Automatic Test Case Generator (Selenium)

Project Overview

The **Automatic Test Case Generator** is a Java-based automation tool that dynamically generates Selenium WebDriver test cases by analyzing a target website's HTML and DOM structure. Instead of manually writing Selenium scripts, this tool automatically identifies UI elements and produces basic test cases, saving time and reducing manual effort in test automation.

This project was developed as part of an internship to strengthen skills in **Java, Selenium, Maven, and Web Automation Frameworks**.

Key Features

- Automatically fetches and parses a given website URL
- Analyzes HTML and DOM structure using **JSoup**
- Identifies common UI elements (inputs, buttons, links, etc.)
- Generates basic Selenium WebDriver test cases
- Reduces manual coding and human errors in automation testing
- Maven-based project for easy dependency management

Technologies Used

- **Java**
- **Selenium WebDriver**
- **JSoup** (HTML parsing)
- **Maven** (Build & Dependency Management)
- **TestNG / JUnit**

Project Structure

```
selenium-autotest-upgrade/
│
├── src/main/java/
│   └── com/autotest/
│       └── Main.java
│
├── pom.xml
├── target/
```

```
│   └── selenium-autotest-upgrade-1.0-SNAPSHOT.jar
└── README.md
```

Prerequisites

- Java JDK 8 or above
- Maven
- Google Chrome / Any supported browser
- ChromeDriver (matching browser version)

How to Run the Project

1 Build the Project

```
mvn clean package
```

2 Run the JAR File

```
java -jar target/selenium-autotest-upgrade-1.0-SNAPSHOT.jar <target-url>
```

Example

```
java -jar target/selenium-autotest-upgrade-1.0-SNAPSHOT.jar
https://example.com
```

Output

- Console output showing fetched URL and identified elements
- Automatically generated Selenium test case logic (based on DOM analysis)

Future Enhancements

- Handle JavaScript-rendered pages (using Selenium + JS execution)
- Generate complete TestNG/JUnit test classes
- Add support for XPath & CSS selector optimization
- Export generated test cases to `.java` files
- Integrate reporting

Learning Outcomes

- Practical knowledge with Selenium automation
- Understanding DOM parsing and element identification

- Maven project structuring
- Practical exposure to automation frameworks

Author

Gokul M
Intern – Automation Testing