Санкт-Петербургский Национальный Исследовательский Университет

Информационных Технологий, Механики и Оптики

Факультет Инфокоммуникационных Технологий

Тестирование программного обеспечения

Лабораторная работа 2

Выполнил:

Колсанов Я. И.

Проверил:

Кочубеев Н. С.

Санкт-Петербург, 2024

Цель: научиться разрабатывать и применять интеграционные тесты для проверки взаимодействия между компонентами в существующем проекте. Провести анализ взаимодействий и подготовить отчёт о проведённом тестировании.

Задачи:

1. Выбрать открытый проект на платформе GitHub.
2. Изучить структуру выбранного проекта и определить, какие модули или компоненты взаимодействуют друг с другом.
   Также уделить особое внимание:
   - ключевым точкам интеграции, где модули обмениваются данными или вызывают функции друг друга;
   - критическим частям системы, от которых зависит работа всей программы;
   - важным сценариям взаимодействия (например, вызов API, работа с базой данных или обмен сообщениями между сервисами).
3. Написать минимум 5 интеграционных тестов, проверяющих основные сценарии взаимодействий, включая граничные случаи и потенциальные ошибки.

Ход работы

Ссылка на репозиторий с тестами:
https://github.com/yaroslavkolsanov/TestPO

1. Выбор проекта на GitHub

Был выбран
https://github.com/vladislav-bordiug/ChatGPT_DALL_E_StableDiffusion_Telegram_Bot

Он представляет из себя Telegram-бот, написанный на Python, который позволяет общаться с ChatGPT и генерировать изображения с помощью DALL·E и Stable Diffusion, платежи реализованы с помощью Crypto Bot. В проекте используются база данных и API.

Основная логика и функционал бота содержатся в файле bot.py. Модуль db.py предназначен для работы с базой данных. А openaitools.py — модуль для работы с OpenAI API. Реализует методы для отправки запросов в ChatGPT и генерации изображений через DALL·E на основе переданных промптов. stablediffusion.py — модуль для взаимодействия со Stable Diffusion API. Содержит функцию для генерации изображений на основе текстовых описаний, которая возвращает сгенерированные изображения для отправки пользователю.

Интеграционные тесты для проекта также написаны на Python.

2. Анализ взаимодействий между модулями

1. bot.py

Связь с db.py: бот постоянно взаимодействует с базой данных, получая и обновляя информацию о пользователях, проверяя доступные токены и генерации, записывая новые счета.

Ключевые точки интеграции:

- При обработке запросов на генерацию текста или изображений бот проверяет наличие токенов и генераций через функции db.py.

- При выставлении счета за покупку токенов или генераций бот обновляет данные о заказах и токенах после подтверждения оплаты.

Связь с openaitools.py и stablediffusion.py: бот использует эти модули для генерации ответов на текстовые запросы и создания изображений.

Связь с cryptopay.py: бот создает счета для платежей с использованием функций из cryptopay.py. Когда пользователь хочет пополнить баланс токенов или генераций, бот вызывает соответствующую функцию для создания счета и получения статуса оплаты.

2. db.py

Связь с bot.py: модуль взаимодействует с bot.py, отвечая за получение и запись информации о пользователях, токенах, генерациях и покупках. Бот использует функции из db.py для записи состояния пользователя и для создания новых записей о счетах и статусах.

Критическая часть системы:

Корректная работа с базой данных необходима для поддержания актуальной информации о пользователях, их доступных токенах и генерациях, а также для управления счетами.

3. openaitools.py

Связь с bot.py: основная задача модуля — передать текстовые запросы на API ChatGPT и получить ответы. Модуль получает входные данные от бота, формирует запрос к API и возвращает результат в bot.py.

Критическая часть системы:

Этот модуль зависит от доступности OpenAI API, так как именно через него бот отвечает на вопросы пользователя.

4. stablediffusion.py

Связь с bot.py: модуль получает запросы на генерацию изображений от bot.py и передает их в API. После получения сгенерированного изображения модуль отправляет его в bot.py для отправки пользователю.

Ключевая точка интеграции:

Так как пользователи могут использовать Stable Diffusion для генерации изображений, важно обеспечить надежную передачу данных и обработку полученных изображений.

5. cryptopay.py

Связь с bot.py: модуль получает запросы на создание счетов от bot.py. После создания счета информация возвращается в bot.py, где происходит обработка дальнейших действий.

Ключевая точка интеграции:

Оплата и проверка счетов. Без корректной работы cryptopay.py бот не сможет обрабатывать платежи и предоставлять доступ к токенам или генерациям, что критично для монетизации бота.

Критические части системы:

1. Связь с базой данных.

Обеспечение корректного хранения информации о пользователях и статусе их покупок.

2. Связь с OpenAI API и Stable Diffusion API.

Так как бот предоставляет чат и генерацию изображений, ошибки в этих модулях или недоступность API может существенно снизить качество обслуживания пользователей.

3. Обработка платежей через CryptoPay.

Этот модуль отвечает за создание счетов и проверку статуса оплаты, что критично для предоставления пользователям доступа к токенам и генерациям.

3. Написание тестов

Далее были написаны тесты для проверки интеграций с использованием библиотеки unittest и unittest.mock.

1. Проверка успешного взаимодействия с API при корректном ответе от сервера.

Данный тест проверяет, что метод get_stable возвращает ожидаемый контент изображения при успешном запросе. Используется patch для имитации ответа от сервера API (через requests.post). С помощью mock объекта устанавливается статус-код 200, что означает успешный ответ, и содержимое ответа как данные изображения b"image data".

```python
import unittest
from unittest.mock import patch, Mock
import asyncio
from stablediffusion import StableDiffusion


class TestStableDiffusionIntegration(unittest.IsolatedAsyncioTestCase):

    async def test_stable_diffusion_success(self):
        with patch("requests.post") as mock_post:
            mock_response = Mock()
            mock_response.status_code = 200
            mock_response.content = b"image data"
            mock_post.return_value = mock_response

            response = await StableDiffusion.get_stable("Test prompt")
            self.assertEqual(response, b"image data", "Должен возвращаться контент изображения")
```

Рисунок 1 – Тестирование успешного взаимодействия с API при корректном ответе от сервера

2. Проверка поведения метода при ошибке от API.

Данный тест проверяет, как метод get_stable обрабатывает ошибочные ответы от API, например, когда сервер возвращает статус-код 401 - неавторизован. Имитация такой ошибки происходит с помощью mock объекта и устанавливается пустое содержимое ответа. Ожидается, что метод get_stable вернет None в случае ошибки, что соответствует логике обработки ошибок в коде.

```python
async def test_stable_diffusion_api_error(self):
    with patch("requests.post") as mock_post:
        mock_response = Mock()
        mock_response.status_code = 401
        mock_response.content = b""
        mock_post.return_value = mock_response

        response = await StableDiffusion.get_stable("Test prompt")
        self.assertIsNone(response, "При ошибке должен возвращаться None")
```

Рисунок 2 – Тестирование при ошибке от API

3. Проверка правильности работы с несколькими асинхронными вызовами одновременно.

Данный тест проверяет, что метод get_stable корректно работает при нескольких параллельных асинхронных вызовах. Используется asyncio.gather для выполнения двух запросов одновременно с разными промптами. Поскольку get_stable — это асинхронный метод, необходимо удостовериться, что он корректно обрабатывает несколько запросов одновременно. Ожидается, что оба запроса вернут одинаковый контент. Также проверяется, что количество ответов из gather равно 2 и каждый из них содержит правильные данные.

```python
async def test_async_behavior(self):
    with patch("requests.post") as mock_post:
        mock_response = Mock()
        mock_response.status_code = 200
        mock_response.content = b"image data"
        mock_post.return_value = mock_response

        response = await asyncio.gather(
            StableDiffusion.get_stable("Prompt 1"),
            StableDiffusion.get_stable("Prompt 2")
        )
        self.assertEqual(len(response), 2, "Должны получить два результата")
        self.assertEqual(response[0], b"image data")
        self.assertEqual(response[1], b"image data")
```

Рисунок 3 – Тестирование асинхронных вызовов

4. Проверка поведения при отсутствии API ключа.

Данный тест проверяет, как метод get_stable будет работать, если API ключ отсутствует. Также используется patch для замены функции getenv на возвращение None, что имитирует отсутствие ключа API. Ожидается, что метод get_stable вернет None, поскольку ключ не был найден.

```
async def test_missing_api_key(self):
    with patch("stablediffusion.getenv", return_value=None):
        response = await StableDiffusion.get_stable("Test prompt")
        self.assertIsNone(response, "Должен возвращаться None, если ключ API отсутствует")
```

Рисунок 4 – Тестирование при отсутствии API ключа

5. Проверка поведения при передаче пустого промпта.

Данный тест проверяет, как метод get_stable работает, если в качестве промпта передается пустая строка. Ожидается, что метод должен вернуть None, поскольку пустой промпт не является допустимым запросом для генерации изображения.

```
async def test_invalid_prompt(self):
    response = await StableDiffusion.get_stable("")
    self.assertIsNone(response, "Должен возвращаться None для пустого промпта")
```

Рисунок 5 – Тестирование при передаче пустого промпта

4. Результаты тестирования

Все тесты пройдены успешно. Тестирование покрывает ключевые сценарии взаимодействия с API и проверяет обработку ошибок для взаимодействия со Stable Diffusion. Эти тесты проверяют как корректную работу в обычных условиях, так и поведение системы при ошибках и неправильных данных, что важно для обеспечения стабильности работы программы. Однако для обеспечения корректной работы всех модулей, стоит также произвести тестирование для работы с ChatGPT, DALL·E и Crypto Bot.

Вывод: в результате выполнения лабораторной работы был проведен анализ взаимодействий между модулями выбранного проекта, написаны интеграционные тесты, проверяющие корректность взаимодействия между ключевыми компонентами системы.