

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

До захисту допущено

Завідувач кафедри

Віталій РОМАНКЕВИЧ

(підпис)

(ініціали, прізвище)

“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

**за освітньо-професійною програмою «Системне програмування та
спеціалізовані комп'ютерні системи»**

спеціальності

123 «Комп'ютерна інженерія»

на тему: Програмні засоби тестування ботів месенджера Telegram

Виконав:

студент IV курсу, групи КВ-62

Стадниченко Антон Вячеславович

(прізвище, ім'я, по батькові)

(підпис)

Керівник ст. викл. Дробязко І.П.

(посада, науковий ступінь, вчене звання, прізвище та ініціали)

(підпис)

Консультант з нормоконтролю, доц. каф. СПСКС, к.т.н. Клятченко Я.М.

(назва розділу)

(посада, вчене звання, науковий ступінь, прізвище, ініціали)

(підпис)

Рецензент

(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали)

(підпис)

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент

(підпис)

Київ – 2020 року

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»**

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування та спеціалізовані комп'ютерні системи»

ЗАТВЕРДЖУЮ

Завідувач кафедри

Віталій РОМАНКЕВИЧ
(підпис) (ініціали, прізвище)

«__»2019 р.

ЗАВДАННЯ

на дипломний проєкт студента

Стадниченко Антона Вячеславовича _____
(прізвище, ім'я, по батькові)

1. Тема проєкту: Програмні засоби тестування ботів месенджера Telegram _____

керівник проєкту ст. викл. Дробязко І.П. _____ ,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «__» _____ 20__ р. № _____

2. Термін подання студентом проєкту: _____

3. Вихідні дані до проєкту: див. Технічне завдання

4. Зміст пояснювальної записки:

- Аналіз існуючих рішень для тестування ботів Telegram;
- Аналіз і вибір технологічних і програмних засобів розроблення програмного проєкту;
- Опис розробленого програмного продукту;

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо):

- Блок-схема алгоритму виконання тесту. Схема структурна;
- Програмні засоби тестування ботів месенджера Telegram. Схема класів;
- Програмні засоби тестування ботів месенджера Telegram. Схема структурна;
- Блок-схема алгоритму завантаження плагіну. Схема структурна;
- Презентація за темою дипломного проєкту.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я. М., доцент		

7. Дата видачі завдання: 25 жовтня 2019

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	15.11.2019	
2.	Розроблення та узгодження технічного завдання	30.11.2019	
3.	Аналіз існуючих рішень	05.02.2020	
4.	Підготовка матеріалів першого розділу дипломного проєкту	05.05.2020	
5.	Підготовка матеріалів другого розділу дипломного проєкту	07.05.2020	
6.	Підготовка матеріалів третього розділу дипломного проєкту	10.05.2020	
7.	Підготовка матеріалів четвертого розділу дипломного проєкту	14.05.2020	
8.	Підготовка графічної частини дипломного проєкту	20.05.2020	
9.	Оформлення документації дипломного проєкту	23.05.2020	

Студент

(підпис)

Антон СТАДНИЧЕНКО

Керівник проєкту

(підпис)

Ірина ДРОБЯЗКО

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (51 с., 15 рис., 2 додатки).

Об'єкт розробки – програмні засоби тестування ботів месенджеру Telegram.

Метою дипломного проєкту є дослідження і структуризація теоретичних відомостей, необхідних для розроблення програмних засобів тестування ботів Telegram і їх розробка на основі отриманих знань.

В результаті роботи було реалізовано програмне забезпечення, у якому було використано оптимальну архітектуру для поставленої задачі. Перевагою даного проєкту є комунікація з серверами Telegram напряму, замість емуляції серверу, що дає більш реалістичну картину під час виконання тестів.

В ході виконання дипломного проєкту:

- Проведено аналіз існуючих рішень;
- Проведено аналіз технологій та програмних засобів розробки;
- Розроблено програмні засоби тестування ботів месенджеру Telegram.

Результати дипломної роботи можуть бути використанні для проведення тестування власних ботів. А можливість розширення функціоналу дозволяє адаптувати розроблене ПЗ для власних потреб.

Ключові слова: ТЕСТУВАННЯ, БОТИ, TELEGRAM, МІКРОЯДЕРНА АРХІТЕКТУРА, МЕСЕНДЖЕРИ.

ABSTRACT

Qualification work Includes Explanatory Note (51 pp., 15 figs., 2 Appendices).

Object of development - Software tools for testing bots messenger Telegram.

The purpose of the diploma project is to study and structure the theoretical information needed to develop software for testing Telegram bots and their development based on the acquired knowledge.

As a result, the software was implemented with usage of the optimal architecture for the task. The advantage of this project is communication with Telegram servers directly, instead of server emulation, which gives a more realistic picture during the tests.

During the implementation of the diploma project:

- The analysis of the existing solutions is carried out;
- The analysis of technologies and software means of development is carried out;
- Developed software tools for testing bots of Telegrammessenger.

The results of the thesis can be used to test your own bots. And the possibility of expanding the functionality allows you to adapt software development for your own needs.

Keywords: TESTING, BOTS, TELEGRAM, MICROKERNEL ARCHITECTURE, MESSENGERS

[illegible]

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	A4	ІАЛЦ. 045490.005	Програмні засоби	1		
			тестування ботів			
			месенджера Telegram.			
			Схема структурна.			
	A4	ІАЛЦ. 045490.006	Програмні засоби	1		
			тестування ботів			
			месенджера Telegram.			
			Схема класів.			
	A4	ІАЛЦ. 045490.007	Виконання тесту.	1		
			Схема алгоритму.			
	A4	ІАЛЦ. 045490.008	Завантаження плагіну	1		
			плагіну.			
			Схема алгоритму.			

					ІАЛЦ. 045490.001 ОА		Арк.
							2
Змін.	Арк.	№ докум.	Підпис	Дата			

Зміст

НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ	2
ПІДСТАВА ДЛЯ РОЗРОБКИ	2
ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ	2
ДЖЕРЕЛА РОЗРОБКИ	2
ТЕХНІЧНІ ВИМОГИ	3
Вимоги допрограминого продукту, що розробляється	3
Рекомендовані вимоги до апаратного забезпечення	3
Вимоги до програминого забезпечення	3
ЕТАПИ РОЗРОБКИ	4

					ІАЛЦ.045490.002 ТЗ					
Зм.	Арк.	№ докум.	Підп.	Дата	Програмні засоби тестування ботів месенджеру Telegram Технічне завдання			Літ.	Аркуш	Аркушів
Розроб.	Стадниченко А.В								1	4
Перевір.	Дробязко І.П.									
								НТУУ «КПІ ім. І. Сікорського»,ФПМ,КВ-62		
Н. контр.	Клятченко Я.М.									
Затв.	Романкевич В.О.									

НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Найменування роботи – «Програмні засоби тестування ботів месенджера Telegram».

Область застосування: інформаційні технології.

ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування та спеціалізованих комп'ютерних систем Національного технічного університету України «Київський Політехнічний Інститут імені Ігоря Сікорського».

ЦІЛЬ І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проєкту є розробка програмного продукту, що забезпечує створення та виконання тестів для ботів, працюючих на платформі Telegram. Це призведе до значного підвищення якості кінцевого продукту, адже усі виявлені недоліки будуть виправлені до релізу.

ДЖЕРЕЛА РОЗРОБКИ

Джерелами інформації для розроблення є технічна література, публікації у періодичних виданнях та Інтернет ресурси з питань розробки.

					ІАЛЦ.045490.002 ТЗ	Арк.
Зм.	Арк.	№ докум.	Підп.	Дата		2

ТЕХНІЧНІ ВИМОГИ

Вимоги допрограминого продукту, що розробляється

Програмний продукт повинен забезпечувати наступні функції:

- Обмін даними з серверами Telegram;
- можливість тестування ботів, незалежно від особливостей їх створення;
- зручний інтерфейс для керування процесом тестування;
- можливість розширення базового функціоналу розробленими користувачем модулями;

Рекомендовані вимоги до апаратного забезпечення

- Процесор: Intel Core i5.
- Оперативна пам'ять: 4 Гб.
- Простір на диску: 1Гб.

Вимоги до програмного забезпечення

- Операційна система Windows, Linuxчи MacOS X.
- Інтерпретатор мови PHP версії 7.4.
- Менеджер пакетів Composer.
- Ключ доступу до TelegramAPI.

					ІАЛЦ.045490.002 ТЗ	Арк.
Зм.	Арк.	№ докум.	Підп.	Дата		3

ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів
1.	Вивчення літератури за тематикою проєкту	15.11.2019
2.	Розроблення та узгодження технічного завдання	30.11.2019
3.	Аналіз існуючих рішень	05.02.2020
4.	Підготовка матеріалів першого розділу дипломного проєкту	05.05.2020
5.	Підготовка матеріалів другого розділу дипломного проєкту	07.05.2020
6.	Підготовка матеріалів третього розділу дипломного проєкту	10.05.2020
7.	Підготовка матеріалів четвертого розділу дипломного проєкту	14.05.2020
8.	Підготовка графічної частини дипломного проєкту	20.05.2020
9.	Оформлення документації дипломного проєкту	23.05.2020
10.	Попередній огляд матеріалів диплому на кафедрі	25.05.2020

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	A4		Завдання на дипломний проєкт	2	
2	A4	ІАЛЦ.045490.004 ПЗ	Пояснювальна записка	51	
3	A4	ІАЛЦ.045490.005 Д1	Програмні засоби тестування ботів месенджера Telegram. Схема структурна	1	
4	A4	ІАЛЦ.045490.006 Д2	Програмні засоби тестування ботів месенджера Telegram. Схема класів	1	
5	A4	ІАЛЦ.045490.007 Д3	Виконання тесту. Схема алгоритму	1	
6	A4	ІАЛЦ.045490.008 Д4	Завантаження плагіну. Схема алгоритму	1	
7		Презентація	Презентація за темою проєкту	1	
8		Диск CD-ROM	Текст ПЗ, тексти програм, графічний матеріал	1	

				ІАЛЦ.045490.003 ТП		
	ПІБ	Підп.	Дата			
Розробн.	Стадніченко А.В			Відомість дипломного проєкту	Лист	Листів
Керівн.	Дробязко І.П.				1	1
Консульт.					КПІ ім. Ігоря Сікорського Каф. СПіСКС Гр. КВ-62	
Н/контр.	Клятченко Я.М.					
Зав.каф.	Романкевич В.О.					

Пояснювальна записка до дипломного проєкту

на тему: Програмні засоби тестування ботів месенджера Telegram

Київ – 2020 року

ЗМІСТ

Перелік скорочень, умовних позначень, термінів	2
ВСТУП.....	3
1 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ТЕСТУВАННЯ БОТІВ TELEGRAM	5
1.1 Тестування. Різновиди тестування.....	5
1.2 Тестування у циклі розробки ПЗ	8
1.3 Аналіз існуючих рішень	10
1.4 Обґрунтування теми дипломного проєкту	14
2 АНАЛІЗ ТЕХНОЛОГІЙ І ПРОГРАМНИХ ЗАСОБІВ РОЗРОБКИ	15
2.1 Мови програмування Python та PHP	15
2.2 Архітектури проєктування програмного забезпечення	21
3 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ	27
3.1 Мова програмування	27
3.2 Комунікація з серверами Telegram.....	27
3.3 Взаємодія користувача з програмою.....	28
3.4 Робота з анотаціями	29

					ІАЛЦ.045490.004 ПЗ									
	Арк.	№ докум.	Підп.	Дата										
		Стадниченко			Програмні засоби тестування ботів месенджера Telegram <i>Пояснювальна записка</i>					Літ.	Аркуш	Аркушів		
Перевір.		Дробязко І.П.											1	51
Н. контр.		Клятченко Я.М.								НТУУ «КПІ ім. І. Сікорського»,ФПМ,КВ-62				
Затв.		Романкевич												

4 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ТЕСТУВАННЯ БОТІВ TELEGRAM	31
4.1 Архітектура проєкту	31
4.2 Опис файлів проєкту.....	33
4.3 Структура тестового файлу.....	35
4.4 Розширення базового функціоналу	39
4.5 Інтерфейс програми	42
5 НАЛАШТУВАННЯ ТА ЗАПУСК РОЗРОБЛЕНОГО ПЗ.....	46
ВИСНОВКИ.....	50
СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ.....	51

ДОДАТКИ

Додаток 1. Копії графічного матеріалу

ІАЛЦ.045490.005 Д1. Програмні засоби тестування ботів месенджера Telegram. Схема структурна

ІАЛЦ. 045490.006 Д2. Програмні засоби тестування ботів месенджера Telegram. Схема класів

ІАЛЦ. 045490.007 Д3. Виконання тесту. Схема алгоритму

ІАЛЦ. 045490.008 Д4. Завантаження плагіну. Схема алгоритму

Додаток 2. Фрагменти програмного коду

Додаток 3. Презентація

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

ПП – програмний продукт

ПЗ – програмне забезпечення

ТС – тестові сценарії

СКТ – системи керування тестуванням

JIT – JustInTime

ООП – Об’єктно Орієнтоване Програмування

OSI – Open Systems Interconnection

API – Application Programming Interface

PHP – PHP: HypertextPreprocessor або Personal Home Page tools

REST – Representational State Transfer

ВСТУП

У останні роки популярність месенджерів зросла у рази. Все більше людей відмовляються від електронної пошти та дзвінків через мережу GSM на користь їх. Одним із таких месенджерів є Telegram, що наразі потрапляє у топ 10 долі на ринку[1]. Одна з причин його популярності є широке та багатофункціональне API для створення чат-ботів. Чат-бот — віртуальний співбесідник, програма, створена для імітації поведінки людини при спілкуванні з користувачем. Такий віртуальний співбесідник дуже корисний у бізнес-середовищі, адже дозволяє замінити ним окремого співробітника або знизити їх кількість у задачах допомоги клієнтам, оформлення замовлень та інших.

Проте розробка таких систем, як і розробка будь-якого ПЗ вимагає тестування продукту. Для зменшення часу, що витрачається на тестування буде доречно використовувати автоматизоване тестування. Адже для такого тестування необхідно лише написати тест кейси, а їх виконання вже не буде вимагати втручання людини. Завданням дипломного проєкту є розробка програмного засобу, що дозволяє зручно писати тести для ботів на платформі Telegram, виконувати їх та переглядати результати виконання, що призведе до значного підвищення якості кінцевого продукту, адже усі виявлені недоліки будуть виправлені до релізу.

					ІАЛЦ.045490.004ПЗ	Арк.
Зм	Лист	№ докум.	Підп.	Дата		4

1 АНАЛІЗ ІСНУЮЧИХ ПРОГРАМНИХ ЗАСОБІВ ДЛЯ ТЕСТУВАННЯ БОТІВ TELEGRAM

1.1 Тестування. Різновиди тестування

Тестування програмного забезпечення – це процес дослідження та випробування програмного продукту(ПП), метою якого є перевірка між реальною поведінкою програми та очікуваною за допомогою кінцевого набору тестів, що будується згідно з технічним завданням. Метою тестування можуть бути:

- Бажання збільшити ймовірність того, що тестована програма матиме коректну працездатність за будь-яких обставин;
- Бажання переконатися, що тестована програма буде відповідати усім встановленим вимогам;
- Отримання актуальної інформації про стан програми в даний момент часу.

При тестуванні можуть бути виявлені дефекти (баги). Це коли отриманий піз час тестування результат не відповідає результатів, що очікується згідно тестовим сценарієм або документації продукту.

Тестування перших програмних систем, що були розроблені у 60-х роках відбувалося у якості окремого процесу, що починався одразу ж після завершення написання коду та проводилося строго формалізовано, з записом усіх тестових даних, процедур та отриманих результатів. Багато часу уділялося вичерпному тестуванню, під час якого повинні бути використані усі можливі варіанти вхідних даних або усі шляхи у коді. Але через те, що кількість таких даних та шляхів дуже велика, таке тестування було визнане теоретично неможливим. Далі на початку 70-х років тестування вже характеризувалося як діяльність, по підтвердженню коректності роботи ПП. Проте знову, такий

підхід був визнаний неефективним, оскільки на практиці був недостатньо всеохоплюючий та потребував значний час на його виконання. Натомість почала розвиватися інша концепція тестування, а саме виконання програми з метою знайти недоліки, а не довести її працездатність. Далі ця концепція почала розвиватися, розширюючись новими термінами і вже у 2000-х роках з'явилося широке визначення тестування, де основний підхід заключався у оцінці та максимізації важливості усіх етапів життєвого циклу розробки ПЗ для досягнення необхідного рівня якості, доступності та продуктивності.

У процесі розробки програмного забезпечення тестування займає значну частину часу, яка пропорційна обсягу функціоналу розроблюваного ПП. Через це тестування поділяється на велику кількість видів та методів тестування. До основних видів відносять:

- Функціональне тестування – перевірка можливості програмного забезпечення(ПЗ) реалізувати функціональні вимоги, тобто у певних умовах виконувати задачі, необхідні користувачам. Функціональні вимоги визначають що саме робить ПО та які задачі воно вирішує.
- Тестування продуктивності – вид тестування, що проводиться для визначення того, наскільки коректно система працює під певним навантаженням. Таке тестування дозволяє оцінити споживання ресурсів системою, її надійність та масштабованість. Такий вид можна поділити ще на 2 підкатегорії: стрес-тестування, коли навантаження на систему збільшується до її відмови, та навантажувальне тестування, коли система навантажувється згідно задокументованим вимогам;
- Тестування безпеки – перевірка того, наскільки система уразлива перед різними видами атак.
- Юзабіліті-тестування – перевірка ПО з метою визначити, наскільки зручним є інтерфейс для його ймовірного використання.

Також тестування, в залежності від того, чи є у розробника тестів доступ до коду програми, що буде тестуватися, можна поділити на такі категорії:

- Тестування білого ящика означає, що розробник тестів має доступ до коду програми та може писати код, який зв'язаний з бібліотеками тестуємого ПЗ;
- Тестування чорного ящика означає, що розробник тестів не має доступу до коду програми й може при тестуванні використовувати лише ті інтерфейси, які доступні замовнику або звичайному користувачу;
- Тестування сірого ящика означає, що хоча й розробник має доступ до коду, проте для тестування він зазвичай не потрібен.

У процесі тестування можна визначити три рівні тестування:

- Тестування компонентів – коли виконується тест для мінімально можливого компоненту, наприклад окремий клас, модуль, функція. Таке тестування зазвичай проводиться не тестувальниками, а безпосередньо розробниками ПЗ;
- Інтеграційне тестування – коли окремі програмні модулі об'єднуються у одну групу і тестуються разом;
- Системне тестування – тестування, мета якого це перевірка відповідності функціоналу розробленого продукту до початкових вимог.

Проте усі вищезазначені категорії можна віднести до однієї групи – динамічне тестування, оскільки у усіх випадках код буде виконуватися, різниця лише в кількості інформації, що доступна тестувальнику. При статичному ж тестуванні аналіз ПЗ відбувається на основі коду програми, аналіз якого відбувається або з використанням спеціальних інструментів, або вичитується вручну. Ще до статичного тестування відносяться тестування документації та вимог.

За ступенем автоматизації тестування можна поділити на ручне та автоматизоване. Ручне тестування виконується тестувальником без

використання додаткових програмних засобів для моделювання дій користувача. Такий тип буде актуальним для юзабіліті тестування, де у більшості ситуацій лише людина зможе побачити недолік. Наприклад, неінформативні дані на елементах інтерфейсу або незручний перехід між сторінками. Автоматизоване тестування ж використовує програмні засоби як для виконання тестів, так і для перевірки їх результатів. Такий підхід дозволяє значно скоротити час тестування та спростити його процес, адже написаний один раз тест можна виконувати повторно, прикладаючи мінімум зусиль.

При тестуванні зазвичай використовуються тестові сценарії(ТС) – описана спеціальним образом послідовність дій в заданій системі та очікованої реакції на проведені маніпуляції. Для зручності розробки ТС зазвичай використовуються спеціальні системи керування тестуванням(СКТ). Мета таких систем – оптимізація процесу тестування, шляхом надання інструментарію для написання тестової документації, тестових сценаріїв, відслідковування прогресу виконання тестування та роботи зі звітністю. При виконанні тестових сценаріїв тестувальник повинен відмічати проходження одного кроку або усього ТС та прикріплювати усю інформацію стосовно результатів проходження.

1.2 Тестування у циклі розробки ПЗ

У циклі розробки ПЗ можна виділити 2 основних підходи до тестування:

- Test-driven розробка (TDD);
- Test-last розробка (TLD);

При Test-driven розробці (розробка через тестування) розробка ведеться короткими циклами, а перед реалізацією функції створюється відмовний тест. Після створення тесту відбувається написання такого коду, який дозволить пройти цей тест, а після вдалого виконання тесту починається рефакторинг написаного коду для того, щоб він відповідав поставленим вимогам. Основним недоліком такого підходу є те, що на розробку витрачається значно більше часу ніж при TLD. Зв'язано це з тим, що під час

розробки розробник не може повністю сконцентруватись на чомусь одному і постійно переключається між написанням тестів та функцій. Серед переваг можна зазначити такі речі:

- Зменшуються витрати на обслуговування розробленого продукту, оскільки розроблене таким чином ПЗ має більшу надійність та стійкість через те, що кількість тестів більша ніж при TLD;
- При зміні коду є гарантія, що нові зміни не вплинуть не бажаним чином на кінцевий продукт. Зв'язано це з тим, що усі написані тести запускаються кожний раз повторно.

При Test-last розробці спочатку створюється функціонуюче рішення, а вже після реалізації відбувається процес тестування. Недоліками такого підходу можна вважати менш якісний код, оскільки при TDD кількість написаних тестів зазвичай вище. Звідси впливають більші витрати на підтримку готового продукту. Серед переваг такого підходу можна зазначити:

- Більшу швидкість розробки;
- Нижчий обсяг коду. При TLD розробці обсяг коду зазвичай менший, оскільки містить менше тест кейсів. Також такий код має простішу структуру;

Оскільки кожен з підходів має як свої переваги, так і недоліки, вибір слід робити в заєжності від характеристик розроблюваного ПЗ: часу, що був виділений на його розробку, тривалість підтримки та плани щодо подальшої кастомізації. TDD краще підходить для великих продуктів, для яких планується довга підтримка. TLD ж краще підійде коли часу на розробку продукту обмаль, або коли розробляється продукт невеликий.

1.3 Аналіз існуючих рішень

Всупереч популярності месенджеру telegram, програмних засобів автоматизованого тестування ботів обмаль. розглянемо найпопулярніші з бібліотек.

Бібліотека ptbtest

Розглянемо бібліотеку ptbtest[2]. Дана бібліотека створена мовою програмування python та використовується для виконання тестів для іншої бібліотеки – python-telegram-bot. Складається з 10-ти основних частин:

- Callbackquerygenerator - постачає клас для генерації запитів зворотнього виклику Telegram;
- Chatgenerator - постачає клас для створення екземплярів чату Telegram;
- Entityparser - постачає клас для перетворення стилізованих повідомлень у простий текст;
- Errors – містить виключення, що будуть виникати під час тестування;
- Inlinequerygenerator - постачає клас для генерації строкових запитів;
- Messagegenerator - постачає клас для генерації повідомлень;
- Mockbot – постачає клас для емуляції серверу Telegram;
- Ptbgenerator – містить базовий клас-генератор;
- Updategenerator – постачає декоратор для генерації об'єкту оновлень Telegram;
- Usergenerator – містить клас для генерації користувачів Telegram;

Особливістю цієї бібліотеки є те, що вона зв'язана з іншою бібліотекою - python-telegram-bot. А отже тестувати ботів, що були розроблені за допомогою інших бібліотек неможливо. Це значно обмежує область застосування ptbtest, проте є й перевага. Тестування може проводитися незалежно від серверів Telegram в офлайн режимі.

Тести пишуться в окремих файлах і являють собою нащадків абстрактного класу TestCase. Обов'язково треба ініціалізувати метод setUp, у якому треба налаштувати емулятор серверу Mosbot. Кожен файл являє собою один тест кейс. Половина файлу займає налаштування емулятора сервера, друга половина – безпосередньо тест кейс. Методи Mosbot дозволяють емулювати базовий функціонал, такий як: відправка тексту та медіаконтенту, редагування та пересилання повідомлень, отримання інформації про користувачів та операції з групами. Також є методи для отримання інформації з сервера, такої як об'єкт оновлення.

Консольний інтерфейс для перегляду списку тестів та інформації про них, запуску окремих тестів, тощо – відсутній. Запускати необхідно одразу файл з тестом через інтерпретатор python.

Бібліотека telegram-test-api

Друга розглядана бібліотека називається telegram-test-api[3]. Бібліотека розроблена мовою програмування JavaScript під фреймворк NodeJS. Дана бібліотека фактично являє собою емулятор серверу Telegram. Для написання ж тестів використовується інший JavaScript фреймворк – chai. Проте використовувати можна й інші бібліотеки для тестування. Бібліотека складається з 3-х основних модулів:

- requestLogger – модуль, що містить процедуру для зручного логування запитів серверу;
- sendResult – модуль, що містить процедуру для надсилання відповіді серверу;

- telegramClient – являє собою емулятор серверу та папки з маршрутами, що містять хендлери для реагування на запити, до емулятора.

Для проведення тестування з використанням цієї бібліотеки потрібно спочатку ініціалізувати та налаштувати сервер. Бібліотека підтримує встановлення таких параметрів, як: порт серверу, адресу, місце для зберігання логів та інших серверних даних, ідентифікатори й детальний опис користувача та чату. Після цього потрібно ініціалізувати бота. При ініціалізації вказується посилання на адресу емулятора та метод отримання оновлень від сервера.

Структурно тести можуть бути оформлені як у вигляді одного файлу з усіма тестами так і по одному тест кейсу на файл. Проте у кожному файлі буде необхідно ініціалізувати емулятор серверу заново. Також за бажанням є можливість налаштувати та використовувати власний інтерфейс для виконання тестів та виводу їх результатів.

Бібліотека telegram-test

Ще одним засобом для тестування ботів є бібліотека telegram-test[4]. Одразу слід зазначити її недолік. Вона як і ptbtest дозволяє тестувати ботів, що були написані з використанням специфічної бібліотеки. У цьому випадку: node-telegram-bot-api. Також дане рішення знову являє собою фактично емулятор серверу і підтримує лише обмежену автором кількість методів.

На відміну від telegram-test-api, дуже обмежений і складається лише з одного модуля:

- TelegramTest.

Його конструктор приймає посилання на бота, ідентифікатор повідомлення та номер оновлення. А його методи дозволяють створити об'єкт повідомлення та надіслати об'єкт оновлення. Тобто дане рішення найобмеженіше з усіх розглянутих і дозволяє оперувати лише текстовими повідомленнями. А значить

для інших тестів необхідно власноруч розшифрувати функціонал. Дане рішення також не має власного фреймворку для написання тестів.

Бібліотека tgintegration

Tgintegration - бібліотека з відкритим кодом, розроблена на мові Python. На відміну від усіх розглянутих вище рішень вона має суттєву відмінність. Вона базується на бібліотеці-клієнті Telegrampyrogram і виконує тестування у реальних сценаріях, замість емуляції серверу. Це по-перше знімає обмеження на тестуємих ботів. Адже тепер відсутня потреба у вихідному коді та обмеження на бібліотеки, за допомогою яких розроблявся бот. А по-друге при тестуванні ми матимемо реальну картину того, як взаємодіє бот з клієнтом.

Структурно дана бібліотека складається з 6 модулів:

- `__init__.py` – містить код для ініціалізації бібліотеки. Імпортує усі інші модулі та визначає необхідні для роботи константи;
- `AwaitableAction` – клас, що представляє собою дію, яка має бути надіслана на сервер під час очікування відповіді від нього;
- `Botcontroller` – клас, що представляє собою бота, тестування якого виконується та надає методи для взаємодії з ним;
- `InteractionClient` – клас-обгортка для бібліотеки pyrogram, що надає методи для взаємодії з серверами Telegram;
- `ResponseCollectorClient` – клас, що використовується для отримання відповіді від серверів;
- `Containers` – модуль, що містить класи-контейнери для представлення певних видів даних. На даний момент наявні 3 контейнери: `Response`, `InlineResults`, `Keyboard`.

Для проведення тестування дана бібліотека використовує `pytest`. Також у планах розробників була інтеграція з `unittest`. Проте через низьку популярність розробка була припинена.

1.4 Обґрунтування теми дипломного проєкту

На сьогодні боти для месенжера Telegram доволі розповсюджені та використовуються багатьма відомими компаніями для популяризації власних сервісів та/або полегшення їх використання. Проте всупереч популярності ботів, сервісів, що дозволяють їх тестувати обмаль. І найпопулярніші рішення зазвичай мають декілька недоліків. По-перше, вони прив'язані до якоїсь конкретної бібліотеки для створення ботів. А значить якщо ваше рішення створене за допомогою іншого інструментарію, то доведеться шукати інші способи для написання та виконання тестів. По-друге, ці рішення являюся собою емулятори серверу, а значить не зображають дійсності при взаємодії користувача і бота. Навіть при найреалістичнішій реалізації емулятора, ми ніколи не знаємо як поводитися сервер у певній ситуації. Адже у користувачів відсутній доступ до коду сервера Telegram. І по-третє, дані рішення не мають достатніх інтерфейсів для розширення власного функціоналу, а отже якщо знадобиться додати певні додаткові можливості, то доведеться вносити значні зміни у код програми.

Тому метою розроблюваної системи буде урахування саме цих недоліків і реалізація зручного рішення, яке буде необхідно налаштувати лише один раз при першому запуску. Розроблюваний програмний продукт повинен відповідати наступним функціональним вимогам:

- Використання серверу Telegram замість емуляції;
- Підтримка тестування будь-яких ботів, незалежно від їх інструментарію;
- Зручний інтерфейс для виконання тестів та перегляду інформації про них.

2 АНАЛІЗ ТЕХНОЛОГІЙ І ПРОГРАМНИХ ЗАСОБІВ РОЗРОБКИ

Ботів для Telegram можна розробляти будь-якою мовою програмування, що здатна реалізовувати інтернет-зв'язок за протоколами HTTP та HTTPS. Серед основних мов, що використовують для розробки додатків можна відзначити: Java, C#, PHP, JavaScript, GoLang, Python. Їх популярність обумовлена широким інструментарієм для налагоджування з'єднання у мережі інтернет, реалізації сокетів, отримання/обробки/відправки даних за протоколом HTTP, реалізації REST API та інших речей. Проаналізуємо та розглянемо технології PHP та Python, так дані технології мають найбільшу популярність серед розроблених ботів для Telegram.

2.1 Мови програмування Python та PHP

Python та PHP доволі популярні мови програмування у середовищі Back-end розробки, про це PHP має у рази більшу популярність. У цей момент, 78.3% усіх веб сайтів розроблені мовою програмування PHP[6]. Зумовлено це тим, що з самого початку PHP позиціонувалася як інструмент для розробки веб-додатків. Перша версія PHP була розроблена у 1995-му році як застосунок на мові Perl для обробки даних на веб сайті її розробника. Після того, як цей застосунок почало використовувати все більше людей, автор назвав його Personal Home Page Tools (PHP) версії 1 і виставив для вільного завантаження. Після цього почався її стрімкий розвиток і для неї було розроблено багато бібліотек та фреймворків. Стандартна бібліотека також була збільшена у рази. Python же у свою чергу розвивався більше як мова загального призначення і тому web-розробка є лише незначною частиною його застосування.

PHP

PHP (PHP: Hypertext Preprocessor) - скриптова мова програмування загального призначення. Створювалася у першу чергу для створення веб застосунків, а отже стандартна бібліотека включає в себе велику кількість інтерфейсів для створення веб додатків. Наразі PHP має лише один актуальний інтерпретатор: Zend Engine, написаний мовою програмування C.

Переваги PHP:

- З 7-ї версії найшвидша скриптова мова програмування [5];
- Базова бібліотека містить дуже велику кількість функцій для різноманітних потреб;
- Велика кількість додаткових бібліотек і фреймворків, зумовлена популярністю PHP;
- Механізм автозавантаження класів;
- Підтримка усіх основних парадигм програмування;
- Низький поріг входу.

Недоліки PHP:

- Відсутня стандартизація назв базових функцій (одні можуть бути написані кемел кейсом, а інші паскал кейсом)
- Низька якість коду деяких сторонніх бібліотек
- Відсутня підтримка асинхронності
- Відсутня підтримка багатопотоковості за замовчанням

Python

Python - високорівнева мова програмування загального призначення. Його основна особливість — унікальний синтаксис, націлений на підвищення

читабельності кода та зручності його написання. Канонічна реалізація інтерпретатора має назву PyPy, написана мовою C і використовує JIT-компіляцію. Такий підхід значно підвищує швидкість виконання python-скриптів.

Переваги Python:

- Висока читабельність коду, написаного на python;
- Підтримка асинхронності та багато потоковості;
- JIT-компіляція;
- Дуже багата бібліотека, що має застосунки не тільки для веб-додатків, а й створення графічних інтерфейсів, математичних обчислень, тощо;
- Підтримка перевантаження операторів.

Недоліки Python:

- Низька продуктивність у порівнянні з мовами з статичною типізацією;
- Неповна реалізація ООП (відсутність приватних методів та полів, тощо).

Спільні риси PHP та Python

Ці мови програмування мають як багато спільного, так і багато відмінностей. Код обох мов виконується в інтерпретаторі замість компіляції у машинний код. Також обидві мови мають динамічну типізацію та збирач сміття. Проте слід зазначити, що в останніх версіях PHP з'явилася можливість вказувати типи даних для полів класу, операндів функцій та даних, які повертає функція. Це підвищує сприйняття коду та трохи прискорює роботу інтерпретатора. Обидві мови мають доволі обширну стандартну бібліотеку з функціями для роботи зі строками, масивами, інтернет-з'єднаннями, сокетом, базами даних, математичними виразами та іншими необхідними речами. Проте знову можна виділити відмінність. Стандартна бібліотека Python реалізована у парадигмі ООП і такі сутності, як строка чи сокет є об'єктами. І для маніпуляції над ними необхідно викликати методи через оператор «.». У PHP ж до 3-ї версії взагалі не було ООП, тому функції базової бібліотеки викликають не як методи,

а як окремі функції, передаючи посилання на об'єкт, що необхідно змінити, параметром. Як приклад, розглянемо послідовність дій, необхідну для переведення строки у верхній регістр та видалення пробільних символів.

Мовою Python: `'Some string'.upper().strip()`

Мовою PHP: `trim(strtoupper('Some string'))`

Багатопотоковість: ще одна з відмінностей — у PHP відсутня підтримка засобів для асинхронного програмування та багатопотоковість (проте можна завантажити спеціальну версію з підтримкою бібліотеки C pthreads). Python же має у стандартній бібліотеці такі модулі, як `multithreading`, `multiprocessing` та `concurrent`. Це дозволяє отримати значну перевагу при розв'язанні задач, які можна розпаралелити.

Автозавантаження класів: на відміну від Python, PHP має механізм автозавантаження класів. Зазвичай більшість ООП-проектів мають чітке узгодження щодо назв класів та їх розташування. Механізм автозавантаження дозволяє замість імпорту кожного файлу з класом окремо, завантажувати їх автоматично, керуючись певною логікою, яка була попередньо встановлена функцією `spl_autoload_register()`. Даний механізм працює наступним чином: при зверненні до класу чи його властивостей, інтерпретатор перевіряє його наявність у поточному просторі імен. Далі, якщо клас відсутній, перевіряється чи була визначена функція для автозавантаження і при позитивному результаті функція викликається з параметром, який містить повне ім'я класу, включаючи його простір імен. Якщо така функція відсутня, то інтерпретатор перериває роботу помилкою про відсутність необхідного класу. Цей механізм використовує менеджер пакетів `Composer` для підключення сторонніх бібліотек.

Синтаксис: основна ж відмінність між цими мовами — синтаксис Python. Якщо PHP має звичний C-подібний синтаксис, то Python використовує власний, чіткий і послідовний синтаксис. Замість крапок з комою та фігурних дужок використовуються пробіли, переноси й табуляції. Такий підхід робить код надзвичайно читабельним, адже, по-перше: зайві символи не відволікають уваги, а по-друге, оскільки пробільні символи частини синтаксичних

конструкцій, то код чітко відформатований і відсутні ситуації, коли в одному місці уся функція записана в одну строку, а в іншому — розділена на окремі строки.

Розробники Python дотримуються спеціальної філософії[7]:

- Красиве краще, ніж потворне;
- Явне краще, ніж неявне;
- Просте краще, ніж складне;
- Складне краще, ніж заплутане;
- Читабельність має значення;
- Особливі випадки не настільки особливі, щоб порушувати правила;
- Практичність важливіша за бездоганність;
- Помилки ніколи не повинні замовчуватися, окрім випадків, коли це робиться явно;
- Зараз краще аніж ніколи;
- Хоча часто ніколи, краще, аніж прямо у даний момент;
- Зустрівши неоднозначність, не намагайся вгадувати;
- Повинен існувати лише один очевидний спосіб щось зробити.

Розробка: обидві мови програмування мульти-платформені та відкриті. Канонічні інтерпретатори розроблені мовою C, проте є й альтернативні реалізації на таких мовах програмування, як Java, C++, C#. Для мови PHP слід відзначити HipHop for PHP, транслятор розроблений компанією Facebook для перетворення вихідного кода PHP в оптимізований код на мові C++. У часи, коли була актуальна PHP версії 5, цей транслятор працював значно швидше за еталонний Zend Engine і такі великі проєкти, як Facebook та Wikipedia використовували саме його. Проте після виходу PHP 7, Zend та HH зрівнялися по швидкості, тому зараз HH доречно застосовувати лише для підтримки старих проєктів.

Через відкритість мов, існує великий набір інструментів, застосунків та продуктів, що доступні для використання в проєктах. Велика частина

невеликих OpenSource проєктів знаходить застосування у комерційних продуктах і розвивається там. Для написання коду є як повноцінні IDE, такі як Pycharm або PHPStorm, так і продвинуті текстові редактори як Sublime texteditor або Visual Code.

Засоби для комунікації з серверами Telegram: для зв'язку та обміном даних з Telegram існує офіційна бібліотека TDLib. Дана бібліотека бере на себе реалізацію низькорівневої мережевої частини, локального сховища та узгодженості даних. Вона має відкритий код та є крос-платформенною, що дозволяє її використовувати з усіма актуальними мовами програмування. Своєю чергою існують бібліотеки з ще більшим рівнем абстракції, які використовують TDLib за власну основу. Такі бібліотеки беруть на себе аутентифікацію клієнта, реалізують спрощені методи для надсилання та отримання даних, містять класи для представлення повідомлень, оновлень та користувачів. Найбільш функціональні подібні бібліотеки це Pyrogramm для Python та MadelineProto для PHP. Обидві бібліотеки підтримують аутентифікацію клієнта у консолі та подальше кешування даних авторизації, отримання та відправку повідомлень з будь-яким змістом. Pyrogramm існує довше і має ширший набір методів для різних ситуацій, в той час як MadelineProto має лише базові методи для відправки та отримання повідомлень, а оперувати з параметрами під час цього доведеться власноруч.

Відладка розробленого ПЗ: Оскільки обидві мови програмування досить популярні, для них наявна велика кількість бібліотек для тестування та відладки зі своїми особливостями. розроблених програм. Найбільш популярні бібліотеки для тестування ПЗ, розробленого для PHP:

- PHPUnit – найвідоміша бібліотека для написання юніт-тестів. Вона дозволяє керувати процесом тестування через консоль та надає зручні інтерфейси для тестування програм за власними сценаріями;
- Codeception – навідміну від PHPUnit дозволяє виконувати не тільки юніт-тестування, а й функціональне та приймальне тестування.

Перевагою цієї бібліотеки є те, що вона інтегрована з більшістю

популярних фреймворків, що застосовуються для розробки веб застосунків, серед них: Symfony, Laravel, Yii, Zend та інші;

- Behat – бібліотека для тестування при розробці, керованій поведінкою. Тести, що були написані з використанням цієї бібліотеки більше схожі на сценарії, ніж як звичайник код.

Для тестування коду на мові Python найбільш відомі бібліотеки:

- Robot – бібліотека, що використовується для розробки, керованої тестами та приймального тестування. Через значну кількість API має великий потенціал до розширення.
- PyTest – бібліотека, що може використовуватися для будь-якого з видів тестування, проте найкраще підходить для функціонального тестування та тестування API. Має простий для розуміння синтаксис, велику кількість плагінів та можливість виконання тестів паралельно;
- Unittest – найперша бібліотека для виконання юніт-тестування, написана на мові Python.

В результаті вищенаведеного аналізу, для розробки проєкту була обрана мова програмування PHP і бібліотека MadelineProto.

2.2 Архітектури проєктування програмного забезпечення

Багаторівнева архітектура

Одна з найрозповсюдженіших архітектур. Багато великих бібліотек та фреймворків побудовано на її основі. А найбільш відомим прикладом її використання є мережева модель Open Systems Interconnection (OSI).

Система поділяється на рівні, кожен який може взаємодіяти лише з двомасусідніми. Тому запити до даних знаходяться у самому кінці ланцюга взаємодії й проходять послідовно через кожен рівень. Така архітектура

не має вимог до кількості рівнів, проте найбільш розповсюджена система з 3-х рівнів (Рис 2.1): рівень представлення, логіки та керування даними.



Рисунок 2.1 – Типова трирівнева архітектура

Переваги: у даній архітектурі рівні не залежать від своїх сусідів і не знають, як вони влаштовані. А отже можна модифікувати будь-який з рівнів, не викликаючи конфлікти у роботі інших.

Недоліки: при великій кількості рівнів робота програми може бути сповільнена, оскільки раз за разом інформація послідовно проходить кожен з рівнем, виконуючи у деяких випадках заяву роботу. Також при виникненні помилки її пошук може бути затрудненим. Адже якщо у базу даних потрапили пошкоджені данні, то для знаходження помилки доведеться розглянути кожен з рівнів окремо.

Найбільш часто застосовується в:

- Програми, які вимагають швидкої розробки;
- Корпоративні системи, де зазвичай все розділено на команди, кожна з яких займається своїм рівнем;

- Через свою простоту - при розробці ПЗ командами розробників, що не мають великого досвіду у розробці;
- Розробка ПЗ, що має чіткі вимоги до тестування та налагоджування.

Мікроядерна архітектура

Дана архітектура складається з двох компонентів: ядра системи та плагінів. Плагіни відповідають за бізнес-логіку, у той час як ядро керує їх завантаженням. Гарним прикладом мікроядерної архітектури є текстові редактори. Їх базовий функціонал дозволяє відкривати файли та змінювати їх. Проте різноманітні плагіни дозволяють значно розширити цей функціонал.

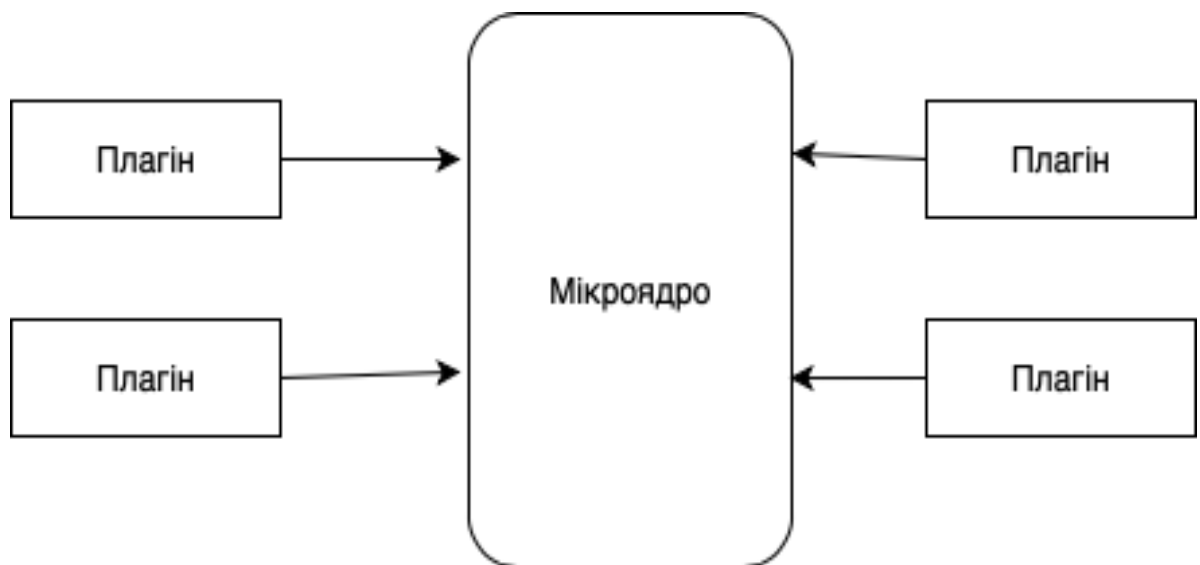


Рисунок 2.2 – Типова мікроядерна архітектура

Переваги: програму легко портувати з одного середовища в інше, оскільки модифікувати треба буде лише мікроядро.

Недоліки: чим більше до системи підключається плагінів — тим повільніше починає працювати система. А знайти баланс кількістю задач мікроядра та кількістю плагінів не завжди просто.

Найбільш часто застосовується в:

- Створення програмних засобів, які будуть з часом розширюватися;
- Створення програмних засобів з чітким розділенням базових методів та високорівневих правил;
- Розробка систем з динамічно змінюваним набором правил.

Фільтри та канали

Дана архітектура складається з фільтрів, які займаються обробкою даних, та каналів, по яким проходять дані та які об'єднують між собою фільтри (Рис 2.3) Фільтр отримує дані із вхідного каналу, та повертає оброблену інформацію у вихідний канал. Фільтри можна замінювати на інші, використовувати по декілька разів, переставляти місцями між собою. І при цьому можуть працювати паралельно, що призводить до значного підвищення швидкодії ПЗ.

Переваги: дана архітектура краще за інші підходить для розпаралелювання задач.

Недоліки: розробка програм з використанням цього архітектурного шаблону має підвищену складність, адже окрім типових проблем, що виникають під час багато потокових програм слід також уділяти увагу інфраструктурі. Вона повина забезпечувати надійну передачу даних між фільтрами. Також збій у одному фільтрі може призвести до того, що наступний отримає два однакових повідомлення.

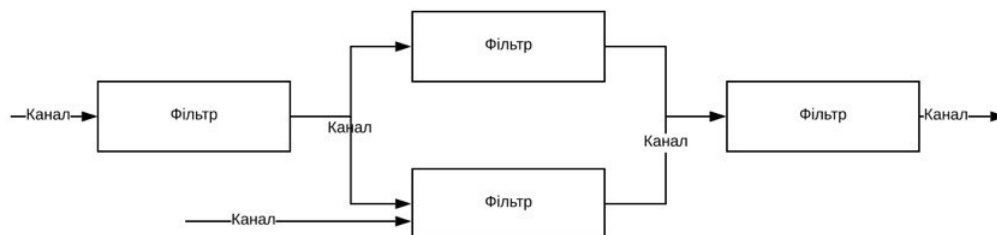


Рисунок 2.3 – Зображення архітектури каналів та фільтрів

Найбільш часто застосовується в:

- Розробці програм, що потребують розділення процесу обробки на ряд незалежних етапів;
- Ситуаціях, коли необхідні мінімальні наслідки при виникненні перешкод на одному з етапів;
- Системи, які розділені на декілька серверів.

Керуєма подіями архітектура

При використанні даної архітектури розробник прописує для програми реакції на події, що виникають під час її роботи. Подія — суттєва зміна стану програми. Така система зазвичай містить два компоненти: джерело події та їх споживачі (Рис 2.4).

Переваги: подібні системи дуже легко масштабувати, оскільки вони складаються з великої кількості асинхронних модулів, які не мають інформації про реалізації друг друга. Також використання асинхронної моделі дозволяє підвищити продуктивність.

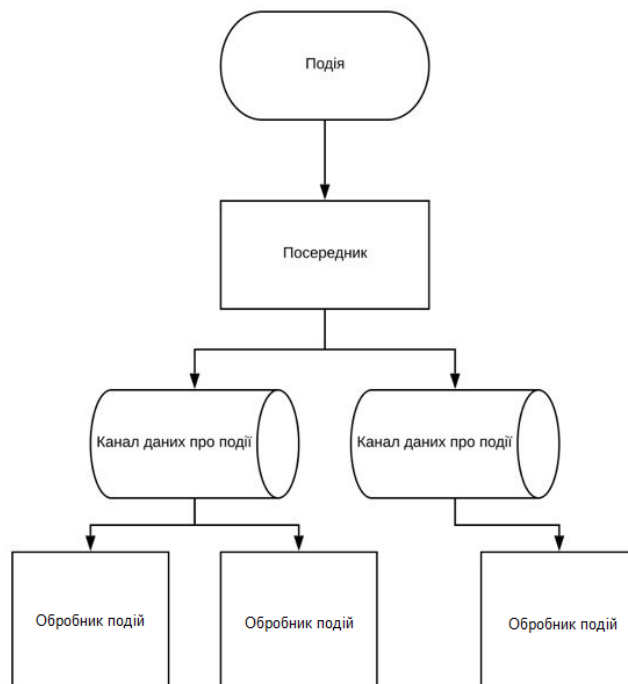


Рисунок 2.4 – Типова архітектура, керуєма подіями

Недоліки: відладка може бути ускладнена через асинхронну модель. Адже одна подія може запустити послідовність інших подій. І якщо таких послідовностей досить багато, то зрозуміти де саме виникнув збій буде складно.

Найбільш часто застосовується в:

- Створення асинхронних систем;
- Розробка інтерфейсів для програмних продуктів;
- Обмін повідомленнями між різними інформаційними системами.

3 ВИБІР ЗАСОБІВ РЕАЛІЗАЦІЇ

У якості засобів реалізації було обрано наступні інструменти та бібліотеки, опираючись на проведений вище аналіз:

3.1 Мова програмування

У якості мови програмування для розробки була обрана PHP версії 7.4. Попри певні її недоліки, вона найкраще підходить для розробки веб-орієнтованих застосунків через її обширну стандартну бібліотеку. А такі механізми, як автозавантаження класів та підтримка анотацій дозволять користувачу зручно модифікувати це ПЗ.

3.2 Комунікація з серверами Telegram

Бібліотека для зв'язку з серверами Telegram: для комунікації з серверами Telegram була обрана бібліотека MadelineProto. MadelineProto - Telegram-клієнт написаний мовою PHP, що надає методи для роботи як від імені користувача, так і від імені бота Це фактично єдиний Telegram-клієнт, розроблений на мові PHP. Серед його функціоналу є:

- Кешування даних авторизації на комп'ютері, що дозволяє не входити до власного аккаунту кожен раз перед запуском тестів;
- Реалізації асинхронності через використання генераторів PHP;
- Підтримка телефонних викликів;
- Легкі у використанні обгортки, для завантаження файлів з/на сервер та виклику методів mtproto;
- Підтримка проксі;
- Широкі можливості для кастомізації клієнта.

Не зважаючи, на менший функціонал у порівнянні з аналогом, на мові python, pyrogram, бібліотека забезпечує увесь необхідний нам функціонал, а саме відправка та отримання даних з сервера. При необхідності додатковий функціонал можна буде реалізувати власноруч. Базові методи дозволяють робити такі речі:

- Вхід та вихід до системи під заданим іменем користувача;
- Зміна паролю;
- Отримання повідомлень з будь-якого чату;
- Відправка повідомлень у будь-який чат;
- Отримання будь-якої інформації про чат або групу та її користувачів;
- Створення секретних чатів;
- Виконання дзвінків.

3.3 Взаємодія користувача з програмою

Взаємодія з користувачем буде реалізована через консоль. Такий підхід дозволяє знизити системні вимоги ПЗ, а також дозволить зробити його легкодоступним у будь-якій точці системи. Консоль повинна забезпечувати наступні функції:

- Виводити список усіх доступних тестів;
- Виводити інформацію про окремий файл з тестом(ім'я, призначення, перелік тест кейсів, ім'я та опис кожного з тест кейсів);
- Запускати окремий тест по імені;
- Запускати усі доступні тести.

Для реалізації консольного інтерфейсу була обрана бібліотека WebmozartConsole. Дана бібліотека дозволяє створювати зручні та багатофункціональні консольні інтерфейси без написання зайвого коду. Для розробленого за допомогою бібліотеки інтерфейсу доступний наступний функціонал:

- Підтримка вибору кольору для тексту;
- Вивід інформації у вигляді таблиці;
- Стилiзація тексту (жирний/курсив/підкреслений);
- Підтримка параметрів для команд;
- Підтримка під команд;
- Можливість використання допоміжних засобів Symfony Console.

Також дана бібліотека надає можливість тестування розроблених за її допомогою інтерфейсів. Для цього у ній присутній окремий модуль.

3.4 Робота з анотаціями

Для підвищення інформативності файлів з тест кейсами та окремих тест кейсів, використовується механізм анотацій РНР. Анотації дозволяють встановити назву для тест кейсу та опис того, що він робить. (Рис 3.1) Ця інформація у свою чергу, по-перше: дозволяє при модифікації файлу з тест кейсами бачити, яка саме мета його роботи, що корисно, якщо над написанням тестів працює декілька людей. По-друге: під час виконання тесту, бачити у консолі, якою є мета виконаних дій. Також анотації використовуються для помічення у файлі, які саме функції представляють тест кейс. Така сегментація дозволяє розміщати у файлі окрім тест кейсів ще й допоміжні функції.

```
/**
 * @title Base behavior
 * @description testing base bot behavior
 */
class TestBaseBehavior extends Test {

    /**
     *
     * @testcase
     * @title Test help command
     * @description Test correctness of reply on command '/help'
     */
    public function testHelp()
    {
```

Рисунок 3.1 – Анотації у файлі з тест кейсами

Для роботи з анотаціями була обрана бібліотека MinimeAnnotations. Дана бібліотека була розроблена ще у 2013-му році й підтримується до сих пір. Її функціонал реалізується за допомогою регулярних виразів та механізму рефлексії PHP (ReflectionClass). При виклику метода getClassAnnotation() створюється екземпляр класу ReflectionClass, який містить у собі повну інформацію про клас, що був у параметрі. Далі через метод getDocComment() бібліотека отримує текст анотацій. І вже після цього, за допомогою регулярних виразів, відбувається синтаксичний аналіз змісту поля з анотаціями, після якого ми можемо отримати зміст будь-якої анотації. Проте ReflectionClass не може працювати з константами, тому для роботи з їх анотаціями реалізує власний клас ReflectionConstant. Даний клас використовує сторонній парсер PHP коду (nikic/php-parser) для знаходження та зчитування анотацій констант у заданих класів.

4 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ТЕСТУВАННЯБОТІВ TELEGRAM

4.1 Архітектура проєкту

Для розробки системи була обрана мікроядерна архітектура (Рис 4.1). Такий підхід був обраний у зв'язку з тим, що для виконання тестів буде виконуватися абсолютно однакова послідовність дій. Тому ця логіка буде розташована у ядрі розроблюваного ПЗ. А такі речі, як функції, для відправки даних на сервери Telegram, обробки даних для визначення успіху чи невдачі виконання тесту будуть реалізовані як окремі плагіни. Також, даний підхід забезпечить можливість значно розширювати можливості ПЗ, при цьому не переживаючи, що якась помилка у плагіні призведе до пошкодження працездатності усієї системи. Також таким чином реалізується можливість розширення базового функціоналу модулями, написаними користувачем.

Структурно проєкт буде складатися з 4-х основних розділів:

- Ядро, яке буде реагувати на взаємодію користувача з системою, підключати плагіни та виконувати тести;
- Простір імен з плагінами, що розширює можливості клієнту Telegram;
- Простір імен з плагінами, що розширюють можливості тестування;
- Директорія з тестами, які доступні для виконання.

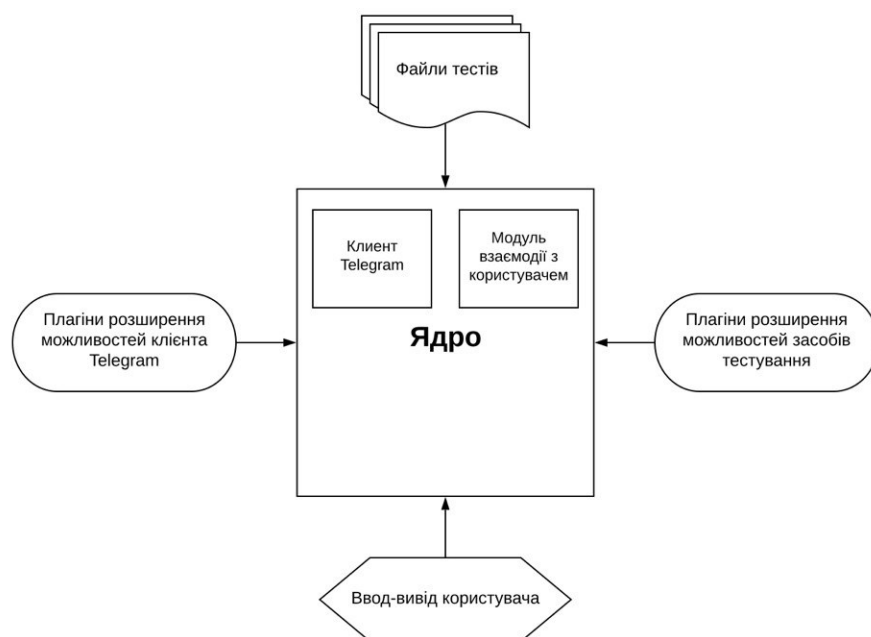


Рисунок 4.1 – Графічне зображення архітектури проєкту

Для зручності усі розділи розташовані у власному просторі імен:

- Ядро – Extravert (кодова назва розроблюваного проєкту);
- Плагіни клієнта Telegram – Plugins\Telegram;
- Плагіни для розширення можливих варіантів тестування – Plugins\Assertion.

Тести не мають власного простору імен, натомість усі доступні тести мають бути розташовані у папці tests, що лежить в корені ПЗ.

Запускається програма через файл ext, який одразу ж звертається до ядра, щоб встановити поточне розташування проєкту та ініціалізувати консольний інтерфейс для взаємодії з користувачем. Далі через цей інтерфейс користувач за допомогою текстових команд взаємодіє з ядром, запускаючи необхідний йому тест, або виводячи на екран інформацію про нього.

4.2 Опис файлів проєкту

Розроблюване програмне забезпечення у базовому варіанті складається з 13 класів, з них 6 належать інтерфейсу програми та 2 до плагінів.

Призначення класів:

- **Kernel** – ядро програми. Надає методи для виконання тестів та встановлення певних параметрів. Реалізує підключення тестів з директорії tests;
- **Result** – клас-обгортка для представлення результатів перевірки твердження на дійсність. Дані, що містяться у класі, виводяться у консоль;
- **AbstractAssertion** – абстрактний клас, що мусить бути батьківським для усіх класів-плагінів типу Plugins\Assertion. Містить методи для легкої та швидкої побудови плагінів;
- **AbstractTelegramAction** – абстрактний клас, що мусить бути батьківським для усіх класів-плагінів типу Plugins\Telegram. Надає доступ розроблюваним плагінам до активного клієнту MadelineProto а також до екземпляру тесту, у якому команда викликається;
- **Test** – абстрактний клас, що мусить бути батьківським для усіх тестів. Містить методи, за допомогою яких ядро реалізує виконання тестів. Також реалізує метод expect, за допомогою якого перевіряється істинність заданих тверджень;
- **Assertion** – клас для ініціалізації та подальшої роботи з твердженнями, які були породженні викликом методу expect класу Test. Також реалізує механізм завантаження плагінів типу Plugins\Assertion;
- **Expectation** – клас-обгортка для об'єктів типу Assertion. У ньому встановлюється очікуваний тип твердження (toBe, toNotBe);
- **TestingConsts** – клас, що містить у собі усі константи, які використовуються у вихідному коді проєкту;

- `ConsoleApp` – клас, у якому ініціалізується консольний інтерфейс для взаємодії користувача з ПЗ;
- `ConsoleConfig` – клас, що містить параметри консольного інтерфейсу. Серед них перелік доступних команд та їх опис;
- `Run` – клас, що реалізує консольну команду для виконання тесту. Приймає один параметр – ім'я тесту, який необхідно виконати;
- `TestInfo` – клас, що реалізує консольну команду для виводу інформації про тестовий файл на екран. Приймає один параметр – ім'я тесту, інформацію про який необхідно вивести;
- `TestList` – клас, що реалізує консольну команду для виводу списку усіх доступних для виконання тестів на екран.

Також окрім класів слід зазначити про 2 окремих файла `config.php` і `ext`. Перший файл містить інформацію, необхідну для налагодження з'єднання з серверами Telegram. Серед них:

- `api_id` – унікальний ідентифікатор, необхідний для дозволу комунікацій між сервером та клієнтом;
- `api_hash` – хеш унікального ідентифікатора;
- `device_model` – ім'я, під яким дане ПЗ буде працювати з серверами;
- `logger` – вказання засобу, для реалізації логування подій, що відбуваються при обміні даними з Telegram;
- `param` – параметр, що отримує `logger` при ініціалізації (нп ім'я файлу, у який буде записуватися інформація);
- `testing_target` – ідентифікатор бота, тестування якого буде виконуватися.

Другий файл `ext` – вхідна точка програми, що передає ввід користувача до ядра.

Тести, які необхідно виконати, мають знаходитися в кореневій директорії у папці `tests`. Програма автоматично сканує директорію та виявляє в ній тести. Перед початком тесту викликається метод ядра `init`, у якому ініціалізується клієнт Telegram. При вдалому входу до акаунту викликається наступний метод

runTest. У методі runTest відбувається перевірка на існування файлу тесту з переданим у параметрі іменем. Якщо файл існує, ініціалізується клас тесту, якщо ні – виводиться повідомлення про помилку.

Після вдалої ініціалізації тесту викликається метод env, у якому повинні бути описані команди для встановлення певних початкових налаштувань бота перед початком тестування. Наприклад, вибір робочої мови бота. Потім у циклі-генераторі починається виклик усіх методів, помічених як тест кейс відповідною анотацією. Після виконання кожного тест кейсу генератор повертає масив з результатами. Ці результати своєю чергою виводяться у консоль. Після проходження усіх тест кейсів програма завершає свою роботу.

4.3 Структура тестового файлу

Основною вимогою до тестових файлів є те, що вони мають бути нащадками класу Test. Адже даний клас містить методи, які необхідні ядру ПЗ для виконання тестів. Ці методи приховані від користувача і доступні лише ядру. Серед цих методів:

- Метод runTestCase() – запускає виконання тестового файлу, зберігаючи отримані результати;
- Метод logExpectationResult() – записує результат перевірки твердження на істинність до результатів;

Також є методи, доступні для використання у власних тестах, серед них:

- Метод log() – дозволяє заносити власну інформацію до результатів виконання тестів.
- Метод expect() – ініціалізує вираз для його подальшої перевірки на істинність;
- Абстрактний метод env() – має бути перевизначеним у кожному тест кейсі. У ньому повинна відбуватися підготовка бота для подальшого

тестування. Наприклад, встановлення мови відповідей для бота, або скидання попередніх налаштувань.

Аналізуючи рішення зі схожою метою роботи, можна побачити наступну структуру тестів:

Є базовий певний базовий клас або метод, один на весь файл(контейнер для тест кейсів). У ньому знаходяться тест кейси для певної структурної одиниці тестуемого ПЗ (наприклад функціонал меню налаштувань). Далі, у ньому знаходиться перелік тест кейсів для перевірки більш конкретних моментів, наприклад чи є анімація при закритті меню, чи увесь текст відображається коректно. І вже у тілі цих тест-кейсів знаходиться послідовність дій для створення тестуємої ситуації та перевірки її на істинність.

У розробленого ПЗ для тестування ботів Telegram буде використовуватися аналогічна структура. У ролі бази виступає окремий, створений користувачем клас, що наслідує Test. Методи цього класу, помічені анотацією @testcase являються тест кейсами. Обов'язкова вимога до методів тест кейсів – модифікатор доступу public. Для того, щоб ядро могло запускати їх. А для перевірки тверджень на істинність використовується метод expect(). Також у тестах має бути визначений метод env(). Мета цього методу – підготовка бота до майбутнього тестування. На рисунку 4.2 можна побачити, як у цьому методі встановлюється мови, з якими буде працювати бот.

```

* @title Base behavior
* @description testing base bot behavior
*/
class TestBaseBehavior extends Test {

    public function env()
    {
        $this->setMyLangRussian();
        Telegram::command('tolang');
        Telegram::plainResponse();
        Telegram::sendText('English');
    }

    private function setMyLangRussian()
    {
        Telegram::command('setmylang');
        Telegram::plainResponse();
        Telegram::sendText('Русский');
    }
}

/**
 *
 * @testcase
 * @title Test help command
 * @description Test correctness of reply on command '/help'
 */
public function testHelp()
{
    Telegram::command('help');
    $this->expect(Telegram::plainResponse()->toBe()
        ->equals('Чтобы указать направление перевода просто наберите /tolang и укажите язык из списка языков'));
}

```

Рисунок 4.2 – Анотації у файлі з тест кейсами

Методи, що не позначені як тест кейс, вважаються допоміжними й можуть викликатися у тілі тест кейсів.

Також підтримуються анотації: @title, @description як для методів, так і для самого класу, @testcase – для методів класу, що являються тест кейсами.

Анотація @title задає заголовок, який буде показуватися у консолі, під час виводу інформації про тест, або під час його виконання. Анотація @description задає опис, який також буде виводитися для користувача. Такі анотації надзвичайно корисні, адже при виконанні файлу або його редагуванню завжди можна визначити задачу, яку виконує той чи інший тест кейс. У більшості рішень для тестування такий опис якщо і задається, то серед коду, як параметр функції під час ініціалізації тест кейсу. Такий підхід при виконанні тестів справляється зі своєю задачею і також виводить інформацію на екран. Проте при перегляді власне файлів з тестами такий варіант оформлення менш читабельний, оскільки опис і сам код тесту зливаються один з одним. Приклад такого можна побачити на Рис 4.3

```
describe('My First Test', () => {
  it('Does not do much!', () => {
    expect(true).to.equal(true)
  })
})
```

Рисунок 4.3 – Типове оформлення опису тестів

Для роботи з серверами Telegram у тестах бажано використовувати існуючий клас Extravert\Client\Telegram. Адже перед запуском тестів він вже автоматично ініціалізується, входячи під заданим користувачем та встановлюючи ідентифікатор боту, тестування якого буде проводитися. Проте жодних обмежень немає й у тестах можна використовувати будь-які власні бібліотеки, які підключені через Composer.

Для роботи з серверами Telegram у тестах бажано використовувати існуючий клас Extravert\Client\Telegram. Адже перед запуском тестів він вже автоматично ініціалізується, входячи під заданим користувачем та встановлюючи ідентифікатор боту, тестування якого буде проводитися. Проте жодних обмежень немає й у тестах можна використовувати будь-які власні бібліотеки, які підключені через Composer.

- toBe – результат порівняння значення, що було передане в expect, має відповідати значенню, що буде передане далі;
- toNotBe – результат порівняння значення, що було передане в expect, не повинно відповідати значенню, що буде передане далі;

Після виклику одного з цих методів, повертається вже об'єкт класу Assertion. Даний об'єкт реалізує магічний метод __call, який реалізує у собі механізм завантаження плагінів типу Plugins\Assertion. Ці плагіни реалізують в собі вже функції, які будуть виконувати порівняння. В базовому варіанті доступний плагін equals, який виконує звичайне порівняння значення, отримане у expect зі значенням, отриманим у ньому.

Таким чином для порівняння двох значень у тілі тесту слід викликати наступну конструкцію:

```
$this->expect(value1)->toBe()->equals(value2);
```

4.4 Розширення базового функціоналу

Розроблений програмний засіб має широкі можливості розширення базового функціоналу. По-перше, мається власна підтримка плагінів для розширення функціоналу клієнта та засобів тестування, по-друге, проєкт розроблено з використанням Composer, а значить можна дуже легко переписати функціонал будь-якого класу за бажанням.

Для створення плагінів слід створити новий клас, що буде наслідувати один з базових класів `AbstractTelegramAction` або `AbstractAssetrion`, в залежності від плагіну, який ми розробляємо. Ці класи містять абстрактний метод `run`, що запускається ядром як початкова точка роботи плагіну. У його тілі необхідно реалізувати логіку роботи плагіну. Також ці класи містить допоміжні методи для спрощення створення плагінів. Початковою точкою роботи є метод `run()`, який ядро викликає одразу після ініціалізації плагіну. У тілі цього методу немає жодних обмежень на використання додаткових бібліотек чи інших класів.

Плагіни для розширення можливостей клієнта

Плагіни розширення можливостей клієнта реалізують методи для відправки та отримання даних з Telegram. У базовому варіанті доступні плагіни для відправки звичайного текстового повідомлення на сервер та відправки команди. Такі плагіни наслідують клас `AbstractTelegramAction`. Цей клас своєю чергою надає доступ до клієнту `MadelineProto`, який же був ініціалізований у ядрі. Також містить 2 допоміжних методи:

- `bot` – повертає ім'я бота, тестування якого проводиться. Використовується для вказання адреси призначення при відправці даних через `MadelineProto`;

- log – заносить у лог тестування заданий текст зі статусом виконання Ok.
Використовується для підвищення інформативності результатів тестів.

Для отримання об'єкту клієнта Telegram слід звертатися до захищеного поля telegramClient. Він являє собою об'єкт класу MadelineProto\Api, а його опис та правила роботи з ним наявні у документації MadelineProto.

Для виклику розроблених плагінів слід викликати у класу Client\Telegram статичний метод з ім'ям базового класу плагіну. Єдина відмінність, починати назву методу при виклику слід з малої літери. Зроблено це для дотримання єдиного стандарту оформлення коду. Розміщаються дані плагіни у директорії Plugins\Telegram.

Плагіни для розширення варіантів тестування

Такі плагіни розширюють кількість правил, за якими перевіряється істинність заданих тверджень. У базовому варіанті доступний плагін для простого порівняння двох переданих величин. Для розширення варіантів тестування розробляє мий плагін повинен наслідувати клас AbstractAssertion. Як і у варіанті з плагінами для клієнту Telegram, даний клас містить допоміжні методи для спрощення написання коду та доступу до необхідних змінних. Наприклад, значення, з яким буде відбуватися порівняння. Доступні методи у тілі класу плагіну:

- setAssertionDetail – встановлює опис для плагіну, що буде відображатися у консолі під час виконання тестів;
- fail – встановити статус невиконання умови істинності твердження;
- succeed - встановити статус виконання умови істинності твердження;
- on – встановлює функцію-обробник для заданого типу об'єкту Expectation. Функція і тип передаються у параметрі.
- done – завершає перевірку і заносить її результати до логу для подальшого виводу
- expectedVal – повертає значення, яке має перевірятися;

Використання даних методів значно підвищує читабельність коду розробляємих плагінів. Побачити приклад цього можна на рисунку 4.4

```
public function run($target)
{
    $this->setAssertionDetail( title: "equals '{$target}'");
    $this->on( expectation: TestingConsts::TESTING_MODE_TOBE, function () use ($target){
        $target == $this->expectedVal() ? $this->succeed() : $this->fail();
    });
    $this->on( expectation: TestingConsts::TESTING_MODE_TONOTBE, function () use ($target) {
        $target != $this->expectedVal() ? $this->succeed() : $this->fail();
    });
    $this->done();
}

public function equals($target)
{
    $info = new Result( title: "Expect '{$this->expectation}' {$this->reason} equals '{$target}'");
    switch ($this->reason) {
        case TestingConsts::TESTING_MODE_TOBE: $target == $this->expectation
            ? $info->setStatus( s: TestingConsts::STATUS_OK)
            : $info->setStatus( s: TestingConsts::STATUS_FAIL);
        break;
        case TestingConsts::TESTING_MODE_TONOTBE: $target != $this->expectation
            ? $info->setStatus( s: TestingConsts::STATUS_OK)
            : $info->setStatus( s: TestingConsts::STATUS_FAIL);
        break;
        default: $info->setStatus( s: TestingConsts::STATUS_UNW);
    }
    $this->expectationInstance->_service_logExpectationResult($info);
}
```

Рисунок 4.4 – Функція, написана з використанням допоміжних методів та без них

Механізм виклику плагінів реалізується за допомогою магічних методів `__call` та `__callStatic` та механізму автозавантаження класів. При виклику плагіну як звичайного методу інтерпретатор бачить, що такий метод відсутній і викликає магічний метод `__call` або `__callStatic` (якщо викликався статичний метод), передаючи йому параметрами ім'я методу, який викликався та аргументи, які були передані з викликом. У тілі цього методу відбувається ініціалізація об'єкту класу `ReflectionClass` з трохи модифікованим іменем, отриманим при виклику `__call`. Завдання `ReflectionClass` визначити чи існує такий плагін та чи наслідує він необхідний абстрактний клас. При дотриманні даних вимог ініціалізується екземпляр цього класу з необхідними параметрами та викликається метод `run`. Якщо такого плагіну не було знайдено у відповідній директорії, або він не є нащадком необхідного класу, то у лог буде виведено повідомлення про помилку.

4.5 Інтерфейс програми

Розроблений ПЗ має консольний інтерфейс, створений з використанням бібліотеки Webmozart\console. Для роботи необхідно відкрити консоль та перейти до директорії з програмою. Для виконання команд слід вводити наступний текст у консоль:

phrexт [назва команди] [параметр]

Парметр не є обов'язковим для усіх команд. Список доступних команд складається з:

- help – виводить перелік доступних команд та їх опис;
- list – виводить список доступних для виконання тестів;
- info – виводить інформацію про тест, ім'я якого було передано у параметрі;
- run – запускає виконання тесту, ім'я якого було передано у параметрі.

При введенні недопустимої команди виводиться повідомлення про помилку.

При запуску команди list виводиться список усіх доступних для виконання тестів. Програма для зручності оперує лише назвами тест файлів, оскільки їх розташування та формат стандартизовані.

```
E:\diploma\Extravert>php ext list
Available tests:
    * sample
```

```
E:\diploma\Extravert>
```

4.5 Результат роботи команди list

При запуску команди info виводиться повна інформація як про тестовий файл, так і про тест кейси, що в ньому знаходяться.

```
E:\diploma\Extravert>php ext info sample
Title: Base behavior
Purpose: testing base bot behavior
Test cases:
```

Title	Purpose
Test help command	Test correctness of reply on command '/help'
Translator test	Test correctness of translation
Test changing translation language	Test possibility of changing translation language

```
E:\diploma\Extravert>
```

4.6 Результат роботи команди info

Спочатку виводяться заголовок та опис самого файлу з тест кейсами. Після них виводиться таблиця тест кейсів, що знаходяться у цьому файлі, яка складається з двох стовпців – title, який містить заголовок тест кейсу та purpose, який містить мету виконання тест кейсу. Для цього використовується інформація, розташована в анотаціях. При відсутності таких анотацій, для поля title використовується назва класу або його методу, а для полів description та purpose встановлюється запис 'none'.

При запуску команди run починається виконання тесту, ім'я якого було передане у параметрі. Спочатку виводиться інформація про сам файл. Його заголовок та опис. Потім виводиться інформація про тест кейс, що буде виконуватися. А за ним таблиця зі статусом проходження тест кейсу. У таблиці наявні як дії, що відбувалися під час тестування (НП відправка повідомлення), так і перевірка тверджень на істинність. Якщо дія чи перевірка скінчилася успіхом, то у полі Result буде запис Ok, інакше – Fail.

```

E:\diploma\Extravert>php ext run sample
Title: Base behavior
Purpose: testing base bot behavior
Test case: Test help command
Task: Test correctness of reply on command '/help'
+-----+-----+
| Assertion | Result |
+-----+-----+
| Send command help | Ok |
| Expect 'Чтобы указать направление перевода просто наберите /tolang и | OK |
| укажите язык из списка языков' to be equals 'Чтобы указать направление | |
| перевода просто наберите /tolang и укажите язык из списка языков' | |
+-----+-----+
Test case: Translator test
Task: Test correctness of translation
+-----+-----+
| Assertion | Result |
+-----+-----+
| Send message test | Ok |
| Expect 'English: test' to not be equals 'test' | OK |
| Expect 'English: test' to be equals 'English: test' | OK |
+-----+-----+
Test case: Test changing translation language
Task: Test possibility of changing translation language
+-----+-----+
| Assertion | Result |
+-----+-----+
| Send command setmylang | Ok |
| Send message English | Ok |
| Expect 'Now your language is English' to be equals 'Now your language is English' | OK |
| Send command tolang | Ok |
| Send message Русский | Ok |
| Expect 'Translate languages: Русский' to be equals 'Translate languages: Русский' | OK |
| Send message test | Ok |
| Expect 'Русский: тест' to be equals 'Русский: тест' | OK |
+-----+-----+

E:\diploma\Extravert>

```

4.7 Результат роботи команди run

Команда help виводить на екран вікно допомоги, де можна побачити опис список доступних команд та опис того, що вони роблять. Білша частина інформації автоматично згенерована бібліотекою Webmozart consoler. Свій опис команд задається при підключенні їх до консолі у класі-конфігурації.

```
E:\diploma\Extravert>php ext help

USAGE
  ext [-h] [-q] [-v [<level>]] [-V] [--ansi] [--no-ansi] [-n] <command> [<arg1>] ... [<argN>]

ARGUMENTS
  <command>      The command to execute
  <arg>          The arguments of the command

GLOBAL OPTIONS
  -h (--help)    Display help about the command
  -q (--quiet)   Do not output any message
  -v (--verbose) Increase the verbosity of messages: "-v" for normal output, "-vv" for more verbose output and "-vvv" for debug
  -V (--version) Display this application version
  --ansi         Force ANSI output
  --no-ansi      Disable ANSI output
  -n (--no-interaction) Do not ask any interactive question

AVAILABLE COMMANDS
  help          Display the manual of a command
  info          Show info about tests container
  list          Show all available test containers
  run           Run single test container

E:\diploma\Extravert>
```

Рисунок 4.8 – Результат роботи команди help

5 НАЛАШТУВАННЯ ТА ЗАПУСК РОЗРОБЛЕНОГО ПЗ

Для того, щоб запустити розроблене ПЗ необхідно мати встановленим інтерпретатор мови php версії 7.4 та менеджер пакетів Composer. Перед першим запуском слід встановити усі необхідні залежності, які потрібні для роботи ПЗ. Це робиться за допомогою команди `composer require`.

Після того, як усі залежності були встановлені, можна приступати до виконання тестування. У якості приклада буде розглянуте тестування бота Yandex.Translate (@YTranslateBot). Даний бот реалізує функціонал перекладача через YandexTranslateAPI. Розроблений нами тест буде включати в себе ініціалізацію бота перед початком тестування, тестування команди `help`, тестування коректності перекладу та тестування можливості зміни мови.

1.) Ініціалізація тесту

Перед запуском теста, у методі `env` ми встановимо російську мову інтерфейсу та переклад на англійську мову. Для цього у метод `env` додамо наступний код:

```
Telegram::command('setmylang');  
Telegram::plainResponse();  
Telegram::sendText('Русский');  
Telegram::command('tolang');  
Telegram::plainResponse();  
Telegram::sendText('English');  
Telegram::plainResponse();
```

У даному коді ми надсилаємо команди `setmylang` та `tolang` й встановлюємо необхідні нам мови.

2.) Створюємо необхідні нам тест-кейси

У нашому випадку необхідно створити 3 тест-кейси:

- Тестування команди `help`;
- Тестування коректності перекладу з російської на англійську;

Зм	Лист	№ докум.	Підп.	Дата

ІАЛЦ.045490.004ПЗ

Арк.
46

- Тестування коректності зміни мов перекладу.

Для тестування команди help напишемо наступний код:

```
/**
 * @testcase
 * @title Test help command
 * @description Test correctness of reply on command '/help'
 */
```

```
public function testHelp()
```

```
{
```

```
    Telegram::command('help');
```

```
    $this->expect(Telegram::plainResponse())->toBe()
```

```
        ->equals(
```

```
            'Чтобы указать направление перевода просто наберите /tolang и
            укажите язык из списка языков'
```

```
        );
```

```
}
```

Тут ми спочатку відправляємо команду help на сервер, а далі за допомогою спеціальної конструкції порівнюємо отриману відповідь з текстом, який повинен був повернутися. Для інших тест-кейсів буде використовуватися схожий підхід: ми відправляємо команди або текстові повідомлення та порівнюємо отриманий результат з очікуваним. У анотаціях ми помічаємо метод як тест кейс, встановлюємо йому заголовок та опис.

Після заповнення тестового файлу зберігаємо його у папці tests та запускаємо його виконання командою `php ext run sample`. Результати, які будуть отримані у результаті виконання тесту ми можемо побачити на рисунку 5.1 Як можна побачити, усі тести було виконано вдало, а отже тестуємий бот відповідає заданим вимогам. При цьому у клієнті Telegram[Рис 5.2] можна побачити діалог, який відбувався з ботом, що свідчить про те, що обмін інформацією відбувався не з емулятором серверу.

Title: Base behavior

Purpose: testing base bot behavior

Test case: Test help command

Task: Test correctness of reply on command '/help'

Assertion	Result
Send command help	Ok
Expect 'Чтобы указать направление перевода просто наберите /tolang и укажите язык из списка языков' to be equals 'Чтобы указать направление перевода просто наберите /tolang и укажите язык из списка языков'	OK

Test case: Translator test

Task: Test correctness of translation

Assertion	Result
Send message test	Ok
Expect 'English: test' to not be equals 'test'	OK
Expect 'English: test' to be equals 'English: test'	OK

Test case: Test changing translation language

Task: Test possibility of changing translation language

Assertion	Result
Send command setmylang	Ok
Send message English	Ok
Expect 'Now your language is English' to be equals 'Now your language is English'	OK
Send command tolang	Ok
Send message Русский	Ok
Expect 'Translate languages: Русский' to be equals 'Translate languages: Русский'	OK
Send message test	Ok
Expect 'Русский: тест' to be equals 'Русский: тест'	OK

Рисунок 5.1 – Результаты виконання створеного тесту

	丹井アロ井 /setmylang	11:52:22 PM
	Yandex.Translate Your language is English. Choose your own language.	11:52:22 PM
	丹井アロ井 Русский	11:52:23 PM
	/tolang	11:52:23 PM
	Yandex.Translate Теперь ваш язык Русский	11:52:23 PM
	Бот переводит на следующие языки: Русский. Выберите язык для перевода.	11:52:23 PM
	丹井アロ井 English	11:52:24 PM
	Yandex.Translate Языки перевода: English	11:52:24 PM
	丹井アロ井 /help	11:52:25 PM
	Yandex.Translate Чтобы указать направление перевода просто наберите /tolang и укажите язык из списка языков	11:52:25 PM
	丹井アロ井 тест	11:52:25 PM
	Yandex.Translate English: test	11:52:26 PM
	丹井アロ井 /setmylang	11:52:27 PM
	Yandex.Translate Ваш язык Русский. Выберите новый язык.	11:52:28 PM
	丹井アロ井 English	11:52:29 PM
	Yandex.Translate Now your language is English	11:52:29 PM
	丹井アロ井 /tolang	11:52:30 PM
	Yandex.Translate Bot translates to languages: English. Choose language to translate.	11:52:30 PM
	丹井アロ井 Русский	11:52:31 PM
	Yandex.Translate Translate languages: Русский	11:52:32 PM
	丹井アロ井 test	11:52:33 PM
	Yandex.Translate Русский: тест	11:52:33 PM

Рисунок 5.2 – Результат виконання тесту

ВИСНОВКИ

Метою даного дипломного проєкту є розроблення програмного засобу для тестування ботів месенджера Telegram. Розроблене ПЗ дозволяє писати тести для ботів на платформі Telegram, виконувати їх та переглядати результати виконання, що призведе до значного підвищення якості кінцевого продукту, адже усі виявлені недоліки будуть виправлені до релізу.

В результаті аналізу існуючих рішень для тестування ботів Telegram було визначено їх основний недолік: усі вони емулюють сервер Telegram, замість прямого звернення до нього. Розробка даної системи є актуальною, оскільки вона пропонує ефективне альтернативне вирішення задачі тестування ботів.

На основі проведеного аналізу засобів проєктування, для розробки була обрана мікроядерна архітектура, а у якості мови програмування – PHP з використанням бібліотек MadelineProto, Webmozart console та Minime annotations.

Розроблена система дозволяє:

- Використовувати сервери Telegram замість їх емуляції;
- Підтримка тестування будь-яких ботів, незалежно від доступності до їх коду та інструментів, що були використані при розробці;
- Використовувати зручний інтерфейс для виконання тестів та перегляду інформації про них.

У подальшому функціонал розробленої системи може бути розширено, а самі засоби можуть бути використані для тестування довільних ботів.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. WhatsApp, WeChat and Facebook Messenger Apps – Global Messenger Usage, Penetration and Statistic URL: <https://www.messengerpeople.com/global-messenger-usage-statistics/> (дата звернення 15.03.2020);
2. Ptbtest a testsuite for Python telegram bot URL: <https://github.com/Eldinnie/ptbtest> (дата звернення 17.03.2020);
3. Fast Telegram Test Api : URL: <https://github.com/jehy/telegram-test-api> (дата звернення 17.03.2020);
4. Telegram test. Simple module for testing telegram bots URL: <https://www.npmjs.com/package/telegram-test> (дата звернення 17.03.2020);
5. The Computer Language Benchmarks Game URL: <http://benchmarksgame.wildervanck.eu/> (дата звернення 19.03.2020);
6. Usage statistics of PHP for websites URL: <https://w3techs.com/technologies/details/pl-php> (дата звернення 19.03.2020);
7. The Zen of Python URL: <https://www.python.org/dev/peps/pep-0020/> (дата звернення 20.03.2020);
8. Майерс Г. Надежность программного обеспечения 1980р.
9. ГОСТ Р 56920-2016 | НАЦИОНАЛЬНЫЕ СТАНДАРТЫ URL: <http://protect.gost.ru/document1.aspx?control=31&baseC=6&page=4&month=6&year=2016&search=&id=203397> (дата звернення 18.03.2020);
10. Немного об архитектурах программного обеспечения URL: <https://habr.com/ru/company/it-grad/blog/276297/> (дата звернення 15.04.2020);
- 11.9 Best Automated Testing Frameworks For PHP URL: <https://www.hongkiat.com/blog/automated-php-test/> (дата звернення 15.05.2020);