Angular 2+

Workshop. HttpClient.

Contents

Task 01. Import Modules

1. Make changes to **AppModule**. Use the following snippet of code:

```
// 1ng
import { HttpClientModule } from '@angular/common/http';

// 2
imports: [
    …
    HttpClientModule,
    AppRoutingModule
]
```

Task 02. Simulating Web API

1. Run the following command from command line:

```
>npm install -g json-server
```

2. Create file **db\db.json (in project folder).** Use the following snippet of code:

```
{
    "tasks": [
        { "id": 1, "action": "Estimate", "priority": 1, "estHours": 8},
        { "id": 2, "action": "Create", "priority": 2, "estHours": 8},
        { "id": 3, "action": "Edit", "priority": 3, "estHours": 4},
        { "id": 4, "action": "Delete", "priority": 3, "estHours": 2},
        { "id": 5, "action": "Build", "priority": 1, "estHours": 4},
        { "id": 6, "action": "Deploy", "priority": 2, "estHours": 8}
    ],
  "users": [
        { "id": 1, "firstName": "Anna", "lastName": "Borisova" },
        { "id": 2, "firstName": "Boris", "lastName": "Vlasov"},
        { "id": 3, "firstName": "Gennadiy", "lastName": "Dmitriev"}
  ]
}
```

3. Run the following command from command line of db folder:

```
>json-server --watch db.json
```

Task 03. Task Promise Service

1. Create **TaskPromiseService**. Use the following snippet of code:

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

// rxjs
import { toPromise } from 'rxjs/operator/toPromise';

import { Task } from './../models/task.model';

@Injectable()
export class TaskPromiseService {
  private tasksUrl = 'http://localhost:3000/tasks';

  constructor(
    private http: HttpClient
  ) {}

  getTasks(): Promise<Task[]> {
    return this.http.get(this.tasksUrl)
            .toPromise()
            .then( response => <Task[]>response)
            .catch(this.handleError);
  }

  private handleError(error: any): Promise<any> {
    console.error('An error occurred', error);
    return Promise.reject(error.message || error);
  }
}
```

2. Create file tasks/services/index.ts. Use the following snippet of code:

```
export * from './task-array.service';
export * from './task-promise.service';
```

3. Make changes to the file **tasks/index.ts.** Use the following snippet of code:

```
export * from './services/task-array.service';
```

4. Make changes to **TasksModule**. Use the following snippet of code:

```
// 1
import {
  TaskListComponent,
  TaskComponent,
  TaskFormComponent,
  TaskArrayService,
  TaskPromiseService
} from '.';

// 2
providers: [
    TaskArrayService,
    TaskPromiseService
]
```

5. Make changes to **TaskListComponent**. Use the following snippet of code:

```
// 1
import { TaskArrayService, TaskPromiseService } from './../services/task-
promise.service';

// 2
constructor(
    …
    private taskPromiseService: TaskPromiseService) { }

// 3
private async getTasks() {
    this.tasks = await this.taskPromiseService.getTasks();
    this.tasks = await this.taskArrayService.getTasks();
  }
```

Task 04. GetTask

1. Make changes to **TaskPromiseService.** Use the following snippet of code:

```
getTask(id: number): Promise<Task> {
    return this.http.get(`${this.tasksUrl}/${id}`)
            .toPromise()
            .then( response => <Task>response )
            .catch( this.handleError );
  }
```

2. Make changes to **TaskFormComponent.** Use the following snippet of code:

```
// 1
import { TaskArrayService, TaskPromiseService } from './../services/task-array.service';

// 2
constructor(
    …
    private taskPromiseService: TaskPromiseService
  ) { }

// 3
this.route.paramMap
      .pipe(
        switchMap((params: Params) =>
                                this.taskArrayService.getTask(+params.get('id'))))
                                this.taskPromiseService.getTask(+params.get('id'))))
      .subscribe(
        task => this.task = {...task},
        err => console.log(err)
);
```

Task 05. UpdateTask

1. Make changes to **TaskPromiseService.** Use the following snippet of code:

```
// 1
import { HttpClient, HttpHeaders } from '@angular/http';

// 2
updateTask(task: Task): Promise<Task> {
    const url = `${this.tasksUrl}/${task.id}`,
      body = JSON.stringify(task),
      options = {
        headers: new HttpHeaders({ 'Content-Type': 'application/json' }),
      };

    return this.http.put(url, body, options)
            .toPromise()
            .then( response => <Task>response )
            .catch( this.handleError );
  }
```

2. Make changes to method **saveTask** of **TaskFormComponent.** Use the following snippet of code:

```
if (task.id) {
      this.taskArrayService.updateTask(task);
      this.taskPromiseService.updateTask(task)
       .then( () => this.goBack() );
}
else {
      this.taskArrayService.addTask(task);
      this.goBack();
}

this.goBack();
```

3. Make changes to **TaskListComponent.** Use the following snippet of code:

```
// 1
import { TaskArrayService, TaskPromiseService } from './../services';

// 2
  constructor(
    private router: Router,
    private taskArrayService: TaskArrayService,
    private taskPromiseService: TaskPromiseService
  ) { }

// 3
completeTask(task: Task): void {
    task.done = true;
    this.taskArrayPromiseService.updateTask(task);
}
```

Task 06. CreateTask

1. Make changes to **TaskListComponent template**. Use the following snippet of HTML:

```
<div>
      <button class="btn btn-primary"
            (click)="createTask()">New Task</button>
      <br><br>
      <app-task
            *ngFor="let task of tasks"
            [task]="task"
            (complete)="completeTask($event)"
            (edit)="editTask($event)">
      </app-task>
</div>
```

2. Make changes to **TaskListComponent.** Use the following snippet of code:

```
createTask() {
    const link = ['/add'];
    this.router.navigate(link);
  }
```

3. Make changes to **TasksRoutingModule**. Use the following snippet of code:

```
const routes: Routes = [
  …
  {
    path: 'add',
    component: TaskFormComponent
  },
  {
    path: 'edit/:id',
    component: TaskFormComponent
  }
];
```

4. Make changes to **TaskPromiseService.** Use the following snippet of code:

```
createTask(task: Task): Promise<Task> {
    const url = this.tasksUrl,
      body = JSON.stringify(task),
      options = {
        headers: new HttpHeaders({ 'Content-Type': 'application/json' }),
      };

    return this.http.post(url, body, options)
            .toPromise()
            .then( response => <Task>response )
            .catch( this.handleError );
  }
```

5. Make changes to method **ngOnInit** of **TaskFormComponent.** Use the following snippet of code:

```
switchMap((params: Params) => this.taskPromiseService.getTask(+params.get('id')))
switchMap((params: Params) => {
        return params.get('id')
```

```
          ? this.taskPromiseService.getTask(+params.get('id'))
          : Promise.resolve(null);
})
```

6. Make changes to method **saveTask** of **TaskFormComponent.** Use the following snippet of code:

```
if (task.id) {
      this.taskPromiseService.updateTask(task)
            .then( () => this.goBack() );
    }
    else {
      this.taskArrayService.addTask(task);
      this.goBack();
    }
const method = task.id ? 'updateTask' : 'createTask';
    this.taskPromiseService[method](task)
      .then( () => this.goBack() );
```

7. Make changes to **TaskFormComponents**. Use the following snippet of code:

```
import { TaskArrayService, TaskPromiseService  } from './../services';
```

Task 07. DeleteTask

1. Make changes to **TaskComponent template**. Use the following snippet of HTML:

```
<div class="panel panel-default">
      <div class="panel-heading">Task</div>
      <div class="panel-body">
            <ul>
                  <li>Action: {{task.action}}</li>
                  <li>Priority: {{task.priority}}</li>
                  <li>Estimate Hours: {{task.estHours}}</li>
                  <li>Actual Hours: {{task.actHours}}</li>
                  <li>Done: {{task.done}}</li>
            </ul>
            <button class="btn btn-primary btn-sm"
                  (click)="completeTask($event)">
                  Done
            </button>
            <button class="btn btn-warning btn-sm"
                  (click)="editTask()">
                  Edit
            </button>
            <button class="btn btn-danger btn-sm"
                  (click)="deleteTask()">
                  Delete
            </button>
      </div>
</div>
```

2. Make changes to **TaskComponent.** Use the following snippet of code:

```
// 1
@Output() delete = new EventEmitter<Task>();

// 2
deleteTask() {
    this.delete.emit(this.task);
}
```

3. Make changes to **TaskListComponent template.** Use the following snippet of code:

```
<app-task
    *ngFor="let task of tasks"
    [task]="task"
    (complete)="completeTask($event)"
    (edit)="editTask($event)"
    (delete)="deleteTask($event)">
</app-task>
```

4. Make changes to **TaskPromiseService.** Use the following snippet of code:

```
deleteTask(task: Task): Promise<Task> {
    const url = `${this.tasksUrl}/${task.id}`;

    return this.http.delete(url)
            .toPromise()
            .then( response => <Task>response)
```

```
            .catch( this.handleError );
}
```

5. Make changes to **TaskListComponent.** Use the following snippet of code:

```
deleteTask(task: Task) {
    this.taskPromiseService.deleteTask(task)
      .then(() => this.tasks = this.tasks.filter(t => t !== task))
      .catch(err => console.log(err));
  }
```

Task 08. User Observable Service

1. Create file **users/users.config.ts.** Use the following snippet of code:

```
import { InjectionToken } from '@angular/core';

const usersBaseUrl = 'http://localhost:3000/users';
export const UsersAPI = new InjectionToken<string>('UsersAPI');

export const UsersAPIProvider = {
    provide: UsersAPI,
    useValue: usersBaseUrl
};
```

2. Create **UserObservableService.** Use the following snippet of code:

```
import { Injectable, Inject } from '@angular/core';
import { HttpClient, HttpHeaders, HttpResponse, HttpErrorResponse } from
'@angular/common/http';

import { Observable } from 'rxjs/Observable';
import { _throw } from 'rxjs/observable/throw';
import { map, catchError } from 'rxjs/operators';

import { User } from './../models/user.model';
import { UsersAPI } from '../users.config';

@Injectable()
export class UserObservableService {

  constructor(
    private http: HttpClient,
    @Inject(UsersAPI) private usersUrl: string
  ) {}

  getUsers(): Observable<User[]> {
    return this.http.get(this.usersUrl)
        .pipe(
          map( this.handleData ),
          catchError( this.handleError )
        );
  }

  getUser(id: number) {

  }

  updateUser(user: User) {

  }

  createUser(user: User) {

  }

  deleteUser(user: User) {

  }
```

```
  private handleData(response: HttpResponse<User>) {
    const body = response;
    return body || {};
  }

  private handleError(err: HttpErrorResponse) {
    let errorMessage: string;

    // A client-side or network error occurred.
    if (err.error instanceof Error) {
      errorMessage = `An error occurred: ${err.error.message}`;
    }
    // The backend returned an unsuccessful response code.
    // The response body may contain clues as to what went wrong,
    else {
      errorMessage = `Backend returned code ${err.status}, body was: ${err.error}`;
    }

    console.error(errorMessage);
    return _throw(errorMessage);
  }
}
```

3. Create file users/services/index.ts. Use the following snippet od code:

```
export * from './user-array.service';
export * from './user-observable.service';
```

4. Make changes to file **users/index.ts.** Use the following snippet of code:

```
export * from './services/user-array.service';
```

1. Make changes to **UsersModule**. Use the following snippet of code:

```
// 1
import { UsersAPIProvider } from './users.config';
import { UserComponent, UserArrayService, UserResolveGuard, UserObservableService } from
'.';

// 2
providers: [
    …
    UsersAPIProvider,
    UserObservableService
]
```

5. Make changes to **UserListComponent.** Use the following snippet of code:

```
// 1
import { UserArrayService, UserObservableService } from './../services/user-
array.service';

// 2
export class UserListComponent implements OnInit {

// 3
```

```
constructor(
    …
    private userObservableService: UserObservableService
) { }

// 4
ngOnInit() {
  this.users$ = this.userObservableService.getUsers();
  this.users$ = this.userArrayService.getUsers();
…
}
```

Task 09. GetUser

1. Make changes to **UserObservableService.** Use the following snippet of code:

```
getUser(id: number): Observable<User> {
     return this.http.get(`${this.usersUrl}/${id})
        .pipe(
          map( this.handleData ),
          catchError( this.handleError )
        );
}
```

2. Make changes to **UserResolveGuard.** Use the following snippet of code:

```
// 1
import { UserArrayService, UserObservableService } from './../services/user-
array.service';

// 2
constructor(
    private userArrayService: UserArrayService,
    private userObservableService: UserObservableService,
    private router: Router
  ) {}

// 3
resolve(route: ActivatedRouteSnapshot): Observable<User> {
…
       return this.userArrayService.getUser(id)
       return this.userObservableService.getUser(id)
…
}
```

Task 10. UpdateUser and CreateUser

1. Make changes to the method **updateUser** of **UserObservableService.** Use the following snippet of code:

```
updateUser(user: User): Observable<User> {
     const url = `${this.usersUrl}/${user.id}`,
        body = JSON.stringify(user),
        options = {
          headers: new HttpHeaders({ 'Content-Type': 'application/json' })
        };

    return this.http
          .put(url, body, options)
          .pipe(
            map( this.handleData ),
            catchError(this.handleError)
          );
  }
```

2. Make changes to the method **createUser** of **UserObservableService.** Use the following snippet of code:

```
createUser(user: User): Observable<User> {
    const url = this.usersUrl,
        body = JSON.stringify(user),
        options = {
          headers: new HttpHeaders({ 'Content-Type': 'application/json' })
        };
       return this.http
          .post(url, body, options)
          .pipe(
            map( this.handleData ),
            catchError( this.handleError )
          );
  }
```

3. Make changes to **UserFormComponent.** Use the following snippet of code:

```
// 1
import { Component, OnInit, OnDestroy } from '@angular/core';
import { Subscription } from 'rxjs/Subscription';
import { UserArrayService } from './../services/user-array.service';
import { UserObservableService } from './../services';
import { Location } from '@angular/common';

// 2
export class UserFormComponent implements OnInit, OnDestroy, CanComponentDeactivate {

// 3
private sub: Subscription;

// 4
constructor(
    private userArrayService: UserArrayService,
    private userObservableService: UserObservableService,
    private location: Location,
```

```
    …
  ) { }



// 5
ngOnDestroy(): void {
    if (this.sub) {
       this.sub.unsubscribe();
    }
}

// 6
 if (user.id) {
       this.userArrayService.updateUser(user);
       // optional parameter: http://localhost:4200/users;id=2
       this.router.navigate(['users', {editedUserID: user.id}]);
     }
     else {
       this.userArrayService.addUser(user);
       this.goBack();
    }
this.originalUser = {...this.user};

const method = user.id ? 'updateUser' : 'createUser';
    this.sub = this.userObservableService[method](user)
      .subscribe(
        () => {
          this.originalUser = {...this.user};
          user.id
            // optional parameter: http://localhost:4200/users;id=2
            ? this.router.navigate(['users', { editedUserID: user.id }])
            : this.goBack();
        },
        error => console.log(error)
      );

// 7
goBack() {
    this.router.navigate(['./../../'], { relativeTo: this.route });
    this.location.back();
}
```

4. Make changes to **UsersComponent template.** Use the following snippet of HTML:

```
<h2>Users</h2>
<button class="btn btn-primary"
        (click)="createUser()">New User</button>
<br><br>
<router-outlet></router-outlet>
```

5. Make changes to **UsersComponent.** Use the following snippet of code:

```
// 1
import { Router } from '@angular/router';
```

```
// 2
constructor(
    private router: Router
) { }

// 3
createUser() {
    const link = ['/users/add'];
    this.router.navigate(link);
}
```

Task 11. DeleteUser

1. Make changes to **UserComponent template.** Use the following snippet of HTML:

```
<button class="btn btn-warning btn-sm"
    (click)="editUser()">
    Edit
</button>
<button class="btn btn-danger btn-sm"
    (click)="deleteUser()">
    Delete
</button>
```

2. Make changes to **UserComponent.** Use the following snippet of code:

```
// 1
@Output() delete = new EventEmitter<User>();

// 2
deleteUser() {
    this.delete.emit(this.user);
}
```

3. Make changes to **UserListComponent template.** Use the following snippet of HTML:

```
<user
  *ngFor='let user of users'
  [user]="user"
  [class.edited]="isEdited(user)"
  (edit)="editUser($event)"
  (delete)="deleteUser($event)">
</user>
```

4. Make changes to **UserObservableService.** Use the following snippet of code:

```
// 1
import { map, switchMap, catchError } from 'rxjs/operators';

// 2
deleteUser(user: User): Observable<User[]> {
    const url = `${this.usersUrl}/${user.id}`;

    return this.http.delete(url)
      .pipe(
        switchMap(() => this.getUsers())
      );
  }
```

5. Make changes to **UserListComponent.** Use the following snippet of code:

```
deleteUser(user: User) {
    this.users$ = this.userObservableService.deleteUser(user);
  }
```

Task 12. AutoUnsubscribe Decorator

1. Create file **app/core/decorators/auto-unsubscribe.decorator.ts.** Use the following snippet of code:

```
export function AutoUnsubscribe(subName: string = 'sub') {
  return function (constructor) {
    const original = constructor.prototype.ngOnDestroy;

    constructor.prototype.ngOnDestroy = function () {
      const sub = this[subName];

      if (sub) {
        sub.unsubscribe();
      }

      if (original && (typeof original === 'function')) {
        original.apply(this, arguments);
      }

      console.log(`Unsubscribe decorator is called. Subscription name is: ${subName}.`);
    };
  };
}
```

2. Create file **app/core/decorators/index.ts.** Use the following snippet of code:

```
export * from './auto-unsubscribe.decorator';
```

3. Make changes to file **app/core/index.ts**. Use the following snippet of code:

```
export * from './decorators';
```

4. Make changes to **UserFormComponent.** Use the following snippet of code:

```
// 1
import { Component, OnInit, OnDestroy } from '@angular/core';
import { AutoUnsubscribe } from './../../core';

// 2
@Component({
  templateUrl: './user-form.component.html',
  styleUrls: ['./user-form.component.css'],
})
@AutoUnsubscribe()
export class UserFormComponent implements OnInit, OnDestroy, CanComponentDeactivate {

// 3
ngOnDestroy(): void {
  if (this.sub) {
    this.sub.unsubscribe();
  }
}
```

Task 13. Request Configuration

1. Make changes to **UserObservableService.** Use the following snippet of code:

```
// Case 1 Handle Body {observe: 'body'}
// getUser(id: number): Observable<User> {
//    return this.http.get(`${this.usersUrl}/${id}`, {observe: 'body'})
//       .pipe(
//         map(this.handleData1),
//         catchError(this.handleError)
//       );
//  }

// private handleData1(response: User) {
//   console.log(response);
//   const body = response;
//   return body || {};
// }
// End Case 1

// Case 2: Handle Response { observe: 'response' }
// getUser(id: number): Observable<User> {
//   return this.http.get<User>(`${this.usersUrl}/${id}`, {observe: 'response'})
//     .pipe(
//         map(this.handleData2),
//         catchError(this.handleError)
//     );
// }

// private handleData2(response: HttpResponse<User>) {
//   console.log(response);
//   const body = response.body;
//   return body || {};
// }
// End Case 2

// Case 3: Specify HttpResponse Type get<T>
// getUser(id: number): Observable<User> {
//   return this.http.get<User>(`${this.usersUrl}/${id}`)
//     .pipe(
//       map(this.handleData3),
//       catchError(this.handleError)
//     );
// }

// private handleData3(response: User) {
//   console.log(response);
//   const body = response;
//   return body || {};
// }
// End Case 3

// Case 4: responseType: text
// getUser(id: number): Observable<User> {
//   return this.http.get(`${this.usersUrl}/${id}`, {responseType: 'text'})
//     .pipe(
//       map(this.handleData4),
//       catchError(this.handleError)
```

```
//     );
// }

// private handleData4(response: string) {
//   console.log(response);
//   const body = JSON.parse(response);
//   return body || {};
// }
// End Case 4
```

2. Comment the method **getUser()** and uncomment snippet of code for first case, then second, … Look to the console.

Task 14. Interceptors

1. Create file **app/core/interceptors/my.interceptor.ts**. Use the following snippet of code:

```
import {Injectable} from '@angular/core';
import { HttpEvent, HttpInterceptor, HttpHandler, HttpRequest, HttpResponse, HttpParams
} from '@angular/common/http';

import { Observable } from 'rxjs/Observable';

@Injectable()
export class MyInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    // request interceptor
    let clonedRequest;
    if (req.url.includes('users')) {
      clonedRequest = req.clone({
        params: new HttpParams()
          .set('ts_interceptor', Date.now().toString())
      });
      console.log(clonedRequest);
    } else {
      clonedRequest = req;
    }

    return next.handle(clonedRequest);
  }
}
```

2. Make changes to **AppModule.** Use the following snippet of code:

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

import { MyInterceptor } from './core/interceptors/my.interceptor';

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

3. Look at the requests in the browser console. Ensure that only the user requests are processed by My interceptor.
4. Make changes to **MyInterceptor.** Use the following snippet of code:

```
// 1
import { map } from 'rxjs/operators';


// 2
return next.handle(clonedRequest);
return next.handle(clonedRequest)
      .pipe(
        // response interceptor
```

```
    map((event: HttpEvent<any>) => {
      if (event instanceof HttpResponse) {
        // do stuff with response
        console.log('Response Interceptor');
        console.log(event);
        console.log(event.body);
        return event;
      }
    })
  );
```

5. Look in the console on the result of applying My interceptor.