Bhartiya Vidya Bhavan's

Sardar Patel Institute of Technology

Bhavan's Campus, Munshi Nagar, Andheri (West), Mumbai-400058-India

(Autonomous College Affiliated to University of Mumbai)

NAME	Deepanshu Aggarwal, Kunal Bhatia
UID	2021300002, 2021300010
BATCH	Comps A (A)
PROJECT TITLE	Crime Reporting Website
EXPERIMENT NO.	7
AIM	To understand and implement the concept of Unit Testing by preparing test cases and performing Unit Testing for some program in each language/framework.

Theory:

Unit testing is a type of software testing that focuses on individual units or components of a software system. The purpose of unit testing is to validate that each unit of the software works as intended and meets the requirements. Unit testing is typically performed by developers, and it is performed early in the development process before the code is integrated and tested as a whole system. Unit tests are automated and are run each time the code is changed to ensure that new code does not break existing functionality. Unit tests are designed to validate the smallest possible unit of code, such as a function or a method, and test it in isolation from the rest of the system. This allows developers to quickly identify and fix any issues early in the development process, improving the overall quality of the software and reducing the time required for later testing. Unit Testing Techniques: There are 3 types of

Unit Testing Techniques:

- Black Box Testing: This testing technique is used in covering the unit tests for input, user interface, and output parts.
- White Box Testing: This technique is used in testing the functional behavior of the system by giving the input and checking the functionality output including the internal design structure and code of the modules
- Gray Box Testing: This technique is used in executing the relevant test cases, test methods, and test functions, and analyzing the code performance for the modules.

Advantages of Unit Testing:

1. Unit Testing allows developers to learn what functionality is provided by a unit and how

to use it to gain a basic understanding of the unit API.
2. Unit testing allows the programmer to refine code and make sure the module works properly.
3. Unit testing enables testing parts of the project without waiting for others to be completed.
4. Early Detection of Issues: Unit testing allows developers to detect and fix issues early in the development process before they become larger and more difficult to fix.
5. Improved Code Quality: Unit testing helps to ensure that each unit of code works as intended and meets the requirements, improving the overall quality of the software.
6. Increased Confidence: Unit testing provides developers with confidence in their code, as they can validate that each unit of the software is functioning as expected.

Mocha and Chai:

Mocha is a feature-rich JavaScript test framework that runs on both Node.js and in the browser. It provides a flexible and comprehensive testing environment for writing and executing tests, supporting various testing styles and methodologies. Mocha is known for its versatility, allowing developers to choose their preferred assertion libraries, mocking, libraries, and other testing tools. It supports asynchronous testing and provides hooks for setup and teardown operations, making it suitable for a wide range of testing scenarios.

Chai is a powerful assertion library for JavaScript and Node.js, designed to work seamlessly with various testing frameworks, with Mocha being one of the most popular ones. Chai provides a flexible and expressive syntax for writing assertions, giving developers the ability to choose a style that best fits their testing preferences.

Implementation

Here, we have implemented three functions named createCrimeArticle, deleteCrime and getCrimeReport based on our application which create the crime, delete the crime, and return the crime details respectively.

Code:

Index.js

```
const crimes = {
    crimes: [
      { id: 1, author: "John Doe", title:"1st test case", category:
"Murder", description: "Hello testing 123" },
      { id: 2, author: "Jane Smith", title:"2nd test case", category:
"Theft", description: "Hello testing 456" },
    ],
```

```
    deleteCrime: function (crimeId) {
      const crime = this.crimes.find(
        (crime) => crime.id === crimeId
      );

      if (crime) {
        this.crimes = this.crimes.filter(crime => crime.id !== crimeId);
        return `Successfully deleted crime article with crime id ${crimeId}`
      } else {
        return `Crime with crime id - ${crimeId} not found.`;
      }
    },

    getCrimeReport: function (crimeId) {
      const crime = this.crimes.find(
        (crime) => crime.id === crimeId
      );

      if (crime) {
        return crime;
      } else {
        return `Crime with crime id - ${crimeId} not found.`;
      }
    },

    createCrimeArticle: function (
      id,
      author,
      category,
      description,
      title,
    ) {
      const newCrimeArticle = {
        id: id, // Assuming the new ID is incremental
        author: author,
        category: category,
        description: description,
        title: title,
        //status: "pending", // New candidates start with a "pending" status
      };

      this.crimes.push(newCrimeArticle);

      console.log(`Crime " ${newCrimeArticle.title} " added to the crime
list.`);
```

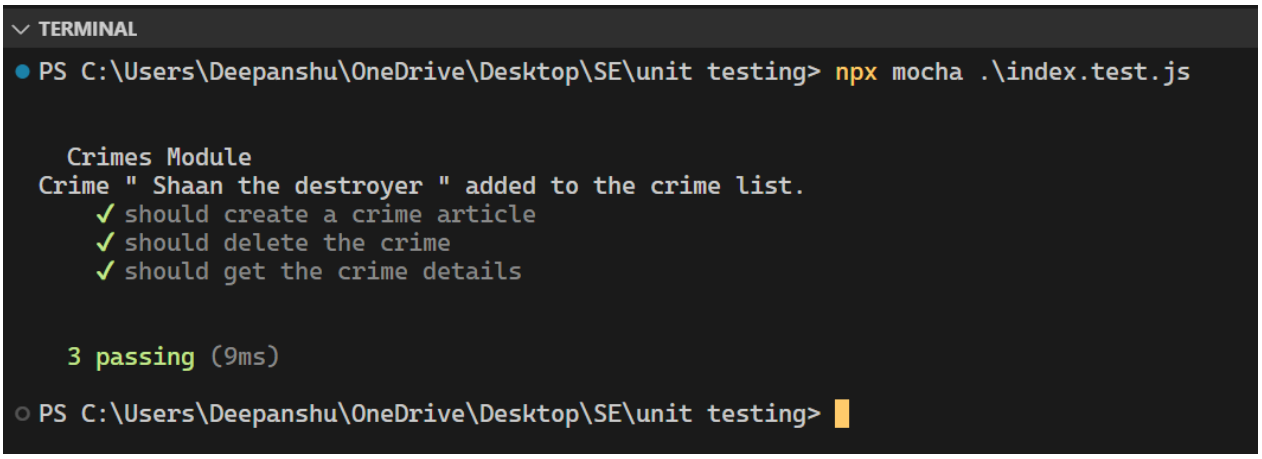
```
      return newCrimeArticle;
    },
  };

module.exports = crimes;
```

index.test.js

```
const chai = require("chai");
const crimes = require("./index"); // Correct path to your crimes module
const expect = chai.expect;

describe("Crimes Module", () => {
  it("should create a crime article", () => {
    const crime = crimes.createCrimeArticle(
       3,
      "Deep",
      "Murder",
      "Recently shaan killed a person with his boring coorporate stories",
      "Shaan the destroyer",
    );

    expect(crime).to.deep.equal({
        id: 3,
        author: "Deep",
        title:"Shaan the destroyer",
        category: "Murder",
        description: "Recently shaan killed a person with his boring
coorporate stories"
    });
  });

  it("should delete the crime", () => {
    const acceptMessage = crimes.deleteCrime(1); // Assuming the ID of the
candidate to accept is 1
    expect(acceptMessage).to.equal("Successfully deleted crime article with
crime id 1");
  });

  it("should get the crime details", () => {
    const crimeDet = crimes.getCrimeReport(2); // Assuming the ID of the
candidate to reject is 2
    expect(crimeDet).to.deep.equal({
        id: 2,
```

```
        author: "Jane Smith",
        title: "2nd test case",
        category: "Theft",
        description: "Hello testing 456"
    });
  });
});
```

Result:



Conclusion:

By performing this experiment, unit testing provides highly effective, swiftly identifying errors and enhancing code quality. Challenges included setup complexity and maintenance, but the efficiency gains outweigh these concerns.