3. Test on Service

Keys

- 尽量把HttpClient的应用集中在少量Service中，使用HttpClientTestingModule进行测试（类似Case1），其他不涉及HttpClient的地方使用spy的方式更简洁
- 不要直接在component的ts中直接调用HttpClient，让视图层和逻辑层分离，测试更容易写。

Case1. UserService will invoke HttpClient directly

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpParams } from '@angular/common/http';
import { UserCredentials } from '../models/user-credentials';
import { User } from '../models/user';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class UserService {

  readonly loginUrl = 'http://conduit.productionready.io/api/users/login';

  constructor(private http: HttpClient) { }

  login(userCredentials: UserCredentials): Observable<User> {
    return this.http.post<User>(this.loginUrl, userCredentials);
  }
}
```

Test with mock HttpClient

- Remember to import HttpClientTestingModule
- Make use of HttpTestingController to simulate dummy HttpClient, with flush to return dummy result.
- For async function, remember to have 'done: DoneFn'/done().

```
import { TestBed } from '@angular/core/testing';

import { UserService } from './user.service';
import { UserCredentials } from '../models/user-credentials';
import { HttpClientTestingModule, HttpTestingController } from '@angular/common/http/testing';
import { User } from '../models/user';

describe('UserService', () => {
 let service: UserService;
 let testingController: HttpTestingController;

 beforeEach(() => {
   TestBed.configureTestingModule({
     imports: [HttpClientTestingModule],
     providers: [UserService]
   });
   service = TestBed.get(UserService);
   testingController = TestBed.get(HttpTestingController);
});

 afterEach(() => {
   testingController.verify();
 });

 it('Should able to get user when login success', (done: DoneFn) => {
  const userCredentials: UserCredentials = {
    email: 'email',
    password: 'password'
  };
  const expectUser: User = {
    email: 'email',
    username: 'username',
    token: 'token'
  };
  service.login(userCredentials).subscribe(
    user => {
      expect(user).toEqual(expectUser);
      done();
    },
    error => fail('Fail to simulate sucess'),
  );

  const request = testingController.expectOne(service.loginUrl);
  expect(request.request.method).toEqual('POST');
  request.flush(expectUser);
 });

  it('Should able to get error when login fail', () => {
  // given
  const userCredentials: UserCredentials = {
    email: 'email',
    password: 'password'
  };
  const mockErrorResponse = { status: 400, statusText: 'Bad Request' };
  const error = 'Invalid request parameters';

  // when
  service.login(userCredentials).subscribe(
    user => fail('Fail to simulate error'),
    err => {
      expect(err.error).toBe(error);
      expect(err.status).toBe(400);
    }
  );

  // then
  const request = testingController.expectOne(service.loginUrl);
  expect(request.request.method).toEqual('POST');
  request.flush(error, mockErrorResponse);
 });
});
```

Case2. UserService invoke ApiService which encapculated HttpClient inside

api.service.ts

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable, throwError } from 'rxjs';
import { environment } from 'src/environments/environment';
import { catchError } from 'rxjs/operators';

@Injectable({
  providedIn: 'root'
})
export class ApiService {

  constructor(private http: HttpClient) { }

  private handleError(error: any) {
    return throwError(error.error);
  }

  // tslint:disable-next-line: ban-types
  post(path: string, body: Object = {}): Observable<any> {
    return this.http.post(
      `${environment.base_url}${path}`,
      JSON.stringify(body)
    ).pipe(catchError(this.handleError));
  }
}
```

user2.service.ts

```
import { Injectable } from '@angular/core';
import { ApiService } from './api.service';
import { UserCredentials } from '../models/user-credentials';
import { User } from '../models/user';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class User2Service {

  constructor(private apiService: ApiService) { }

  login(userCredentials: UserCredentials): Observable<User> {
    return this.apiService.post('/users/login', {user: userCredentials});
  }
}
```

Test with spy ApiService

- apiClientSpy = jasmine.createSpyObj('ApiService', ['post']);
- apiClientSpy.post.and.returnValue(of(expectUser));

user2.service.spec.ts

```
import { TestBed } from '@angular/core/testing';

import { User2Service } from './user2.service';
import { UserCredentials } from '../models/user-credentials';
import { User } from '../models/user';
import { of } from 'rxjs';

describe('User2Service', () => {
  let service: User2Service;
  let apiClientSpy;

  it ('should able to login', (done: DoneFn) => {
    // given
    const userCredentials: UserCredentials = {
      email: 'email',
      password: 'password'
    };
    const expectUser: User = {
      email: 'email',
      username: 'username',
      token: 'token'
    };
    apiClientSpy = jasmine.createSpyObj('ApiService', ['post']);
    apiClientSpy.post.and.returnValue(of(expectUser));
    service = new User2Service(apiClientSpy);

    // when
    service.login(userCredentials).subscribe (user => {
        expect(user).toEqual(expectUser);
        done();
      },
      fail
    );
  });

});
```