A **TDD (Test-Driven Development)** és a **BDD (Behavior-Driven Development)** két népszerű fejlesztési módszer, amelyek tesztelési folyamatokra építenek, de eltérő szemlélettel és célkitűzésekkel.

Fő különbségek:

## 1. Fókusz

- **TDD (Test-Driven Development):**
  - A TDD középpontjában a **kód implementációja** áll. A cél az, hogy a fejlesztő **előre megírja a teszteket** a kód implementálása előtt, majd a tesztek alapján hozza létre a funkciókat. A TDD alacsonyabb szintű tesztekre (pl. unit tesztekre) összpontosít, amelyek konkrét kódrészleteket vizsgálnak.
  - A TDD lépései:
    - 1. Írj egy tesztet (amely először el fog bukni).
    - 2. Írd meg a kódot, hogy a teszt sikeres legyen.
    - 3. Refaktoráld a kódot, ha szükséges.
- **BDD (Behavior-Driven Development):**
  - A BDD a **viselkedésre** összpontosít, azaz arra, hogy a rendszernek hogyan kell viselkednie a felhasználó szempontjából. A tesztek a rendszer viselkedését írják le, nem pedig a kód részleteit. A BDD teszteket természetes nyelv közeli, mindenki által érthető formában írják meg, gyakran felhasználva a Gherkin szintaxist (`Given`, `When`, `Then` struktúrában).
  - A BDD célja az üzleti elemzők, fejlesztők és tesztelők közötti **együttműködés elősegítése**, hogy minden érintett jobban megértse a rendszer elvárt viselkedését.

## 2. Szint

- **TDD:**
  - Főként alacsony szintű (unit tesztek) tesztelésre összpontosít. A tesztek a kódrészletek helyes működését ellenőrzik.
- **BDD:**
  - Magasabb szintű tesztelés, amely a rendszer viselkedését vizsgálja, például hogyan reagál bizonyos felhasználói interakciókra vagy üzleti folyamatokra.

## 3. Szemléletmód

- **TDD:**
  - A kódtervezés **teszt-alapú**. A fejlesztő először tesztet ír, majd ehhez igazítja a kódot. A TDD során a fejlesztők inkább a funkciók implementálására és a kód helyességére koncentrálnak.
- **BDD:**
  - A tervezés **viselkedés-alapú**. A tesztek a rendszer által elvárt viselkedést írják le, tehát a felhasználói élményt és üzleti igényeket helyezik előtérbe.

## 4. Nyelvezet

- **TDD:**
  - Teszteket gyakran programozási nyelveken írják meg, amelyeket elsősorban a fejlesztők értenek. Például egy TDD teszt Pythonban, JUnitban stb. íródik.
- **BDD:**
  - A teszteket emberi nyelven közeli módon fogalmazzák meg, így nemcsak fejlesztők, hanem üzleti elemzők és más érintettek is megértik. A Gherkin szintaxis egy példa erre:

```
Given the user is on the login page
When they enter valid credentials
Then they should be logged in successfully
```

# Példa

A nulláról indulunk, lépésenként bemutatjuk a módszert.

## 1. Projekt inicializálása

Először hozzunk létre egy új projekt könyvtárat, és inicializáld a Node.js projektet.

```
mkdir tdd-project
cd tdd-project
npm init -y
```

Ez létrehoz egy alap **package.json** fájlt.

## 2. Függőségek telepítése

Telepítsük a szükséges fejlesztői függőségeket: **Mocha** a teszteléshez, **Chai** az asszertálásokhoz, és **Sinon** a mockoláshoz és stuboláshoz. Mivel a projektben jelszó hash-elésre is szükség lesz, telepítük a **bcrypt** könyvtárat is.

```
npm install mocha chai sinon bcrypt --save-dev
```

## 3. Mappastruktúra létrehozása

Hozzuk létre a szükséges mappákat és fájlokat a projekt szerkezetéhez.

```
mkdir test
mkdir services
```

```
mkdir repositories
ni ./test/userService.test.js
ni ./services/userService.js
ni ./repositories/userRepository.js
```

*megjegyzés*: az `ni` parancs powershell-ben a linuxos `touch` parancs megfelelője.

Most a projekt struktúrája így néz majd ki:

```
tdd-project/
│
├── test/
│   └── userService.test.js    // Tesztek a UserService-hez
│
├── services/
│   └── userService.js         // UserService osztály
│
├── repositories/
│   └── userRepository.js      // UserRepository osztály
│
└── package.json               // Node.js projekt leíró fájl
```

Létrejött három üres állomány.

## 4. Mocha konfigurálása

A **Mocha** futtatásához a `package.json` fájlban hozzá kell adni egy részt, amely a `mocha` parancsot futtatja a `test` mappában:

Nyissuk meg a `package.json` fájlt, és addjuk hozzá a `scripts` részhez a következőt:

```
"type": "module",
"scripts": {
  "test": "mocha"
}
```

## 5. Tesztek írása (TDD módszerrel)

Most kezdhetjük a TDD folyamatot: először a teszteket írdjuk meg. Például a `userRepository.test.js` fájlba írjuk a következő teszteket:

```
import assert from 'assert';
import UserRepository from '../repositories/userRepository.js';

describe('UserRepository', function() {
```

```
  let userRepository;

  beforeEach(function() {
    userRepository = new UserRepository();
  });

  it('should return null if user is not found by email', async function() {
    const user = await
userRepository.findUserByEmail('notfound@example.com');
    assert.strictEqual(user, null);
  });

  it('should save a user and retrieve it by email', async function() {
    const newUser = { email: 'test@example.com', password: 'hashedPassword'
};
    await userRepository.saveUser(newUser);
    const foundUser = await
userRepository.findUserByEmail('test@example.com');
    assert.strictEqual(foundUser.email, 'test@example.com');
    assert.strictEqual(foundUser.password, 'hashedPassword');
  });

  it('should handle saving multiple users', async function() {
    const user1 = { email: 'user1@example.com', password: 'password1' };
    const user2 = { email: 'user2@example.com', password: 'password2' };

    await userRepository.saveUser(user1);
    await userRepository.saveUser(user2);

    const foundUser1 = await
userRepository.findUserByEmail('user1@example.com');
    const foundUser2 = await
userRepository.findUserByEmail('user2@example.com');

    assert.strictEqual(foundUser1.email, 'user1@example.com');
    assert.strictEqual(foundUser2.email, 'user2@example.com');
  });
});
```

## 6. Tesztek futtatása

Futtasd a Mocha teszteket, hogy megbizonyosodj arról, hogy a tesztek elbuknak (mivel még nem írtad meg a tényleges implementációt).

```
npm test
```

Ez a parancs futtatja a `mocha` parancsot, amely végigmegy a `test` mappában lévő teszteken. Mivel a

`UserRepository` még nem implementált, a tesztek elbuknak, ami a TDD módszer lényege: először a tesztek buknak el, majd az implementáció következik.

```
UserService
  1) "before each" hook for "should return error if email is already in use"

0 passing (5ms)
1 failing

1) UserService
     "before each" hook for "should return error if email is already in use":
   TypeError: UserService is not a constructor
    at Context.<anonymous> (test\userService.test.js:14:19)
    at process.processImmediate (node:internal/timers:491:21)
```

## 7. Implementáció megírása

Most írd meg a tényleges kódot a tesztek sikeressé tételéhez.

`userRepository.js`:

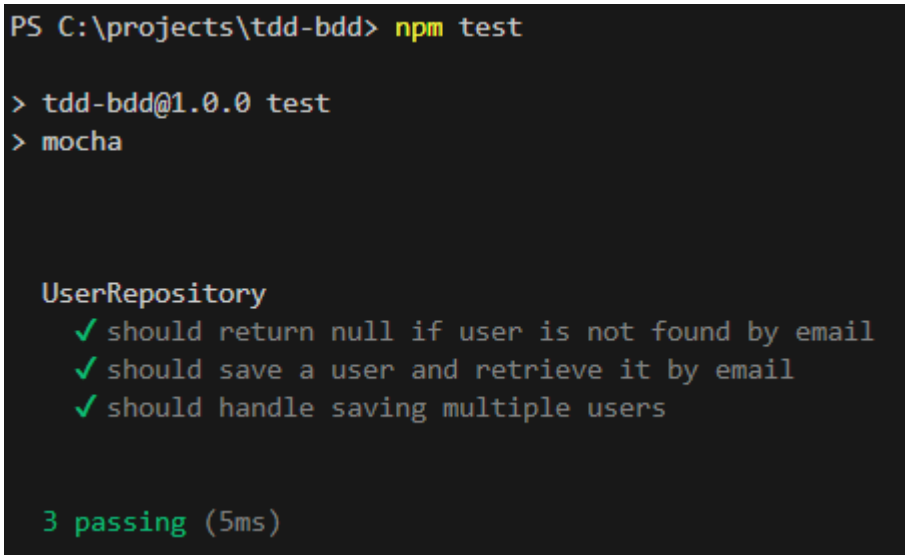
```
// repositories/userRepository.js

class UserRepository {
    constructor() {
        this.users = []; // Szimulált adatbázis tömbként
    }

    // Felhasználó keresése e-mail alapján
    async findUserByEmail(email) {
        const user = this.users.find(user => user.email === email);
        return user || null;
    }

    // Új felhasználó mentése az adatbázisba
    async saveUser(user) {
        this.users.push(user);
        return user;
    }
}

export default UserRepository;
```

## 8. Tesztek újrafuttatása

Most futtassuk újra a teszteket:

```
npm test
```

Most a teszteknek sikeresen át kell menniük, mivel az implementáció megfelel a tesztek elvárásainak.




## 9. BDD stílusú tesztek

Hozzuk létre a `test/userRepository.test.js`-t az alábbi tartalommal.

```
import { expect } from 'chai';
import sinon from 'sinon';
import bcrypt from 'bcrypt';
import UserRepository from '../repositories/userRepository.js';
import UserService from '../services/userService.js';

describe('UserService', function() {
  let userService;
  let userRepositoryStub;

  beforeEach(function() {
    // Mockoljuk az adatbázis hívásokat
    userRepositoryStub = sinon.stub(UserRepository.prototype,
'findUserByEmail');
    userService = new UserService(new UserRepository());
  });

  afterEach(function() {
    // Restore minden stubolt funkciót a tesztek után
    sinon.restore();
  });

  it('should return an error if the email is already in use', async
function() {
    // Szimuláljuk, hogy az email már létezik
```

```
    userRepositoryStub.resolves({ email: 'existing@example.com' });

    const result = await userService.registerUser('existing@example.com',
'password123');

    expect(result.success).to.be.false;
    expect(result.message).to.equal('Email already in use');
  });

  it('should hash the password and register the user if the email is not in
use', async function() {
    // Szimuláljuk, hogy az email nem létezik
    userRepositoryStub.resolves(null);

    // Mockoljuk a bcrypt hash funkciót
    const bcryptStub = sinon.stub(bcrypt,
'hash').resolves('hashedPassword');

    const result = await userService.registerUser('newuser@example.com',
'plainPassword');

    // Ellenőrizzük, hogy a bcrypt hash funkciót hívták
    expect(bcryptStub.calledOnce).to.be.true;
    expect(bcryptStub.calledWith('plainPassword')).to.be.true;
    expect(result.success).to.be.true;
    expect(result.message).to.equal('User registered successfully');
  });

  it('should not call bcrypt hash if the email already exists', async
function() {
    // Szimuláljuk, hogy az email már létezik
    userRepositoryStub.resolves({ email: 'existing@example.com' });

    const bcryptStub = sinon.stub(bcrypt, 'hash');

    const result = await userService.registerUser('existing@example.com',
'plainPassword');

    // Ellenőrizzük, hogy a bcrypt hash nem lett meghívva
    expect(bcryptStub.called).to.be.false;
    expect(result.success).to.be.false;
    expect(result.message).to.equal('Email already in use');
  });
});
```

A teszt futtatás természetesen hibát ad.

Hozzuk létre az `services/userService.js` implementációt az alábbiak szerint:

```
import bcrypt from 'bcrypt';

class UserService {
    constructor(userRepository) {
        this.userRepository = userRepository;
    }

    async registerUser(email, password) {
        const existingUser = await
this.userRepository.findUserByEmail(email);
        if (existingUser) {
            return { success: false, message: 'Email already in use' };
        }

        const hashedPassword = await bcrypt.hash(password, 10);
        const newUser = { email, password: hashedPassword };
        await this.userRepository.saveUser(newUser);

        return { success: true, message: 'User registered successfully' };
    }
}

export default UserService;  // CommonJS helyett exportáljuk ESM
szintaxissal
```