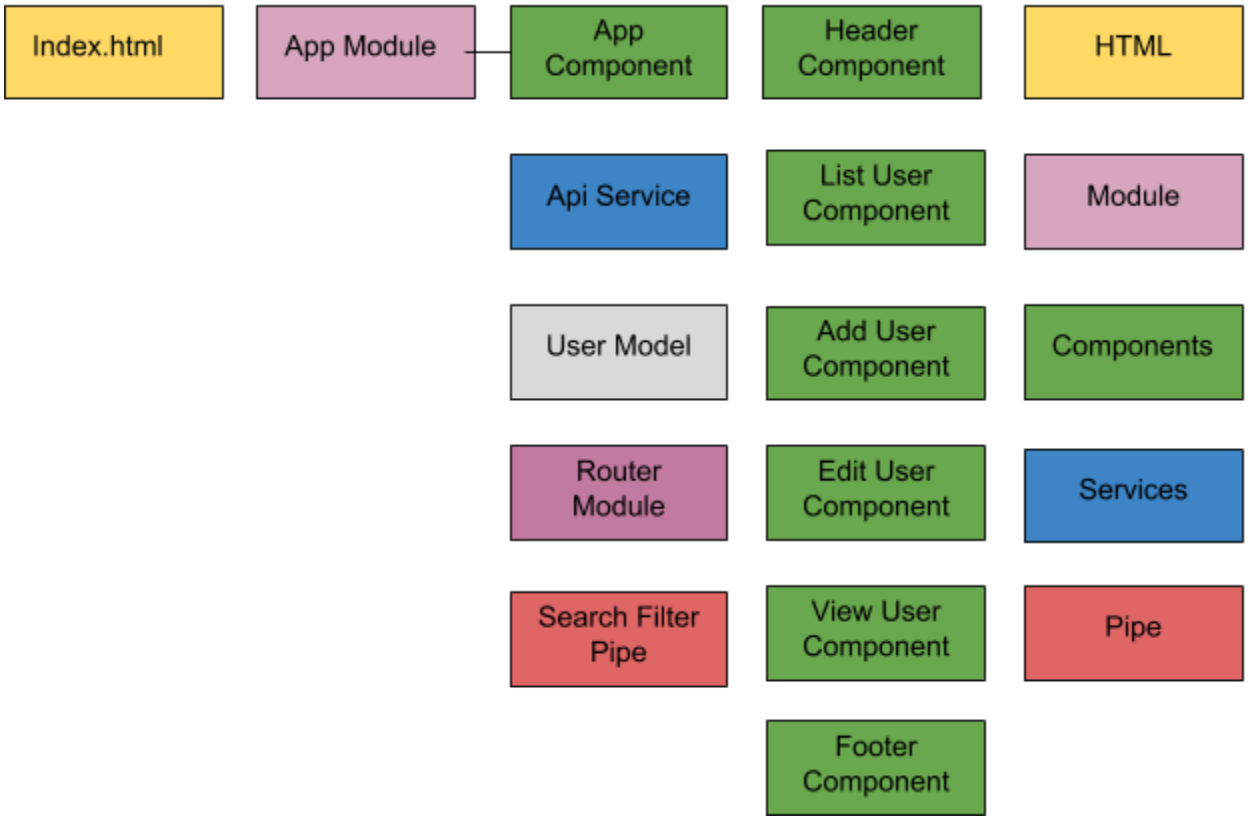
Documentation

This is the brief document regarding the contact information application built using Angular 6 and Bootstrap 4. Making use of latest reactive library RxJS Observables and reactive form validations.

Application Architecture



Exploring Angular 6

Trying to make use of maximum possible angular functionality with Components, Directives, REST API Service, Routing, Modules, Reactive Forms, Pipes.

List of Components

add-user.component.ts
edit-user.component.ts
view-user.component.ts
list-user.component.ts
header.component.ts
footer.component.ts

User Model

```
export interface User {
  id: number;
  firstname: string;
  lastname: string;
  email: string;
  phone: number;
  status: boolean;
}
```

List of Services

Api.service.ts

List of API calls using **Httpclient** and **Observables** for handling data:

API CALLS

```
//API GET: Users
public getAllUsers(): Observable<User[]> {
  return this.http.get<User[]>(API_URL + '/users').pipe(
    catchError(this.handleError)
  );
}

//API POST: User
public createUser(user: User): Observable<User> {
```

```
  return this.http
  .post<User>(API_URL + '/users', user).pipe(
    retry(2),
    catchError(this.handleError)
  );
}
// //API GET /users/:id
public getUserById(userid: number): Observable<User> {
  return this.http
  .get<User>(API_URL + '/users/' + userid).pipe(
    retry(2),
    catchError(this.handleError)
  );
}
//API PUT /users/:id
public updateUser(user: User): Observable<User> {
  return this.http
  .put<User>(API_URL + '/users/' + user.id, user).pipe(
    retry(2),
    catchError(this.handleError)
  );
}
//API DELETE /users/:id
public deleteUser(userid: number): Observable<null> {
  return this.http
  .delete<null>(API_URL + '/users/' + userid).pipe(
    retry(2),
    catchError(this.handleError)
  );
}

//Error Handler
public handleError(error: HttpErrorResponse | any) {
console.error('ApiService::handleError', error);
return Observable.throw(error.message || 'Server Error');
}
```

Reactive Form Validation

Making use of latest Angular 6 reactive form for form submit and validation using **FormBuilder, FormGroup, FormControl**

Form builder and Form Validation:

```
this.addUserForm = this.formBuilder.group({
    id: [''],
    firstname: [
      '', Validators.compose([
        Validators.required
      ])
    ],
    lastname: [
      '', Validators.compose([
        Validators.required
      ])
    ],
    email: [
      '', Validators.compose([
        Validators.required,
        Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,63}$')
      ])
    ],
    phone: [
      '', Validators.compose([
        Validators.required,
        Validators.minLength(10),
        Validators.maxLength(10),
        Validators.pattern('[0-9]*')
      ])
    ],
    status: [
      '', Validators.compose([
        Validators.required
      ])
    ]
```

```
  })
```

App Routing using **RouterModule, Routes :**

```
const routes: Routes = [

 { path: '', redirectTo: '/list', pathMatch: 'full'},

 { path: 'add', component: AddUserComponent },
 { path: 'view/:id', component: ViewUserComponent },
 { path: 'edit/:id', component: EditUserComponent },
 { path: 'list', component: UserListComponent }
];
```

Custom Pipe Search Filter

Pipe is used for filtering the user list values. Search Filter pipe filters the user table based on the firstname of user.

```
export class SearchFilterPipe implements PipeTransform {

 transform(value: any, args?: any): any {
  if (!args) {
    return value;
  }

  return value.filter(items => {
    return items.firstname.startsWith(args) === true;
  });
 }

}
```

Unit Test

Unit test using Karma and Jasmine.

```
beforeEach(async(() => {
  TestBed.configureTestingModule({
    declarations: [ AddUserComponent ]
  })
  .compileComponents();
}));

beforeEach(() => {
  fixture = TestBed.createComponent(AddUserComponent);
  component = fixture.componentInstance;
  fixture.detectChanges();
});
it('should create', () => {
  expect(component).toBeTruthy();
});
```

Scope of Improvement

Contact Directory/Information application has a lot of scope for improvement adding material design and material data tables can enhance the usability. Using Toastr messages and other notifications can also improve the usability.

We can connect to actual server api and fetch, add and update the database records.

Thank You

Thank you for reviewing the application. Please provide your feedback!