-

Contents

Task 01. Import Modules

1. Добавьте в файле **app.module.ts** следующий фрагмент кода:

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

imports: [
    BrowserModule,
    CommonModule,
    FormsModule,
    TasksModule,
    HttpClientModule,
    AppRoutingModule
]
```

Task 02. Simulating Web API

1. npm install -g json-server
2. create file \db\db.json следующего содержания (в папке проекта)

```
{
    "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 json-server from cmd: json-server --watch db.json

Task 03. Task Promise Service

1. Внесите изменения в файл **app/services/rx-js-extensions.ts** используя следующий фрагмент кода

```
import 'rxjs/add/operator/toPromise';
```

2. Создайте файл **tasks/services/task-promise.service.ts** используя следующий фрагмент кода

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import './../../services/rxjs-extensions';

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

@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);
  }
}
```

3. Внесите изменения в файл **tasks/index.ts**

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

4. Внесите изменения в файл **tasks/tasks.module.ts**

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

providers: [
    TaskArrayService,
    TaskPromiseService
]
```

5. Внесите изменения в компонент **TaskListComponent**

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

constructor(
    private taskArrayService: TaskArrayService,
    private taskPromiseService: TaskPromiseService) { }

ngOnInit() {
    this.taskPromiseService.TaskArrayService.getTasks()
      .then(tasks => this.tasks = tasks)
      .catch((err) => console.log(err));
}
```

Task 04. GetTask

1. Добавьте метод **getTask** в сервис **TaskPromiseService** используя следующий фрагмент кода

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

2. Внесите изменения в компонент **TaskFormComponent**

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


constructor(
    private taskArrayService: TaskArrayService,
    private taskPromiseService: TaskPromiseService,
    private router: Router,
    private route: ActivatedRoute
  ) { }
```

3. Внесите изменения в метод **ngOnInit** компонента **TaskFormComponent**

```
this.route.params
      .switchMap((params: Params) => this.tasksService.getTask(+params.get('id')))
      .switchMap((params: Params) => this.taskPromiseService.getTask(+params.get('id')))
      .subscribe(
        task => this.task = Object.assign({}, task),
        err => console.log(err)
);
```

Task 05. UpdateTask

1. Внесите изменения в **TaskPromiseService**

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

2. Добавьте метод **updateTask** в сервис **TaskPromiseService** используя следующий фрагмент кода

```
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 );
  }
```

3. Внесите изменения в метод **saveTask** компонента **TaskFormComponent**

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

    this.goBack();
```

4. Внесите изменения в метод **completeTask** компонента **TaskListComponent**

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

Task 06. CreateTask

1. Внесите изменения в темплейт компонента **TaskListComponent** использу следующий фрагмент разметки

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

2. Внесите изменения в компонент **TaskListComponent**

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

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

3. Добавьте метод **createTask** в компонент **TaskListComponent** используя следующий фрагмент кода

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

4. Внесите изменения в файл **tasks/tasks.routing.module.ts**

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

5. Добавьте метод **createTask** в сервис **TaskPromiseService** используя следующий фрагмент кода

```
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 );
  }
```

6. Внесите изменения в **ngOnInit** компонента **TaskFormComponent** используя следующий фрагмент кода

```
.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);
})
```

7. Внесите изменения в метод **saveTask** компонента **TaskFormComponent** используя следующий фрагмент кода

```
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() );
```

Task 07. DeleteTask

1. Внесите изменения в темплейт компонента **TaskComponent** используя следующий фрагмент разметки

```
<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. Внесите изменения в компонент **TaskComponent** используя следующий фрагмент кода:

```
@Output() onComplete = new EventEmitter<Task>();
@Output() onDelete = new EventEmitter<Task>();

deleteTask() {
    this.onDelete.emit(this.task);
}
```

3. Внесите изменения в темплейт компонента **TaskListComponent** используя следующий фрагмент разметки

```
<task
	*ngFor='let task of tasks'
	[task]="task"
	(onComplete)="completeTask($event)"
	(onDelete)="deleteTask($event)">
</task>
```

4. Добавьте метод **deleteTask** в сервис **TaskPromiseService** используя следующий фрагмент разметки

```
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. Добавьте метод **deleteTask** в компонент **TaskListComponent** используя следующий фрагмент разметки

```
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. Создайте сервис **UserObservableService** в файле **users/services/user-observable.service.ts** используя следующий фрагмент кода

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

import { Observable } from 'rxjs/Observable';
import './../../services/rxjs-extensions';

import { User } from './../../models/user';

@Injectable()
export class UserObservableService {
  private usersUrl = 'http://localhost:3000/users';

  constructor(
    private http: HttpClient
  ) {}

  getUsers() {
    return this.http.get(this.usersUrl)
            .map( this.handleData )
            .catch( 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 Observable.throw(errorMessage);
  }
}
```

2. Внесите изменения в файл **users/index.ts**

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

3. Внесите изменения в файл **users/users.module.ts**

```
import { UserComponent, UserArrayService, UserObservableService } from '.';

providers: [
    UserArrayService,
    UserObservableService
  ]
```

4. Внесите изменения в **UserListComponent**

```
import { Subscription } from 'rxjs/Subscription';

import { User } from './../../models/user';
import { UserObservableService } from './../services/user-observable.service';

export class UserListComponent implements OnInit, OnDestroy {

users: Array<User>;
errorMessage: string;
private subscription: Subscription[] = [];

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

ngOnInit() {
    this.userArrayService.getUsers()
      .then(users => this.users = users)
      .catch((err) => console.log(err));

    const sub = this.userObservableService.getUsers()
      .subscribe(
        users => this.users = users,
        error => this.errorMessage = <any>error
      );
    this.subscription.push(sub);
  }
```

```
ngOnDestroy() {
    this.subscription.forEach(sub => sub.unsubscribe());
}
```

Task 09. GetUser

1. Внесите изменения в метод **getUser** сервиса **UserObservableService** используя следующий фрагмент кода

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

2. Внесите изменения в **guards/user-resolve-guard.ts** используя следующий фрагмент кода

```
// 1
import { Observable } from 'rxjs/Observable';

import { UserArrayService } from './../users/services/user-array.service';
import { UserObservableService } from './../users';

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

// 3
resolve(route: ActivatedRouteSnapshot): PromiseObservable<User> {
  return this.userArrayService.getUser(id).then(user => {
       if (user) {
         return user;
       }
       else { // id not found
         this.router.navigate(['/users']);
         return null;
       }
  });

    return this.userObservableService.getUser(id)
      .catch(() => {
        this.router.navigate(['/users']);
        return Observable.of(null);
      });
}
```

Task 10. UpdateUser and CreateUser

1. Внесите изменения в метод **updateUser** сервиса **UserObservableService** используя следующий фрагмент кода

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

2. Внесите изменения в метод **createUser** сервиса **UserObservableService** используя следующий фрагмент кода

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

3. Внести изменения в компонент **UserFormComponent** используя следующий фрагмент кода

```
// 1
import { Subscription } from 'rxjs/Subscription';

// 2
import { UserArrayService } from './../services/user-array.service';
import { UserObservableService } from './../services/user-observable.service';

// 3
private sub: Subscription[] = [];

// 4
constructor(
    private userArrayService: UserArrayService,
    private userObservableService: UserObservableService,
    private route: ActivatedRoute,
    private router: Router,
    public dialogService: DialogService
  ) { }



// 5
ngOnDestroy(): void {
    this.sub.forEach(sub => sub.unsubscribe());
```

```
}

// 6
 if (user.id) {
       this.userArrayService.updateUser(user);
       this.originalUser = Object.assign({}, this.user);
       // optional parameter: http://localhost:4200/users;id=2
       this.router.navigate(['users', { id: user.id }]);
     }
     else {
       this.userArrayService.addUser(user);
       this.originalUser = Object.assign({}, this.user);
       this.router.navigate(['users']);
    }
const method = user.id ? 'updateUser' : 'createUser';
    const sub = this.userObservableService[method](user)
      .subscribe(
        () => {
          this.originalUser = Object.assign({}, this.user);
          user.id
            // optional parameter: http://localhost:4200/users;id=2
            ? this.router.navigate(['users', { id: user.id }])
            : this.router.navigate(['users']);
        },
        error => console.log(error)
      );
    this.sub.push(sub);
```

4. Внесите изменения в темплейт компонента **UsersComponent** использу следующий фрагмент разметки

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

5. Внесите изменения в компонент **UsersComponent**

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


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

6. Добавьте метод **createUser** в компонент **UsersComponent** используя следующий фрагмент кода

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

7. Внесите изменения в **guards/user-resolve.guard.ts** используя следующий фрагмент кода

```
return this.userObservableService.getUser(id)
      .catch(() => {
        this.router.navigate(['/users']);
        return Observable.of(null);
      });
if (id) {
      return this.userObservableService.getUser(id)
        .catch(() => {
          this.router.navigate(['/users']);
          return Observable.of(null);
        });
    }
    else {
      return Observable.of(new User(null, '', ''));
    }
```

Task 11. DeleteUser

1. Внесите изменения в темплейт компонента **UserComponent** используя следующий фрагмент разметки

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

2. Внесите изменения в компонент **UserComponent** используя следующий фрагмент кода:

```
// 1
import { Component, Input, Output , EventEmitter } from '@angular/core';

// 2
@Output() onDelete = new EventEmitter<User>();

deleteUser() {
    this.onDelete.emit(this.user);
}
```

3. Внесите изменения в темплейт компонента **UserListComponent** используя следующий фрагмент разметки

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

4. Внесите изменения в метод **deleteUser** сервиса **UserObservableService** используя следующий фрагмент кода

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

    return this.http.delete(url)
      .map( this.handleData )
      .catch(this.handleError);
  }
```

5. Добавьте метод **deleteUser** в компонент **UserListComponent** используя следующий фрагмент разметки

```
deleteUser(user: User) {
    this.userObservableService.deleteUser(user)
       .subscribe(
          () => this.users = this.users.filter(u => u !== user),
          err => console.log(err)
        );
  }
```

Task 12. AutoUnsubscribe Decorator

1. Создайте файл **app/decorators/auto-unsubscribe.decorator.ts** используя следующий фрагмент кода

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

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

      if (sub && isArray) {
        sub.forEach(s => s.unsubscribe());
      }
      else if (sub && !isArray) {
        sub.unsubscribe();
      }

      original
        && typeof original === 'function'
        && original.apply(this, arguments);
      console.log(`Unsibscribe decorator is called. Subscription name is: ${subName}.
Subscription is array: ${isArray}`);
    };
  }
}
```

2. Создайте файл **decorators/index.ts** используя следующий фрагмент кода

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

3. Внесите изменения в компонент **UserFormComponent** используя следующий фрагмент кода

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

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

// 3
ngOnDestroy(): void {
    this.sub.forEach(sub => sub.unsubscribe());
}
```

4. Внесите изменения в компонент **UserListComponent** используя следующий фрагмент кода

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

```
// 2
@Component({
  templateUrl: 'user-list.component.html',
  styleUrls: ['user-list.component.css']
})
@AutoUnsubscribe('subscriptions')
export class UserListComponent implements OnInit, OnDestroy {

// 3
ngOnDestroy() {
    this.subscriptions.forEach(sub => sub.unsubscribe());
}
```

Task 13. Request Configuration

1. Внесите изменения в сервис **UserObservableService** используя следующий фрагмент кода

```
// Case 1 Handle Body {observe: 'body'}
// getUser(id: number): Observable<User> {
//   return this.http.get(`${this.usersUrl}/${id}`, {observe: 'body'})
//     .map(this.handleData1)
//     .catch(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'})
//     .map(this.handleData2)
//     .catch(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}`)
//     .map(this.handleData3)
//     .catch(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'})
//     .map(this.handleData4)
//     .catch(this.handleError);
// }

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

```
// End Case 4
```

2. Закоментируйте метод **getUser()** и по очереди расскоментируйте фрагменты кода, который Вы добавили. Посмотрите на ответ сервера в консоли.

Task 14. Interceptors

1. Создайте сервис **MyInterceptor** в файле **services/interceptors.service.ts** используя следующий фрагмент кода

```
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. Внесите изменения в модуль **app.module.ts** используя следующий фрагмент кода

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

import { MyInterceptor } from './services/interceptors.service';

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

3. Посмотрите на запросы в консоли браузера. Убедитесь, что только для запросов пользователя работает интерсептор.
4. Внесите изменения в сервис **MyInterceptor** используя следующий фрагмент кода

```
return next.handle(clonedRequest);
return next.handle(clonedRequest)
      // 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. Посмотрите в консоли на результат применения интерсептора.