Automated Test Case Generation

TEAM BST
Bhargava Elavarthi, Tim Son, Srikaran Bachu, Patrick Hardy

Problem Statement

The Challenge of Manual Test Case Creation

- Time-Consuming Process: Manual test case creation is labor-intensive, diverting resources from critical development tasks.
- Inefficiency and Delays: Slows down project timelines, impacting productivity and software delivery.
- Lack of Standardization: Hinders collaboration across teams, complicating communication on technical tasks.
- Integration Issues: Difficulty in integrating testing with CI/CD pipelines and version control systems, affecting early issue detection and software quality.

Proposed Solution

Proposal: Launch an Automated Test Case Generator to revolutionize software testing.

Objectives: Simplify test creation, boost developer efficiency, and enhance collaboration.

Benefits:

Efficiency: Automates routine testing, freeing up time for development.

Productivity: Accelerates project timelines with faster test case generation.

Collaboration: Regulates tests to improve team communication and consistency.

Quality: Increases test accuracy and coverage, improving software reliability.

Related Work

- AI-Infused Test Automation: "AI-Infused Test Automation: Revolutionizing Software Testing through Artificial Intelligence" by Anup Sahoo. https://www.sei.cmu.edu/our-work/artificial-intelligence-engineering/
- Model-Based Testing (MBT): Explore recent advancements in MBT, a technique that generates test cases from models representing system behavior. https://www.researchgate.net/publication/222425707_Classifier_ensembles_Select_real-world_applications
- Test Generation Frameworks: Overview of frameworks like Selenium, discussing their limitations and the potential of AI to overcome these challenges. https://www.selenium.dev/
- Academic Research: Significant studies on machine learning in test case generation, such as using algorithms to predict defect-prone areas. MIT Research on AI and Testing.

Tech Stack

- **User-Friendly Frontend:** Built with React for robust, scalable user interfaces, combined with Tailwind CSS for rapid, responsive design. This combination enhances user experience and facilitates easy navigation.
- **Flask Integration for Backend:** Employs Flask, a lightweight and flexible Python web framework, to handle backend operations. This setup is ideal for rapid development and easy integration with the OpenAI API.
- **OpenAI ChatGPT API:** Utilizes advanced AI to dynamically generate test cases based on user inputs, ensuring high-quality, relevant test scenarios.

Relevance

- Increases Team Knowledge of Tasks:

  - Encourages a deep understanding of project requirements through shared test cases, helping team members grasp the full scope and details of what needs to be built or fixed.

  - Facilitates learning and adoption of best practices in testing and coding standards across the team.

- Improves Team Efficiency:

  - Minimizes repetitive manual testing efforts, allowing team members to focus on complex problem-solving and innovation.

  - Streamlines the development process with automated, consistent test executions, significantly reducing the time from development to deployment.

- Improves Team Communication:

  - Standardizes the language and criteria for quality and success through automated tests, making technical discussions more productive and focused.

Class Concept #1 Discussed in Class

Scrum/Sprints

Overview: Introduction to Scrum as a flexible framework for managing complex projects through iterative Sprints.

Application in Project:

Sprint Planning: Scheduled meetings to define what can be delivered in the sprint and how the work will be achieved.

Daily Scrum: Brief daily meetings for the development team to synchronize activities and create a plan for the next 24 hours.

Benefits:

Adaptability: Quick adaptation to project changes based on continuous feedback.

Milestone Tracking: Regular sprint reviews ensure that the project milestones are met on time.

Class Concept #2 Discussed in Class

GitHub Version Control

Overview of GitHub as a powerful tool for version control and collaborative software development.

Usage in Project:

Branches: Managed new features and bug fixes separately, ensuring stability in the main branch.

Pull Requests and Code Review: Enabled thorough review processes before integration, enhancing code quality.

Issue Tracking: Organized task management and bug tracking to streamline workflows.

Advantages:

Team Collaboration: Simplified contributions and feedback among team members.

Historical Record: Provided a detailed record of changes, aiding in debugging and understanding project evolution.

Class Concept #3 Discussed in Class

Agile Development

Principles:

Emphasis on flexibility, team collaboration, and customer feedback in software development.

Implementation:

Iterative Development: Divided the project into manageable phases allowing for regular assessment and adaptation.

Feedback Integration: Continuously integrated user and stakeholder feedback to better align the product with user needs.

Impact:

Efficiency: Reduced time-to-market with faster release cycles.

Product Quality: Incremental improvements led to a higher quality final product.

Planned Features/Use Cases

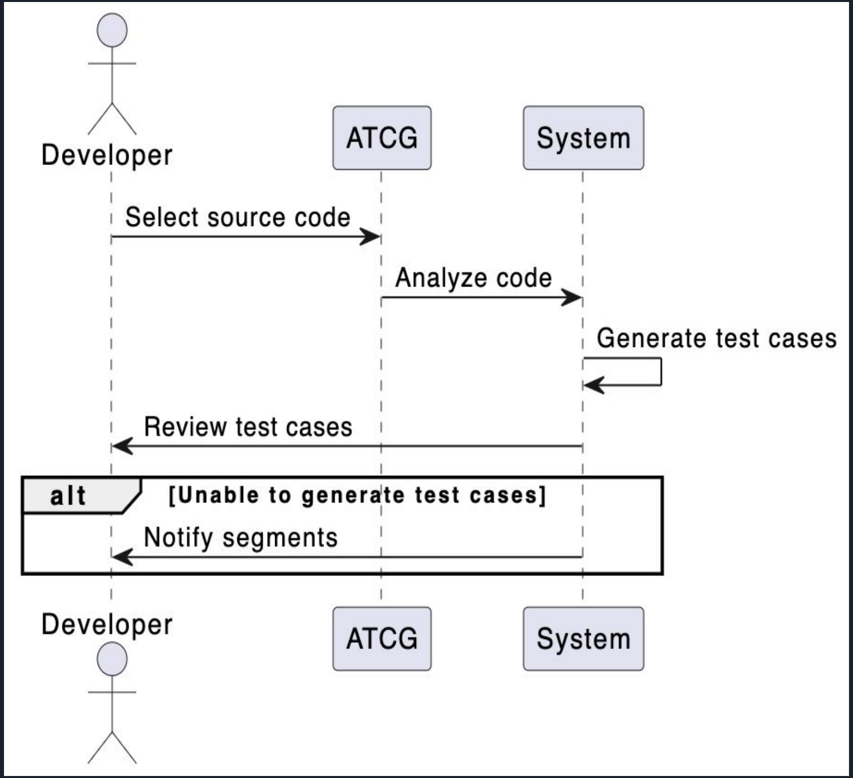
#1 Use Case

Generate Test Cases: Goal is to generate test cases automatically for a given piece of code or description

Primary Actor: Developer

Precondition: Dev already has access to source code and ATCG
Main Scenario: Dev selects source code for with test cases are needed

Extensions: System is unable to generate test cases for specific code segments
System notifies developer of segments where it cannot be generated and developer manually completes them



#2 Use Case

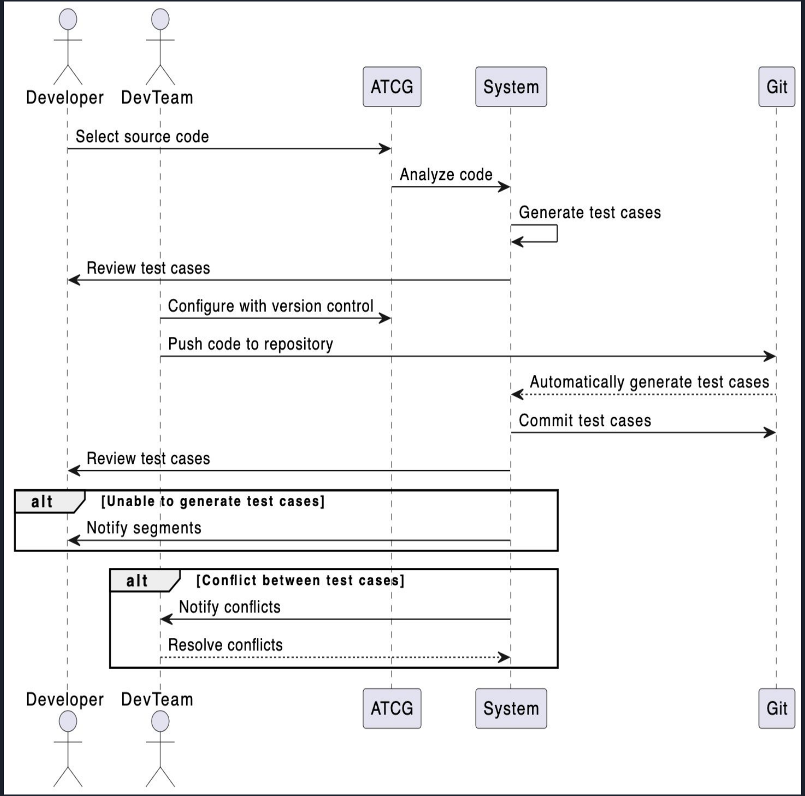
Integrate with Version Control: Goal is to integrate the Automated Test Case Generator with team’s version control system

Primary Actor: Dev team

Precondition: Dev team has version control system like git

Main Scenario: Team configures ATCG w their version control system
When code is pushed to repository, sys automatically generates test cases

Extensions: Conflict between new test cases and existing ones
System notifies team of conflicts
Team resolves conflicts manually



#3 Use Case

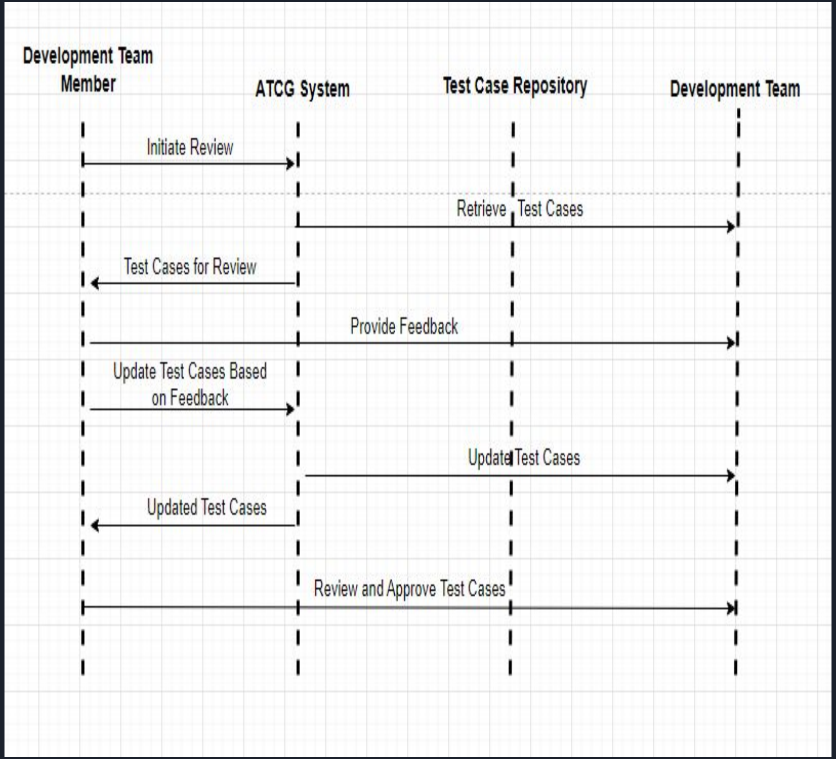
Execute Test Cases: Goal is to automatically execute generated test cases as part of the CI pipe

Primary Actor: Continuous integration system

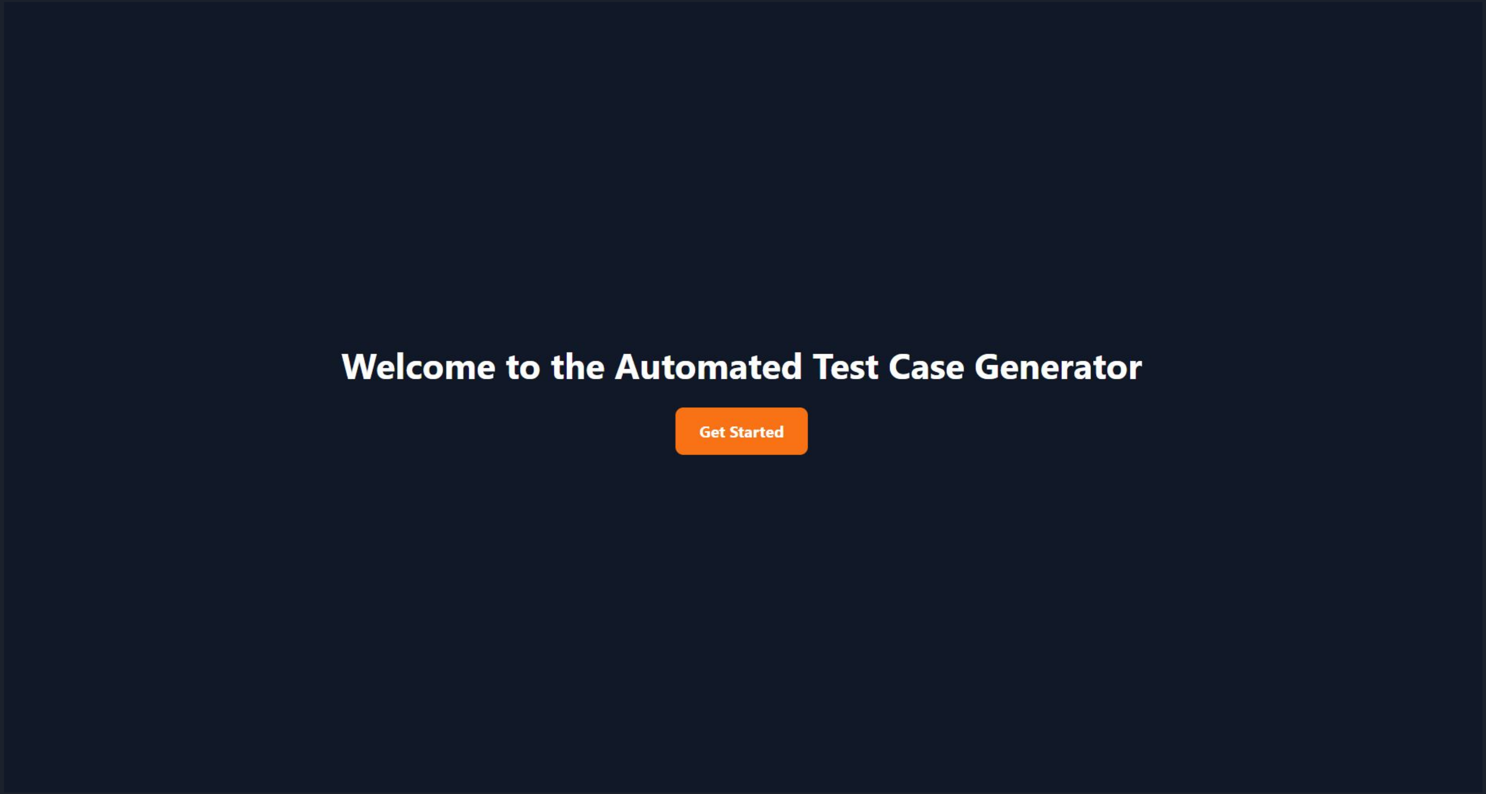
Precondition: Test cases generated and available in repository

Main Scenario: New commit triggers CI pipeline
CI system fetches latest cases

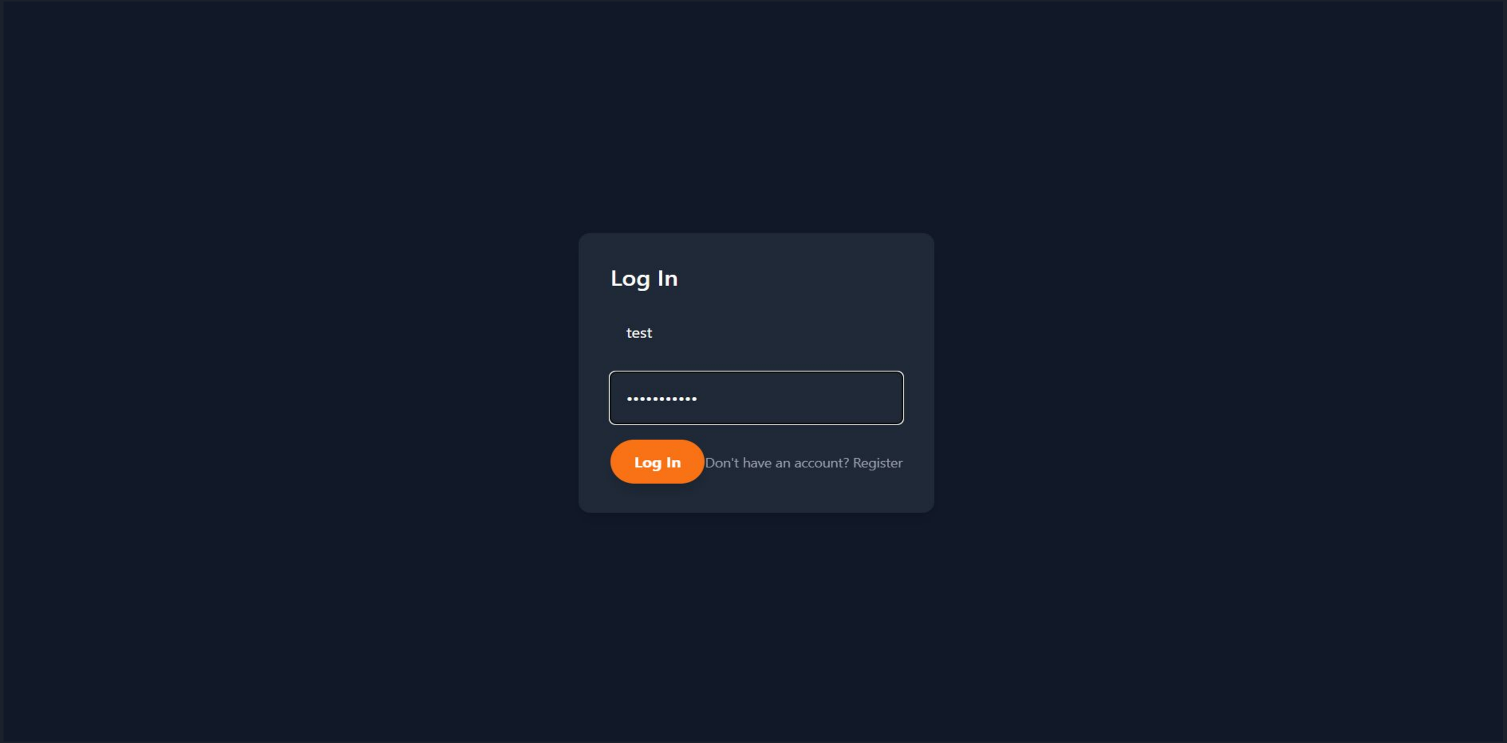
Extensions: Some tests fail
System notifies devs who investigate failures



Frontend


Welcome to the Automated Test Case Generator

Frontend

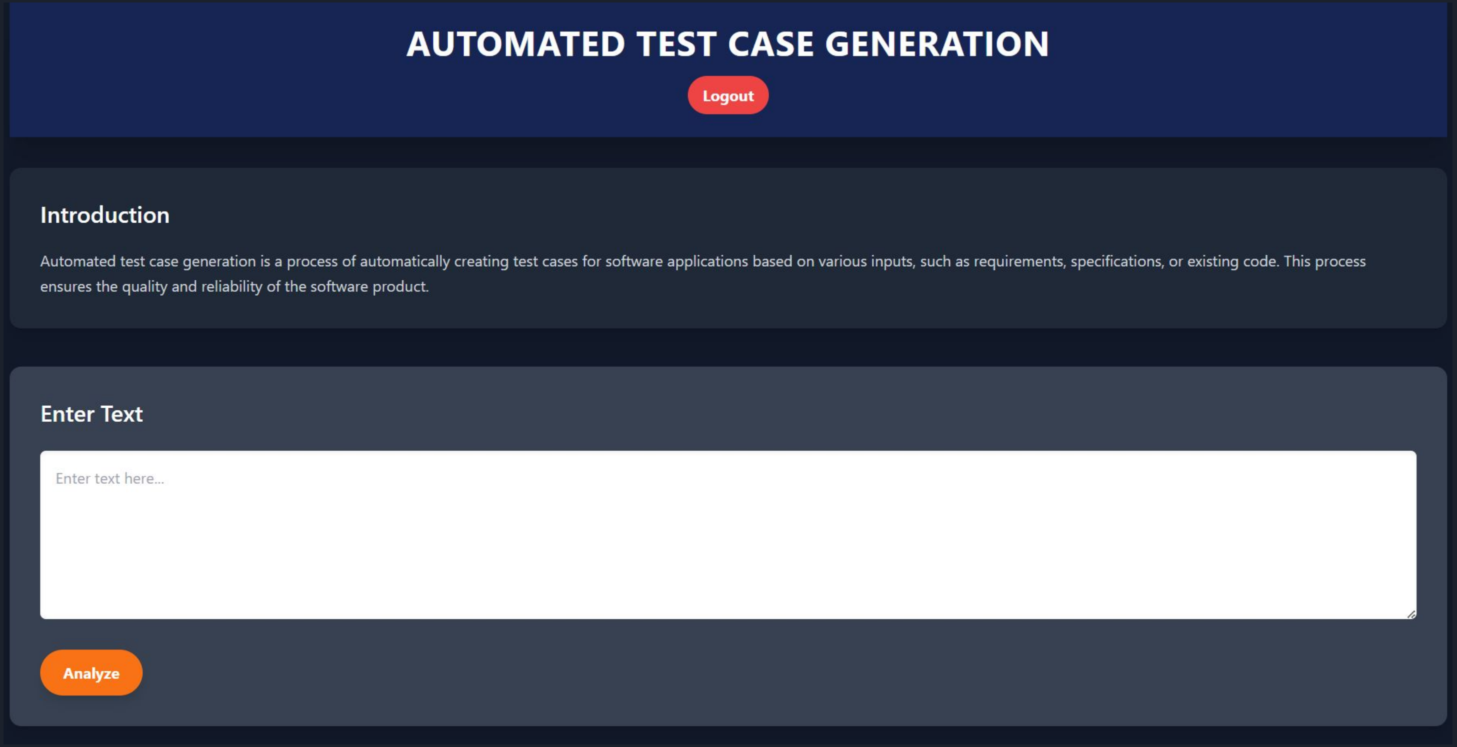


Log In

test

•••••••••••

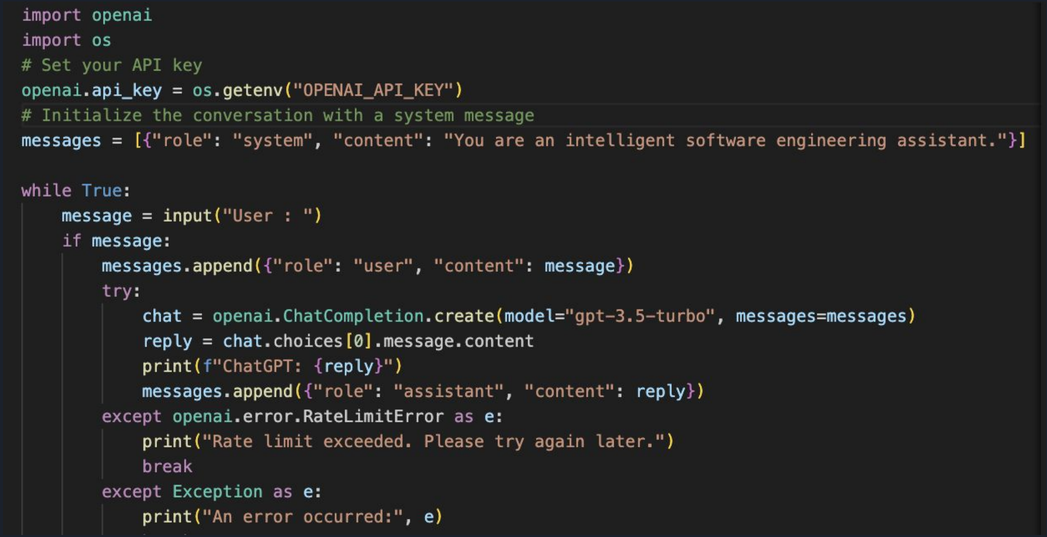
Frontend



Introduction

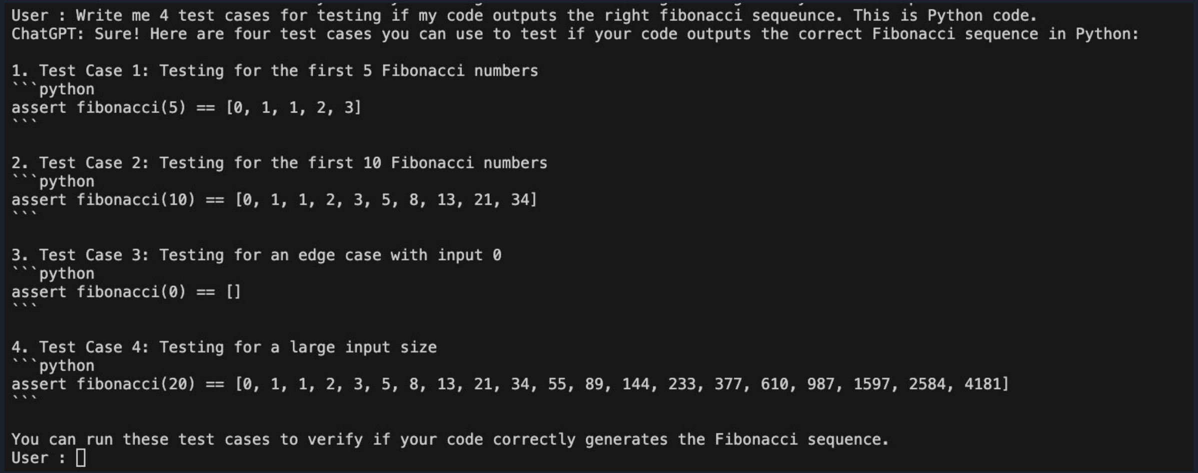
Automated test case generation is a process of automatically creating test cases for software applications based on various inputs, such as requirements, specifications, or existing code. This process ensures the quality and reliability of the software product.

Backend



```python
import openai
import os
# Set your API key
openai.api_key = os.getenv("OPENAI_API_KEY")
# Initialize the conversation with a system message
messages = [{"role": "system", "content": "You are an intelligent software engineering assistant."}]

while True:
    message = input("User : ")
    if message:
        messages.append({"role": "user", "content": message})
        try:
            chat = openai.ChatCompletion.create(model="gpt-3.5-turbo", messages=messages)
            reply = chat.choices[0].message.content
            print(f"ChatGPT: {reply}")
            messages.append({"role": "assistant", "content": reply})
        except openai.error.RateLimitError as e:
            print("Rate limit exceeded. Please try again later.")
            break
        except Exception as e:
            print("An error occurred:", e)
```



~~~
User : Write me 4 test cases for testing if my code outputs the right fibonacci sequeunce. This is Python code.
ChatGPT: Sure! Here are four test cases you can use to test if your code outputs the correct Fibonacci sequence in Python:

1. Test Case 1: Testing for the first 5 Fibonacci numbers
```python
assert fibonacci(5) == [0, 1, 1, 2, 3]
```

2. Test Case 2: Testing for the first 10 Fibonacci numbers
```python
assert fibonacci(10) == [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
```

3. Test Case 3: Testing for an edge case with input 0
```python
assert fibonacci(0) == []
```

4. Test Case 4: Testing for a large input size
```python
assert fibonacci(20) == [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]
```

You can run these test cases to verify if your code correctly generates the Fibonacci sequence.
User : 
~~~

Basic Demo!

Future Work

Goal: Integrate sophisticated AI algorithms for more accurate and relevant test case generation.

Details:

Technologies: Deploy machine learning models that utilize extensive datasets including code changes, user behavior, and performance metrics.

Methodologies: Implement continuous learning and predictive analytics to enhance decision-making and pinpoint potential failure points.

Impact:

Enhanced Capabilities: The tool will manage complex scenarios with greater precision, reducing manual testing effort.

Deeper Insights & Efficiency: Offer deeper insights into potential software failures and improve the efficiency and accuracy of test generation, leading to faster, more reliable outcomes.

Any Questions?