Angular 2+

Workshop. HttpClient.

Contents

Task 01. Import Modules

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

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

// 2
imports: [
    …
    // import HttpClientModule after BrowserModule
    HttpClientModule,
    AppRoutingModule
]
```

Task 02. Simulating Web API

1. Run the following command from command line:

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

2. Create file **db\db.json (in project root 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": "Clara", "lastName": "Dmitrieva"},
        { "id": 4, "firstName": "Dariya", "lastName": "Egorova"},
        { "id": 5, "firstName": "Fatima", "lastName": "Georg"},
        { "id": 6, "firstName": "Hunna", "lastName": "Jackson"}
  ]
}
```

3. Make changes to **package.json** file.

```
Windows:
"start": "concurrently --kill-others \"ng serve -o\" \"json-server --watch
db\\db.json\"",
Mac
"start": "concurrently --kill-others \"ng serve -o\" \"json-server --watch
db\/db.json\""
"start": "ng serve",
```

4. Run project:

```
>npm start
```

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';

import { firstValueFrom } from 'rxjs';

import type { TaskModel } from './../models/task.model';

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

  constructor(private http: HttpClient) {}

  getTasks(): Promise<TaskModel[]> {
    const request$ = this.http.get(this.tasksUrl);
    return firstValueFrom(request$)
      .then(response => response as TaskModel[])
      .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';
```

4. 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
ngOnInit(): void {
    this.tasks = this.taskArrayService.getTasks();
    this.tasks = this.taskPromiseService.getTasks();
  }
```

Task 04. GetTask

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

```
getTask(id: NonNullable<TaskModel['id']> | string): Promise<TaskModel> {
    const url = `${this.tasksUrl}/${id}`;

    const request$ = this.http.get(url);
    return firstValueFrom(request$)
      .then(response => response as TaskModel)
      .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: ParamMap) =>
           // notes about "!"
           // params.get() returns string | null, but getTask takes string | number
           // in this case taskID is a path param and can not be null
             this.taskArrayServicetaskPromiseService.getTask(params.get('taskID')!)
        ),
        // transform undefined => {}
        map(el => el ? el : {} as TaskModel)
      .subscribe(observer);
```

Task 05. UpdateTask

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

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

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

    return firstValueFrom(request$)
      .then(response => response as TaskModel)
      .catch(this.handleError);
  }
```

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

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

this.onGoBack();
```

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
onCompleteTask(task: Task): void {
    const updatedTask = { ...task, done: true };
    this.taskArrayService.updateTask(updatedTask);
    this.updateTask(task).catch(err => console.log(err));
}

// 4
private async updateTask(task: TaskModel) {
    const updatedTask = await this.taskPromiseService.updateTask({
      ...task,
```

```
      done: true
    });

    const tasks: TaskModel[] = await this.tasks;
    const index = tasks.findIndex(t => t.id === updatedTask.id);
    tasks[index] = { ...updatedTask };
}
```

Task 06. CreateTask

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

```
<div>
	<button class="btn btn-primary"
		(click)="onCreateTask()">New Task</button>
	<br><br>
	<app-task
		*ngFor="let task of tasks | async"
		[task]="task"
		(completeTask)="onCompleteTask($event)"
		(editTask)="onEditTask($event)">
	</app-task>
</div>
```

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

```
// 1
onCreateTask(): void {
    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/:taskID',
    component: TaskFormComponent
  }
];
```

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

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

    const request$ = this.http.post(url, task, options);

    return firstValueFrom(request$)
      .then(response => response as TaskModel)
      .catch(this.handleError);
  }
```

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

```
switchMap((params: ParamMap) =>
            // notes about "!"
```

```
            // params.get() returns string | null, but getTask takes string | number
            // in this case taskID is a path param and can not be null
            this.taskPromiseService.getTask(params.get('taskID')!);
        ),
switchMap((params: ParamMap) => {
            // notes about "!"
            // params.get() returns string | null, but getTask takes string | number
            // in this case taskID is NOT a path param and can not be null
            if (params.has('taskID')) {
               return this.taskPromiseService.getTask(params.get('taskID')!);
            } else {
               return Promise.resolve(undefined);
            }
        }),
```

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

```
if (task.id) {
      this.taskPromiseService.updateTask(task)
            .then( () => this.onGoBack() );
    }
    else {
      this.taskArrayService.createTask(task);
      this.onGoBack();
    }
const method = task.id ? 'updateTask' : 'createTask';
    this.taskPromiseService[method](task)
      .then(() => this.onGoBack())
      .catch(err => console.log(err));
```

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

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

// 2
constructor(
    private taskArrayService: TaskArrayService,
    …
  ) { }
```

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)="onCompleteTask()"
                  [disabled]="task.done">
                  Done
            </button>
            <button class="btn btn-warning btn-sm"
                  (click)="onEditTask()">
                  Edit
            </button>
            <button class="btn btn-danger btn-sm"
                  (click)="onDeleteTask()">
                  Delete
            </button>
      </div>
</div>
```

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

```
// 1
@Output() deleteTask = new EventEmitter<TaskModel>();

// 2
onDeleteTask(): void {
    this.deleteTask.emit(this.task);
}
```

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

```
<app-task
    *ngFor="let task of tasks | async"
    [task]="task"
    (completeTask)="onCompleteTask($event)"
    (editTask)="onEditTask($event)"
    (deleteTask)="onDeleteTask($event)">
</app-task>
```

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

```
deleteTask(task: TaskModel): Promise<unknown> {
    const url = `${this.tasksUrl}/${task.id}`;
    const request$ = this.http.delete(url);

    return firstValueFrom(request$)
```

```
        // json-server return empty object
        // so we don't use .then(...)
        .catch(this.handleError);
}
```

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

```
onDeleteTask(task: TaskModel): void {
    this.taskPromiseService
      .deleteTask(task)
      .then(() => (this.tasks = this.taskPromiseService.getTasks()))
      .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';

export const UsersAPI = new InjectionToken<string>('UsersAPI', {
  providedIn: 'any',
  factory: () => 'http://localhost:3000/users'
});
```

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

```
import { Injectable, Inject } from '@angular/core';
import { HttpClient, HttpHeaders, type HttpResponse, type HttpErrorResponse } from
'@angular/common/http';
import { type Observable, throwError, catchError, retry, share } from 'rxjs';

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

@Injectable({
  providedIn: 'any'
})
export class UserObservableService {
  constructor(
    private http: HttpClient,
    @Inject(UsersAPI) private usersUrl: string
  ) {}

  getUsers(): Observable<UserModel[]> {
    return this.http.get<UserModel[]>(this.usersUrl).pipe(
      retry(3),
      share(),
      catchError(this.handleError)
    );
  }

  getUser(id: NonNullable<UserModel['id']> | string) {}

  updateUser(user: UserModel) {}

  createUser(user: UserModel) {}

  deleteUser(user: UserModel) {}

 private handleError(error: HttpErrorResponse) {
  if (error.status === 0) {
    // A client-side or network error occurred. Handle it accordingly.
    console.error('An error occurred:', error.error);
  } else {
    // The backend returned an unsuccessful response code.
    // The response body may contain clues as to what went wrong.
    console.error(
      `Backend returned code ${error.status}, body was: `, error.error);
  }
  // Return an observable with a user-facing error message.
  return throwError(() => new Error('Something bad happened; please try again later.'));
```

```
}
}
```

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

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

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

```
// 1
import { UserArrayService, UserObservableService } from './../../services/user-
array.service';
import { EMPTY, Observable, catchError, switchMap } from 'rxjs';

// 2
constructor(
    …
    private userObservableService: UserObservableService
) { }

// 3
ngOnInit(): void {
  this.users$ = this.userObservableService.getUsers();
  this.users$ = this.userArrayService.users$
      .pipe(
        catchError(err => {
          console.log(err);
          return EMPTY;
        })
      );
…
}
```

Task 09. GetUser

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

```
getUser(id: NonNullable<UserModel['id']> | string): Observable<UserModel> {
    const url = `${this.usersUrl}/${id}`;

    return this.http.get<UserModel>(url)
        .pipe(
            retry(3),
            share(),
            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,
    …
) {}

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

3. Make changes to method **ngOnInit** of **UserListComponent.** Use the following snippet of code:

```
// 1
import { EMPTY, Observable, switchMap } from 'rxjs';
import { UserArrayService, UserObservableService } from './../../services';

// 2
constructor(
…
private userArrayService: UserArrayService,
) { … }

// 2 ngOnInit
switchMap((params: ParamMap) =>
this.userArrayService.getUser(params.get('editedUserID')!))
switchMap((params: ParamMap) => {
    return params.has('editedUserID')
      ? this.userObservableService.getUser(params.get('editedUserID')!)
      : EMPTY;
  })
```

Task 10. UpdateUser and CreateUser

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

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

    return this.http
          .put<UserModel>(url, user, options)
          .pipe( catchError(this.handleError) );
  }
```

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

```
createUser(user: UserModel): Observable<UserModel> {
    const url = this.usersUrl;
    const options = {
      headers: new HttpHeaders({ 'Content-Type': 'application/json' })
    };

       return this.http
          .post<UserModel>(url, user, options)
          .pipe(
            catchError( this.handleError )
          );
  }
```

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

```
// 1
import { Component, type OnInit, type OnDestroy } from '@angular/core';
import { type Observable, type Subscription, map } from 'rxjs;
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 {
```

```
      this.sub?.unsubscribe();
}

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

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

// 7
onGoBack(): void {
    this.onGoBackClick = true;
    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)="onCreateUser()">New User</button>
<br><br>
<router-outlet></router-outlet>
```

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

```
// 1
import { Component, type OnInit } from '@angular/core';
import { Router } from '@angular/router';


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

// 3
```

```
onCreateUser(): void {
    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)="onEditUser()">
    Edit
</button>
<button class="btn btn-danger btn-sm"
    (click)="onDeleteUser()">
    Delete
</button>
```

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

```
// 1
@Output() deleteUser = new EventEmitter<UserModel>();

// 2
onDeleteUser(): void {
    this.deleteUser.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)"
  (editUser)="onEditUser($event)"
  (deleteUser)="onDeleteUser($event)">
</user>
```

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

```
// 1
import { Observable, throwError, catchError, retry, share, concatMap } from 'rxjs';

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

    return this.http.delete(url).pipe(
      concatMap(() => this.getUsers()),
      catchError(this.handleError)
    );
  }
```

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

```
onDeleteUser(user: UserModel): void {
    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 (target: Function | object, propName?: string) => {
    let constructor;

    if (typeof target === 'function') {
      constructor = target;
      console.log(`AutoUnsubscribe class decorator is called. Subscription name is:
${propName ?? subName}.`);
    } else {
      constructor = target.constructor;
      console.log(`AutoUnsubscribe property decorator is called. Subscription name is:
${propName ?? subName}.`);
    }

    const original = constructor.prototype.ngOnDestroy;

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

      sub?.unsubscribe();

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

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, type OnInit, type OnDestroy } from '@angular/core';
import { AutoUnsubscribe, DialogService, type CanComponentDeactivate } 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 {
    this.sub?.unsubscribe();
```

Commented [VZ1]: Не работает в 9 версии в режиме AoT + Ivy – запрещено модифицировать хуки жизненного цикла компонента, хотя создавать другие методы можно.

```
}
```

Task 13. Interceptors

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

```
import {Injectable} from '@angular/core';
import { HttpHeaders, HttpEventType } from '@angular/common/http';
import type {
  HttpEvent,
  HttpInterceptor,
  HttpHandler,
  HttpRequest,
  HttpResponse
} from '@angular/common/http';
import { type Observable } from 'rxjs';

@Injectable()
export class TsInterceptor implements HttpInterceptor {
  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    // request interceptor
    let clonedRequest;
    if (req.method === 'POST' || (req.method === 'PUT')) {
      console.log('req.method:', req.method);
      clonedRequest = req.clone({
        headers: new HttpHeaders({
          'Content-Type': 'application/json',
          'Authorization': 'user-token'
        })
      });
      console.log(clonedRequest);
    } else {
      clonedRequest = req;
    }

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

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

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

import { TsInterceptor } from './ts.interceptor';

export const httpInterceptorProviders = [
  {
    provide: HTTP_INTERCEPTORS,
    useClass: TsInterceptor,
    multi: true
  }
];
```

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

```
import { httpInterceptorProviders } from './core/interceptors';

providers: [ httpInterceptorProviders  ]
```

4. Look at the requests in the browser console. Ensure that only the user requests are processed by TsInterceptor.
5. Make changes to **TaskPromiseService.** Use the following snippet of code:

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

updateTask(task: TaskModel): Promise<TaskModel> {
    const url = `${this.tasksUrl}/${task.id}`;
    const options = {
        headers: new HttpHeaders({ 'Content-Type': 'application/json' })
      };
    const request$ = this.http.put(url, task, options);

    return firstValueFrom(request$)
      .then(response => response as TaskModel)
      .catch(this.handleError);
  }

  createTask(task: TaskModel): Promise<TaskModel> {
    const url = this.tasksUrl;
    const options = {
      headers: new HttpHeaders({ 'Content-Type': 'application/json' })
    };
    const request$ = this.http.post(url, task, options);

    return firstValueFrom(request$)
      .then(response => response as TaskModel)
      .catch(this.handleError);
  }
```

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

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

    return this.http
      .put<UserModel>(url, body, options)
      .pipe(catchError(this.handleError));
  }

  createUser(user: UserModel): Observable<UserModel> {
    const url = this.usersUrl;
    const body = JSON.stringify(user);
    const options = {
      headers: new HttpHeaders({ 'Content-Type': 'application/json' })
    };

    return this.http
      .post<UserModel>(url, body, options)
      .pipe(catchError(this.handleError));
  }
```

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

```
// 1
import { type Observable, filter, map } from 'rxjs';

// 2
return next.handle(clonedRequest);
    // response interceptor
    return next.handle(clonedRequest).pipe(
      filter((event: HttpEvent<any>) => event.type === HttpEventType.Response),
      map((event: HttpEvent<any>) => {
        // do stuff with response
        if ((event as HttpResponse<any>).url!.includes('users')) {
          console.log('Response Interceptor:');
          console.log(event);
          console.log((event as HttpResponse<any>).body);
        }
        return event;
      })
    );
```

8. Look in the console on the result of applying TsInterceptor.
9. Make changes to **UserObservableService.** Use the following snippet of code

```
// 1
import { HttpClient, HttpContextToken, HttpContext } from '@angular/common/http';

// 2
export const interceptorTOKEN = new HttpContextToken(() => 'Some Default Value');

// 3
getUsers(): Observable<UserModel[]> {
    const httpOptions = {
      context: new HttpContext().set(interceptorTOKEN, 'Some Value')
    };

    return this.http.get<UserModel[]>(this.usersUrl, httpOptions).pipe(
      retry(3),
      share(),
      catchError(this.handleError)
    );
}
```

10. Make changes to **TSInterceptor.** Use the following snippet of code

```
// 1
import { interceptorTOKEN } from './../../users';

// 2
// request interceptor
    const contextValue = req.context.get(interceptorTOKEN);
    console.log('contextValue:', contextValue);
```