Mutation Test

Mutation tests sunt testele in care se aplica diferiti mutanti(foarte mici modificari la cod, operatori sau modificari la atribuiri, lucru pe care il vom exemplifica mai jos) pentru a se observa cum se comporta codul in urma acestor schimbari.

Pentru aceste mutation tests ne-am folosit de o librarie de javascript numita stryker si am configurat-o sa functioneze pentru jest. Continutul fisierului (stryker.config.json) de configurare este:

|  |
| --- |
| {  "$schema": "./node\_modules/@stryker-mutator/core/schema/stryker-schema.json",  "\_comment": "This config was generated using 'stryker init'. Please take a look at: <https://stryker-mutator.io/docs/stryker-js/configuration/> for more information",  "packageManager": "npm",  "reporters": [  "html",  "clear-text",  "progress",  "dashboard"  ],  "testRunner": "jest",  "coverageAnalysis": "perTest"  } |

Fisierul de testare se genereaza in momentul in care scriem comanda stryker init. Ce face aceasta librarie este ca se uita in fisierele proiectului si aplica diferite modificari mici la cod, pentru a se observa daca mutantul este omorat sau supravietuieste acestei modificari. Un mutant este distrus de un test daca testul se comporta diferit in cele doua programe, cel original si cel alterat.

Prima parte voi pune analiza pe care a facut-o tool-ul folosit pentru a face mutation testing-ul.



Dupa cum prezinta si analiza, a supravietuit un singur mutant si au fost omorati 21.

I) Mutantul viu

|  |
| --- |
| // forma initiala  private userList: User[] = [];  // forma modifica  private userList: User[] = ["Stryker was here"];  // Tipul de date user este  export class User{  public id: number;  public firstName: string;  public lastName: string;  public email: string;  public username: string;  } |

Aceasta variabila este folosita pentru a retine userii ce sunt adaugati la noi in aplicatie ca intr-o forma de memorie cache. Mutantul nu a modificat comportamentul testelor noastre unitare, deoarece eu in pogram cand fac iteratia prin useri, sub forma for(const user of this.userList), verific daca user.email este egal cu o valoare data, cum in cazul nostru user.email ar fi undifiend, si testul s-ar comporta la fel, deoarece nu tine cont programul ca obiectul acesta din array nu este user.

II) Mutantii omorati

Vom analiza o parte din mutantii omorati si vom explica de ce au schimbat valorile testelor.

1)

|  |
| --- |
| // Functie originala  public update(id: number, user: User): User {  if (this.isEmailTaken(user.email)) {  throw new Error('Email is already taken')  }  const indexUser = this.findIndexById(id);  if (indexUser > -1) {  this.userList[indexUser] = user;  return user;  }  return undefined;  } |
| // forma cu mutant  public update(id: number, user: User): User {  if (false)) {  throw new Error('Email is already taken')  }  const indexUser = this.findIndexById(id);  if (indexUser > -1) {  this.userList[indexUser] = user;  return user;  }  return undefined;  } |

Mutantul a fost omorat, deaorece erau momente cand email era luat, adica functia ar fi trebuit sa returneze true, dar nu se intampla asta. Functia de isEmailTaken verifica daca un email este folosit printre useri.

2)

|  |
| --- |
| // Functie originala  public update(id: number, user: User): User {  if (this.isEmailTaken(user.email)) {  throw new Error('Email is already taken')  }  const indexUser = this.findIndexById(id);  if (indexUser > -1) {  this.userList[indexUser] = user;  return user;  }  return undefined;  } |
| // forma cu mutant  public update(id: number, user: User): User {  if (true)) {  throw new Error('Email is already taken')  }  const indexUser = this.findIndexById(id);  if (indexUser > -1) {  this.userList[indexUser] = user;  return user;  }  return undefined;  } |

Mutantul a fost omorat, deaorece erau momente cand email nu era luat, adica functia ar fi trebuit sa returneze false, dar nu se intampla asta.

3)

|  |
| --- |
| // Functie originala  public update(id: number, user: User): User {  if (this.isEmailTaken(user.email)) {  throw new Error('Email is already taken')  }  const indexUser = this.findIndexById(id);  if (indexUser > -1) {  this.userList[indexUser] = user;  return user;  }  return undefined;  } |
| // forma cu mutant  public update(id: number, user: User): User {  if (this.isEmailTaken(user.email))) {  throw new Error('Email is already taken')  }  const indexUser = this.findIndexById(id);  if (indexUser >= -1) {  this.userList[indexUser] = user;  return user;  }  return undefined;  } |

Mutantul a fost omorat, deoarece in momentul in care nu exista niciun user cu un anumit id specificat, functia findIndexById imi va returna -1, adica vom folosi aceasta valoare pentru a ne duce pe cazul in care nu se poate actualiza user-ul, deci nu se potriveste cu functionalitatea noastra.

4)

|  |
| --- |
| // Functie originala  public deleteById(id: number): void {  this.userList = this.userList.filter((user) => user.id !== id);  } |
| // forma cu mutant  public deleteById(id: number): void {} |

Este evident ca functia noastra ar fi trebuit sa stearga user-ul cu id-ul respectiv, lucru ce nu se intampla, in momentul in care vom face testul pentru functia deleteById, testul va pica.

|  |
| --- |
| describe('deleteById', () => {  it('should be deleted.', () => {  const userList: User[] = [{ id: 1 } as User, { id: 2 } as User];  service['userList'] = [...userList];  service.deleteById(1);  expect.assertions(1);  expect(service.findAll())  .toEqual(  userList.filter(user => user.id !== 1)  );  });  }); |

5) Similar testelui anterior, functia careia i s-au aplicat muntantii a fost aceasi doar ca a avut urmatoarele variatii.

|  |
| --- |
| // Functie originala  public deleteById(id: number): void {  this.userList = this.userList.filter((user) => user.id !== id);  } |
| // Functie mutant 1  public deleteById(id: number): void {  this.userList = this.userList.filter((user) => user.id === id);  } |
| // Functie mutant 2  public deleteById(id: number): void {  this.userList = this.userList.filter((user) => true);  } |
| // Functie mutant 3  public deleteById(id: number): void {  this.userList = this.userList.filter((user) => undefiend);  } |
| // Functie mutant 4  public deleteById(id: number): void {  this.userList = this.userList.filter((user) => false);  } |

Este clar ca in primul car nu ar functiona deoarece imi returneaza array-ul cu fix un element, elementul cu id-ul meu, sau array-ul gol daca nu exista. Al doilea si ultimul caz imi va returna mereu array-ul gol, iar al treilea tot array-ul.

Restul mutantiilor s-au comportat similar cu aceste teste, in if-uri s-a inlocuit !== cu === sau false sau true, iar diferite blocuri din functiile ce returnau au fost inlocuite cu {} in loc de instructiuni, iar rezultatul a fost cel asteptat, testele au observat modificarile in cod.