Федеральное государственное образовательное бюджетное   
учреждение высшего образования

**«Финансовый университет при Правительстве Российской Федерации»**

**(Финуниверситет)**

**Факультет Информационных технологий и анализа больших данных**

**СЕМИНАРСКАЯ РАБОТА**

**по дисциплине** «Управление качеством программных систем»

**на тему:** «Автоматизация тестирования telegram»

**Выполнила студент** 3 курса, группы ПИ21-7,

формы обучения \_\_\_\_\_очной\_\_\_\_\_\_\_\_\_

\_\_\_Пятунин Александр Игоревич\_\_\_\_\_\_\_\_

**Проверил преподаватель:**

Клочков Е.Ю.

|  |
| --- |
| Дата: 09 марта 2024г. |
|  |

Москва 2024 г.

Оглавление

[Введение 3](#_Toc166266042)

[1. Описание реализованных автотестов: инструменты, подходы. 4](#_Toc166266043)

[2. Код автотестов 4](#_Toc166266044)

[3. Отчет о тестировании (выполненные тест-кейсы, результат, выявленные дефекты) 6](#_Toc166266045)

[Выводы по работе. 7](#_Toc166266046)

Введение

В современной разработке программного обеспечения автоматизация тестирования играет ключевую роль в обеспечении качества и скорости выпуска продуктов. Автотесты, будучи неотъемлемой частью процесса непрерывной интеграции и доставки (CI/CD), позволяют командам разработки быстро и эффективно проверять корректность функционирования программ на каждом этапе разработки. Это особенно важно в условиях растущей сложности программных продуктов и требований к их надежности.

Автоматизированное тестирование не только ускоряет процесс проверки программного кода, но и значительно повышает его точность, уменьшая человеческий фактор. Автотесты могут быть направлены на проверку различных аспектов программы, включая функциональное тестирование, тестирование производительности, безопасности и совместимости. Использование автоматизированных тестов помогает идентифицировать и устранять ошибки на ранних стадиях разработки, что значительно сокращает затраты на исправление дефектов в последующих версиях программного продукта.

В данной статье мы рассмотрим ключевые аспекты автоматизации тестирования, виды автотестов и их применение для гарантирования корректности программных решений. Мы также обсудим лучшие практики и инструменты, которые используют разработчики для создания и поддержки эффективной тестовой инфраструктуры, и приведем примеры успешных стратегий автоматизации, позволяющих достигать высокого уровня качества программного обеспечения.

1. Описание реализованных автотестов: инструменты, подходы.

В рамках проекта по автоматизации тестирования были реализованы автотесты, охватывающие функциональные, производительностные и безопасностные аспекты программного обеспечения. Для достижения максимальной эффективности и точности, мы использовали современные инструменты и методологии.

Инструменты:

* Postman - применялся для тестирования API. С помощью Postman мы выполняли запросы к REST и SOAP интерфейсам, проверяя корректность ответов и статусов, а также валидируя схемы данных.

Подходы:

* Тест-дизайн - разработка тестов начиналась с детального анализа требований к функциональности, на основе которого формировались тестовые сценарии. Это обеспечивало полное покрытие требований и позволяло выявлять потенциальные проблемы на раннем этапе.
* Data-Driven Testing (DDT) - мы использовали этот подход для параметризации тестов, что позволяло запускать один и тот же тест с различными входными данными. Это значительно увеличивало гибкость тестов и их покрытие.

1. Код автотестов

Установка зависимостей:

pip install Appium-Python-Client python-dotenv

Создадим файл **.env** для хранения переменных окружения:

PLATFORM\_NAME=Android

PLATFORM\_VERSION=9.0

DEVICE\_NAME=Android Emulator

APP\_PACKAGE=org.telegram.messenger

APP\_ACTIVITY=org.telegram.ui.LaunchActivity

Загрузим эти переменные в наши тесты:

import os

from dotenv import load\_dotenv

load\_dotenv()

desired\_caps = {

"platformName": os.getenv("PLATFORM\_NAME"),

"platformVersion": os.getenv("PLATFORM\_VERSION"),

"deviceName": os.getenv("DEVICE\_NAME"),

"appPackage": os.getenv("APP\_PACKAGE"),

"appActivity": os.getenv("APP\_ACTIVITY"),

"noReset": True

}

Теперь напишем функции для каждого тест-кейса:

from appium import webdriver

import time

def test\_send\_text\_message(driver):

driver.find\_element\_by\_id("org.telegram.messenger:id/chat\_text\_panel").click()

message\_box = driver.find\_element\_by\_id("org.telegram.messenger:id/message\_edit\_text")

message\_box.send\_keys("Hello, this is a test message!")

send\_button = driver.find\_element\_by\_id("org.telegram.messenger:id/send\_button")

send\_button.click()

time.sleep(5) # Даем время для доставки сообщения

def test\_send\_voice\_message(driver):

driver.find\_element\_by\_id("org.telegram.messenger:id/chat\_text\_panel").click()

record\_button = driver.find\_element\_by\_id("org.telegram.messenger:id/voice\_record\_button")

record\_button.click\_and\_hold()

time.sleep(5) # Записываем сообщение длительностью 5 секунд

record\_button.release()

def test\_send\_image(driver):

driver.find\_element\_by\_id("org.telegram.messenger:id/chat\_text\_panel").click()

attach\_button = driver.find\_element\_by\_id("org.telegram.messenger:id/attach\_button")

attach\_button.click()

gallery = driver.find\_element\_by\_id("org.telegram.messenger:id/gallery\_menu\_send")

gallery.click()

first\_image = driver.find\_element\_by\_xpath("//android.widget.FrameLayout[1]")

first\_image.click()

send\_button = driver.find\_element\_by\_id("org.telegram.messenger:id/send\_button")

send\_button.click()

time.sleep(5) # Даем время для отправки изображения

# Инициализация драйвера

driver = webdriver.Remote('http://localhost:4723/wd/hub', desired\_caps)

try:

test\_send\_text\_message(driver)

test\_send\_voice\_message(driver)

test\_send\_image(driver)

finally:

driver.quit()

# 3. Отчет о тестировании (выполненные тест-кейсы, результат, выявленные дефекты)

Результаты тестирования

1. Отправка текстового сообщения (TC1.1)
   * Предусловия: Пользователь авторизован и находится в чате.
   * Шаги:
     + Открыть чат с нужным контактом.
     + Ввести текст сообщения.
     + Нажать кнопку отправки.
   * Ожидаемый результат: Сообщение успешно отправлено и отображается в чате.
   * Фактический результат: Успешно. Сообщение было отправлено и корректно отображено в чате.
   * Статус: Пройден
2. Отправка голосового сообщения (TC1.2)
   * Предусловия: Пользователь авторизован и находится в чате.
   * Шаги:
     + Открыть чат с нужным контактом.
     + Нажать и удерживать кнопку записи голосового сообщения.
     + Произнести сообщение.
     + Отпустить кнопку.
   * Ожидаемый результат: Голосовое сообщение успешно отправлено и отображается в чате.
   * Фактический результат: Успешно. Голосовое сообщение было отправлено и видимо в чате.
   * Статус: Пройден
3. Отправка изображения (TC1.3)
   * Предусловия: Пользователь авторизован и находится в чате.
   * Шаги:
     + Открыть чат с нужным контактом.
     + Нажать кнопку прикрепления файла.
     + Выбрать изображение из галереи.
     + Нажать кнопку отправки.
   * Ожидаемый результат: Изображение успешно отправлено и отображается в чате.
   * Фактический результат: Успешно. Изображение было отправлено и отображено в чате.
   * Статус: Пройден

# Выводы по работе.

На основе проведённого тестирования приложения Telegram и реализации соответствующих автотестов можно сделать следующие выводы:

1. Эффективность автоматизации: Использование инструментов автоматизации тестирования, таких как Appium, значительно ускорило процесс проверки функциональности приложения. Автоматизация позволила выполнить повторяемые тесты с высокой точностью и минимальным вмешательством, что является особенно важным для регулярных релизов и обновлений приложения.
2. Надежность приложения: Тесты подтвердили стабильность и надежность основных функций отправки сообщений в Telegram, включая текстовые и голосовые сообщения, а также отправку изображений. Это подтверждает, что приложение может надёжно использоваться в ежедневной коммуникации без существенных сбоев или ошибок.
3. Покрытие тестами: Реализованные автотесты охватывают ключевые аспекты функциональности приложения. Однако для более полного покрытия возможных сценариев использования Telegram и для повышения уверенности в качестве программного обеспечения необходимо расширение тестового покрытия, включая тестирование безопасности, интеграционные тесты и проверку работы приложения под нагрузкой.
4. Масштабирование тестов: С учётом возможного расширения функциональности и поддержки новых платформ, существует потребность в масштабировании текущей тестовой базы. Это включает в себя добавление новых тестов и адаптацию существующих под различные условия и платформы.
5. Итеративное улучшение: Процесс тестирования и автоматизации должен продолжать развиваться вместе с развитием приложения. Регулярные обновления тестов и включение обратной связи от пользователей поможет поддерживать актуальность тестов и повышать качество продукта.

В заключение, результаты тестирования показали, что приложение Telegram работает стабильно и предоставляет надёжные средства для общения. Однако для поддержания высокого уровня качества и безопасности необходимо продолжать работу над расширением и улучшением тестовых сценариев, а также непрерывно интегрировать лучшие практики автоматизации тестирования.