

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

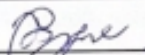
Направление подготовки: 09.03.01 «Информатика и вычислительная техника»
Профиль подготовки: «Программное обеспечение средств вычислительной
техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

на тему

ПРИЛОЖЕНИЕ ЗАПИСИ И
ВОСПРОИЗВЕДЕНИЯ СЦЕНАРИЕВ
ТЕСТИРОВАНИЯ ГРАФИЧЕСКИХ
ПОЛЬЗОВАТЕЛЬСКИХ ИНТЕРФЕЙСОВ

Студент


(подпись, дата)

Бормотов Антон Антонович
(ФИО полностью)

Руководитель


(подпись, дата)

25.05.2026 Кузнецов А.В.
(фамилия, инициалы)

Нормоконтролёр


(подпись, дата)

25.05.2026 Бычков А.С.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 22.05.2026 г. № 14)

Заведующий кафедрой



(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

08 июня 2026 № 11)

Секретарь ГЭК



(подпись)

Е.И. Гурин
(фамилия, инициалы)

Пенза 2026

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»



«Утверждаю»
Заведующий кафедрой
М.А. Митрохин

« 07 » 04 20 26 г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Бормотов Антон Антонович гр. 22ВВП2 факультета вычислительной техники

2. Тема работы: «Приложение записи и воспроизведения сценариев тестирования графических пользовательских интерфейсов»

Тема утверждена приказом ПГУ № 347/о от "07" апреля 2026 г.

3. Руководитель бакалаврской работы доцент кафедры ВТ ПГУ Кузнецов А.В.

4. Задание на работу (назначение разработки, исходные данные и т.п.)

Разработать приложение для автоматизации тестирования графических пользовательских интерфейсов позволяющее записывать и воспроизводить тестовые сценарии, выполнять визуальную верификацию через сравнение скриншотов и формировать PDF-отчёты. Язык программирования Python, среда разработки OpenIDE. Программа должна обладать интуитивно понятным интерфейсом и использовать следующие технологии:

1. PyQt5
2. pytest и pyautogui
3. Pillow и OpenCV
4. ReportLab
5. jsonschema

5. Перечень подлежащих разработке вопросов

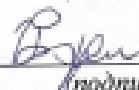
Требуется произвести анализ предметной области: изучить существующие аналоги, изучить способы проектирования и разработки десктопных приложений для автоматизации GUI. Выбрать язык программирования, среду разработки и формат хранения сценариев.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	48	07.04.2026 – 13.04.2026	
2	Анализ требований	36	14.04.2026 – 18.04.2026	
3	Выбор технологий, применяемых при проектировании и разработке	40	19.04.2026 – 24.04.2026	
4	Создание графического интерфейса	12	25.04.2026 – 26.04.2026	
5	Разработка приложения	168	27.04.2026 – 18.05.2026	
6	Оформление пояснительной записки	20	19.05.2026 – 21.05.2026	

Дата выдачи задания "07" апреля 2026 г.

Руководитель бакалаврской работы  Кузнецов А.В.
(подпись, дата)

Задание к исполнению принял студент  Бормотов А.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  Фионова Л.Р.

03.06.2026

Реферат

Бакалаврская работа содержит 119 листов, 27 рисунков, 2 таблицы, 11 источников, 3 приложения.

АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ GUI, ЗАПИСЬ И ВОСПРОИЗВЕДЕНИЕ СЦЕНАРИЕВ, JSON, DSL, СРАВНЕНИЕ СКРИНШОТОВ, PDF-ОТЧЁТЫ, PYTHON, PYQT5

Цель работы – разработать приложение для записи и воспроизведения тестовых сценариев графических пользовательских интерфейсов с возможностью визуальной верификации.

Приложение должно позволять пользователю интерактивно записывать действия (клики мыши, ввод текста, нажатия клавиш, выполнение CLI-команд), редактировать сценарии, воспроизводить их как по отдельности, так и группами, а также автоматически сравнивать сделанные на этапе воспроизведения скриншоты с эталонными, сделанными на этапе записи, выделяя различия между ними. По окончании воспроизведения должен формироваться подробный PDF-отчёт с визуализацией каждого шага.

В конечном итоге было разработано приложение на языке программирования Python с использованием библиотек: PyQt5 для реализации графического интерфейса, ruyput и ruyautogui для перехвата и эмуляции ввода, Pillow и OpenCV для обработки и сравнения изображений, ReportLab для генерации отчетов в формате PDF и jsonschema для валидации тестовых сценариев - которое удовлетворяет всем поставленным задачам. Приложение работает на устройствах с Windows 10/11 и разрабатывалось в среде разработки OpenIDE.

					ПГУ1.090301.30.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Приложение записи и воспроизведения сценариев тестирования графических пользовательских интерфейсов	Лит.	Лист	Листов
Разраб.		Бормотов А.А.		25.05.26				
Провер.		Мухомов А.В.		25.05.26			2	119
Н. Контр		Бычков А.С.		25.05.26		ФВТ, каф. ВТ		
Утверд.		Митрохин М.А.		25.05.26		Экз. 2288772		

Содержание

Перечень сокращений и обозначений	5
Введение	6
1 Приложение записи и воспроизведения сценариев тестирования графических пользовательских интерфейсов	8
1.1 Анализ актуальности темы	8
1.2 Обзор существующих инструментов-аналогов	11
1.3 Постановка задачи на разработку	16
2 Выбор инструментов разработки	21
2.1 Выбор языка программирования	21
2.2 Выбор среды разработки	24
2.3 Выбор формата хранения тестовых сценариев	28
2.4 Обзор используемых библиотек	29
3 Реализация приложения	31
3.1 Общая архитектура приложения	31
3.2 Описание работы приложения с точки зрения пользователя	33
3.3 Реализация основных модулей	36
4 Тестирование приложения	41
4.1 Цели и задачи тестирования	41
4.2 Виды тестирования	41
4.3 План тестирования	43
4.4 Среда и инструменты тестирования	47
4.5 Критерии начала и окончания тестирования	47
4.6 Проверка тестовых сценариев по плану тестирования	49
4.7 Вывод	58
5 Безопасность жизнедеятельности	60
5.1 Правила и техника безопасности при работе с ПК	60

5.2 Организация рабочего места и профилактика утомления	61
Заключение	63
Список использованных источников	65
Приложение А UML-диаграммы	66
Приложение Б Листинг программы	72
Приложение В Презентация	112

Перечень сокращений и обозначений

В данной работе применяют следующие сокращения и обозначения.

ОС – операционная система

ПО – программное обеспечение

API – Application Programming Interface (интерфейс программирования приложений)

CLI – Command Line Interface (интерфейс командной строки)

DSL – Domain-Specific Language (предметно-ориентированный язык)

GUI – Graphical User Interface (графический пользовательский интерфейс)

HTML – HyperText Markup Language (язык гипертекстовой разметки)

IDE – Integrated Development Environment (интегрированная среда разработки)

JSON – JavaScript Object Notation (текстовый формат обмена данными)

JVM – Java Virtual Machine (виртуальная машина Java)

PDF – Portable Document Format (переносимый формат документов)

UML – Unified Modeling Language (унифицированный язык моделирования)

WinAPI – Windows Application Programming Interface (интерфейс прикладного программирования Windows)

XML – eXtensible Markup Language (расширяемый язык разметки)

YAML – YAML Ain't Markup Language (язык сериализации данных)

Введение

В современном мире разработка программного обеспечения идёт высокими темпами, а графические пользовательские интерфейсы (GUI) становятся всё более сложными и функциональными. Качество интерфейса напрямую влияет на восприятие продукта пользователем, а его ошибки могут привести к финансовым потерям и снижению доверия к приложению. Регулярное тестирование GUI необходимо на всех этапах жизненного цикла программного продукта, однако ручное выполнение однотипных проверок требует значительных временных затрат, подвержено человеческим ошибкам и плохо масштабируется при частых изменениях интерфейса.

Автоматизация тестирования графических интерфейсов позволяет многократно воспроизводить заданные сценарии, фиксировать результаты и быстро выявлять регрессионные ошибки. Однако многие существующие инструменты либо требуют глубоких знаний программирования, либо не предоставляют встроенных средств визуальной верификации (сравнения скриншотов), либо не поддерживают выполнение системных команд и CLI-утилит в процессе теста. Также зачастую отсутствует возможность удобной записи сценариев без написания кода и формирования наглядных отчётов о прохождении тестов.

Для удобства тестировщиков и разработчиков целесообразно создать отдельное приложение, которое позволит записывать действия пользователя (клики, ввод текста, нажатия клавиш, выполнение команд), воспроизводить их в автоматическом режиме, сравнивать эталонные и фактические скриншоты, а также генерировать подробные PDF-отчёты. Такое приложение будет полезно как для профессиональных тестировщиков, так и для разработчиков при отладке интерфейсов, а также в учебных целях.

Цели выпускной квалификационной работы:

- исследовать существующие подходы и инструменты автоматизации тестирования GUI;
- определить функциональные требования к приложению для записи и воспроизведения тестовых сценариев;
- разработать современное, удобное и расширяемое приложение для автоматизации тестирования GUI;
- реализовать в приложении ключевые возможности: запись и воспроизведение действий пользователя, визуальную верификацию через сравнение скриншотов, поддержку выполнения внешних команд и формирование детализированных отчётов о результатах тестирования;
- провести тестирование разработанного приложения;
- оценить практическую применимость и эффективность разработанного инструмента;
- предложить рекомендации по дальнейшему развитию и расширению функционала;
- проанализировать полученные результаты и сделать выводы о потенциале разработки данного приложения.

1 Приложение записи и воспроизведения сценариев тестирования графических пользовательских интерфейсов

1.1 Анализ актуальности темы

Современная разработка программного обеспечения характеризуется высокой динамикой изменений, частыми релизами и необходимостью обеспечения стабильного качества продукта. Графический пользовательский интерфейс (GUI) является ключевым звеном взаимодействия конечного пользователя с программой, и его некорректная работа может привести к потере данных, снижению производительности и негативному восприятию продукта в целом. Согласно исследованиям в области качества ПО, до 40 % всех дефектов, обнаруживаемых после выпуска продукта, связаны именно с ошибками в графическом интерфейсе. При этом стоимость исправления такого дефекта на этапе эксплуатации может быть в 10–15 раз выше, чем на этапе разработки.

Традиционным подходом к проверке GUI является ручное тестирование. Тестировщик выполняет последовательность действий (открытие окон, ввод данных, нажатие кнопок, проверка результатов) и визуально оценивает корректность поведения приложения. Такой метод обладает рядом серьёзных недостатков:

- высокая трудоёмкость — для проведения регрессионного тестирования после каждого изменения необходимо заново выполнять сотни и тысячи шагов, что требует больших временных затрат;

- человеческий фактор — даже опытный тестировщик может пропустить ошибку из-за усталости, невнимательности или случайного отклонения от сценария;

— плохая повторяемость — ручной прогон одного и того же сценария в разных условиях (разрешение экрана, версия ОС, масштабирование) даёт разные результаты, что затрудняет локализацию проблем;

— низкая масштабируемость — при росте числа тестовых сценариев пропорционально растёт и требуемый штат сотрудников, что не всегда экономически оправдано.

В связи с этим всё большее распространение получает автоматизированное тестирование GUI. Инструменты автоматизации позволяют записывать действия пользователя, воспроизводить их в нужный момент, а также автоматически проверять ожидаемые результаты. Это даёт возможность выполнять регрессионные тесты после каждой сборки, интегрировать тестирование в CI/CD-пайплайны и значительно сокращать время выхода продукта на рынок.

Однако большинство существующих решений для автоматизации GUI либо ориентированы на конкретные технологии, либо требуют написания кода на языках программирования, что подразумевает наличие у тестировщика навыков разработки. Инструменты класса *captureand replay* (запись и воспроизведение) часто страдают от ограниченной функциональности: они могут имитировать клики и ввод текста, но не умеют выполнять команды операционной системы, проверять состояние файловой системы или сравнивать сложные графические результаты. Кроме того, многие из них не предоставляют встроенных средств для формирования детализированных отчётов.

Особую важность приобретает возможность визуальной верификации — сравнения фактического состояния экрана после выполнения действия с эталонным. Например, при тестировании графического редактора необходимо убедиться, что отрисованная фигура совпадает с ожидаемой. Традиционные инструменты автоматизации, работающие на уровне свойств элементов (текст кнопки, видимость окна), не способны оценить визуальное качество — правильность цветопередачи, расположение пикселей, отсутствие артефактов. Решением является попиксельное сравнение скриншотов, однако далеко не все инструменты

предоставляют такую возможность «из коробки». Среди рассмотренных аналогов ни один не имеет встроенного механизма сравнения эталонного и фактического скриншотов с формированием наглядного отчёта о расхождениях.

Кроме того, в процессе автоматизированного тестирования часто возникает необходимость выполнения внешних команд: запуск тестируемого приложения с определёнными аргументами, очистка временных файлов, перезапуск сервисов. Отсутствие встроенной поддержки командной строки (характерное для многих существующих решений) заставляет разработчика использовать обходные пути, что усложняет инфраструктуру тестов.

Анализ популярных инструментов автоматизации GUI показал, что ни один из них не предоставляет интегрированного решения, включающего:

- интерактивную запись сценариев с возможностью вставки специальных действий (клик по области скриншота, выполнение CLI-команды);
- хранение сценариев в открытом, платформонезависимом формате с валидацией по схеме;
- автоматическое сравнение эталонного и фактического скриншотов с настраиваемым порогом совпадения;
- генерацию подробного отчёта с визуальным отображением расхождений.

Таким образом, разработка приложения, объединяющего перечисленные возможности и не требующего от пользователя навыков программирования, является актуальной научно-практической задачей. Такое приложение может быть востребовано не только профессиональными тестировщиками, но и разработчиками при отладке интерфейсов, а также преподавателями в учебных курсах по тестированию ПО.

Практическая значимость работы заключается в создании инструмента, позволяющего:

- сократить время на регрессионное тестирование GUI-приложений в 3–5 раз по сравнению с ручным прогоном;

- автоматизировать проверку визуального качества интерфейса (вёрстка, отрисовка графики);
- обеспечить единообразное хранение и воспроизведение тестовых сценариев в различных проектах;
- получать наглядные отчёты, которые могут служить доказательной базой при приёмке версий.

1.2 Обзор существующих инструментов-аналогов

В качестве аналогов для анализа были выбраны десктопные приложения и фреймворки, наиболее часто применяемые для автоматизации тестирования GUI. Для сравнения были выбраны три инструмента: AutoIt, SikuliX, Katalon Studio, каждый из которых представляет собой готовый программный продукт, а не библиотеку.

1.2.1 AutoIt

AutoIt — это скриптовый язык и утилита для автоматизации действий в операционной системе Windows (см. рисунок 1). Позволяет имитировать нажатия клавиш, движения мыши, работу с окнами и элементами управления. Скрипты могут компилироваться в исполняемые файлы. Инструмент существует с 1999 года и имеет большое сообщество пользователей.

Достоинства:

- высокая скорость выполнения скриптов;
- поддержка низкоуровневого взаимодействия с окнами через WinAPI;
- возможность компиляции скриптов в EXE-файлы для распространения.

Недостатки:

1. отсутствие встроенного графического механизма «запись-воспроизведение» (скрипты пишутся вручную);
2. нет возможности сравнения скриншотов для верификации результатов;
3. ограниченная работа с .NET-приложениями и современными UI-фреймворками;

4. сценарии хранятся в проприетарном формате, сложны для автоматической генерации отчётов.

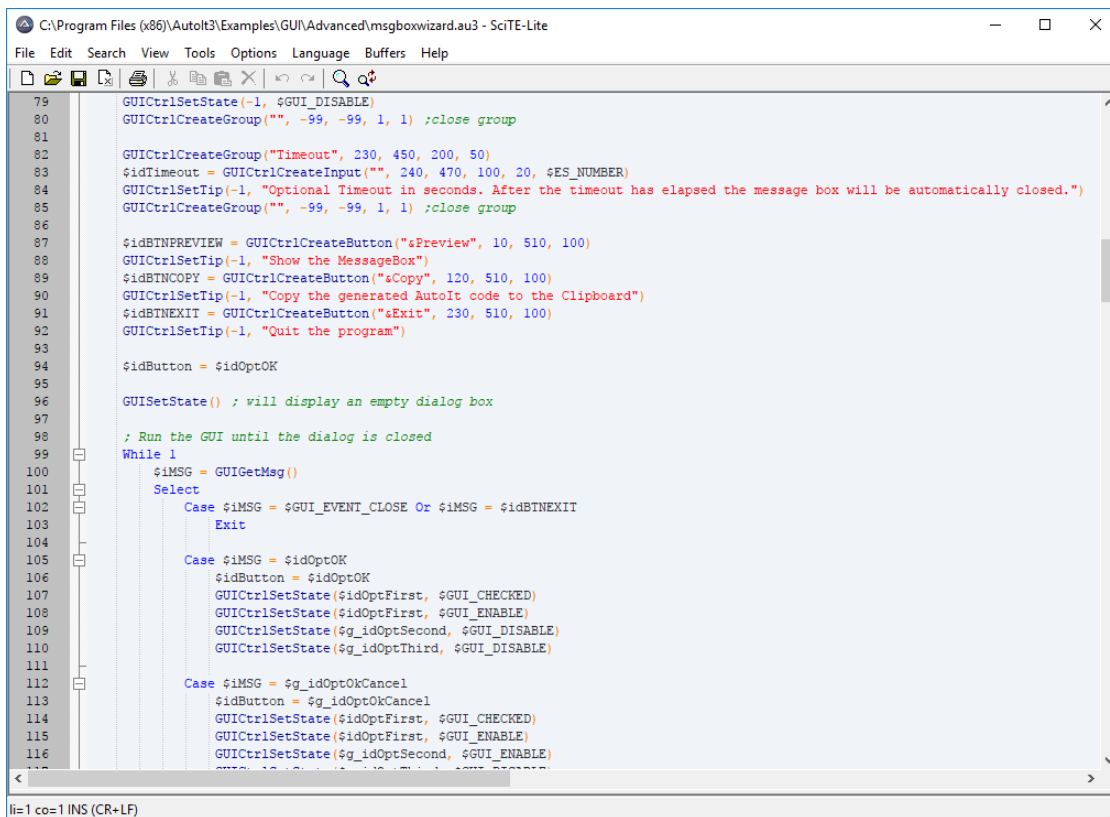


Рисунок 1 – Интерфейс редактора скриптов AutoIt

1.2.2 SikuliX

SikuliX (см. рисунок 2) использует технологию распознавания изображений для поиска элементов на экране. Пользователь может указать область экрана (скриншот), по которой программа будет кликать или вводить текст. Инструмент написан на Java и кроссплатформенен.

Достоинства:

- интуитивно понятный подход «что вижу – то кликаю»;
- возможность работы с любыми приложениями, независимо от их внутреннего устройства;
- поддержка скриптов на Python (Jython) для сложной логики.

Недостатки:

- нет встроенной поддержки выполнения команд командной строки непосредственно в сценарии;
- отсутствует автоматическая генерация отчётов с визуальным сравнением «эталон – результат»;
- формат сценариев (Python-скрипты) не унифицирован для обмена и затрудняет параметризацию тестов;
- работа с изображениями чувствительна к разрешению экрана и масштабированию.

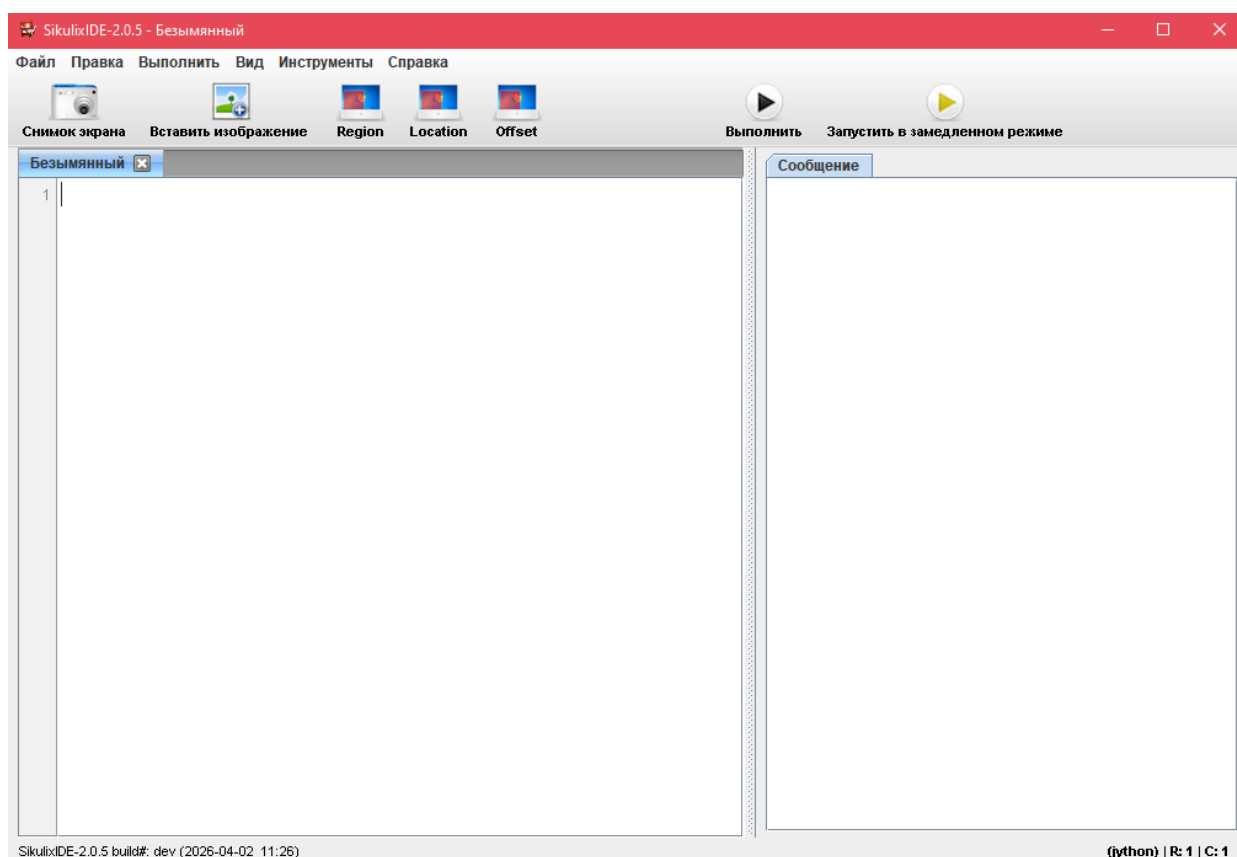


Рисунок 2 – Окно редактора SikuliX

1.2.3 Katalon Studio

Katalon Studio (см. рисунок 3) – это универсальная платформа для автоматизации тестирования, поддерживающая веб-, API-, мобильные и десктопные приложения. Инструмент построен на основе Selenium и Appium и предоставляет как режим записи и воспроизведения, так и возможность ручного

написания тестов. Katalon Studio существует как в бесплатной (Open Source), так и в коммерческой версиях.

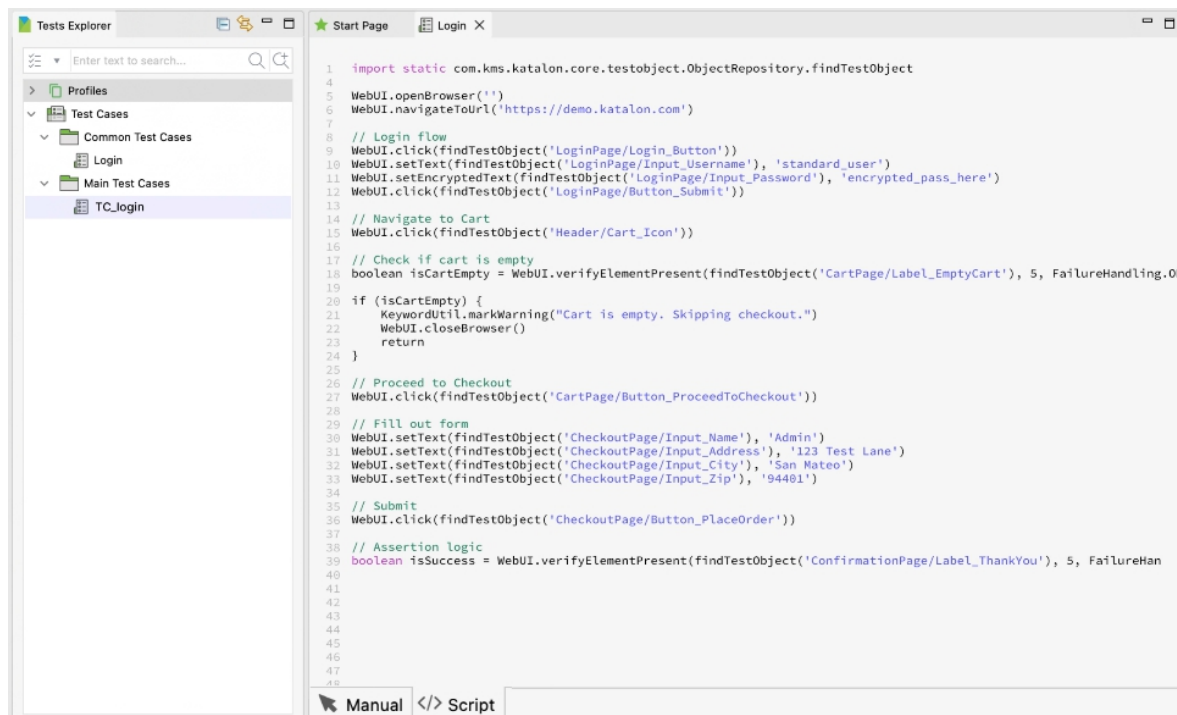


Рисунок 3 – Интерфейс Katalon Studio

Достоинства:

- встроенная поддержка записи и воспроизведения действий (Record & Playback);
- обширные возможности для тестирования API и веб-приложений;
- интеграция с CI/CD-системами (Jenkins, GitLab CI и др.);
- генерация HTML-отчётов с логами выполнения.

Недостатки:

- основной фокус на тестирование веб- и мобильных приложений; для десктопных Windows-приложений требуется дополнительная настройка;
- относительно высокий порог входа для освоения всех возможностей;
- отсутствие встроенного механизма сравнения скриншотов для визуальной верификации;
- лицензирование коммерческой версии ограничивает использование в некоторых проектах.

1.2.5 Сравнительный анализ

В таблице 1 приведено сравнение рассмотренных аналогов по ключевым критериям.

Таблица 1 – Сравнение существующих инструментов

Критерий	AutoIt	SikuliX	Katalon Studio	Разрабатываемое приложение
Графический интерфейс для записи сценариев	–	+	+	+
Воспроизведение сценария	+	+	+	+
Клик по области скриншота	–	+	–	–
Выполнение команд CLI	–	–	–	+
Вставка текста / горячие клавиши	+	+	+	+
Сравнение скриншотов	–	–	–	+
Формат сценариев	скриптовый	Python	внутренний (на основе Groovy/Java)	JSON
Генерация отчёта	–	–	+ (HTML)	+ (PDF)
Бесплатность	+ (Open Source)	+ (Open Source)	+ (есть бесплатная версия)	+

Таким образом, ни один из существующих инструментов не предоставляет полного набора требуемых функций: интерактивной записи сценариев с возможностью выполнения системных команд, автоматической визуальной верификации по скриншотам, хранения сценариев в унифицированном формате JSON и формирования детализированного PDF-отчёта. Это обосновывает актуальность разработки нового программного решения.

1.3 Постановка задачи на разработку

Требуется реализовать standalone-приложение для записи и воспроизведения тестовых сценариев графических пользовательских интерфейсов. Приложение должно обеспечивать следующие функциональные возможности.

1.3.1 Функциональные требования

1.3.1.1 Запись сценария. Приложение должно обеспечивать фиксацию следующих действий пользователя:

- клики левой, правой и средней кнопкой мыши с сохранением абсолютных координат;
- двойные клики, распознаваемые по быстрым повторным нажатиям в одной точке;
- перетаскивание объектов (drag-and-drop) с фиксацией начальных и конечных координат;
- ввод текста с сохранением вводимой строки;
- нажатия обычных клавиш (буквы, цифры, знаки) и специальных клавиш (Enter, Tab, F1–F12, PageUp, PageDown, Home, End, Delete, Escape, PrintScreen, CapsLock, NumLock, ScrollLock, а также Ctrl, Alt, Shift, Win);
- выполнение произвольных команд в командной строке;
- создание паузоожиданий заданной длительности.

В процессе записи должна быть предусмотрена возможность временной приостановки фиксации действий (например, нажатием отдельной управляющей клавиши) с возможностью последующего возобновления записи. Предусматривается возможность дозаписи новых действий в уже существующий сценарий без потери ранее сохранённых шагов. Все записанные действия должны сохраняться в файл формата JSON.

1.3.1.2 Воспроизведение сценария. Приложение должно обеспечивать последовательное выполнение действий из сохранённого JSONсценария. Для эмуляции ввода должны использоваться средства имитации нажатия клавиш, движений мыши и ввода текста, поддерживающие корректное воспроизведение как

на уровне физических клавиш (виртуальные коды), так и на уровне символьного ввода с учётом раскладки клавиатуры.

Требуется поддержка всех типов действий, перечисленных в п. 1.3.1.1: клик, двойной клик, перетаскивание, ввод текста (с автоматическим переключением раскладки при необходимости), нажатие клавиш, выполнение CLI-команд, ожидание.

Должна быть предусмотрена возможность воспроизведения как одного выбранного сценария, так и группы сценариев последовательно (один за другим). При воспроизведении группы сценариев при возникновении ошибки в любом из них выполнение должно останавливаться.

Во время выполнения мышинных действий (клик, двойной клик, перетаскивание) система должна создавать вспомогательное графическое изображение (скриншот) с визуальной маркировкой положения курсора или траектории перетаскивания для последующего включения в отчёт. При этом основное окно приложения должно автоматически скрываться, чтобы не мешать тестируемому приложению.

1.3.1.3 Верификация и сравнение скриншотов. В процессе записи пользователю должна предоставляться возможность выделить произвольную прямоугольную область экрана (например, с помощью зажатия специальной клавиши и перемещения мыши) для создания эталонного скриншота. После выделения области в сценарий должно автоматически добавляться соответствующее действие, а эталонный скриншот сохраняться в файловой системе.

При воспроизведении сценария на шаге, связанном с проверкой визуального состояния, система должна делать фактический скриншот той же области и сравнивать его с ранее сохранённым эталонным снимком (попиксельно или с настраиваемым порогом допуска).

При обнаружении различий должна формироваться разностная диаграмма (diff-изображение) с визуальным выделением изменившихся областей. Результат

сравнения (совпадение / несовпадение) должен фиксироваться и сохраняться для последующего включения в отчёт.

1.3.1.4 Формат сценариев и валидация. Сценарий должен представлять собой JSON-файл определённой структуры, а именно – массив объектов, каждый из которых содержит поля `type` (тип действия) и `parameters` (параметры действия). Допустимые типы действий: клик, двойной клик, перетаскивание, ввод текста, нажатие клавиши (по имени или по виртуальному коду), выполнение CLI-команды, ожидание, создание скриншота для верификации.

Для проверки корректности структуры сценария должна применяться JSON Schema (библиотека `jsonschema`). Дополнительно для удобства ручного редактирования должен поддерживаться текстовый DSL-формат, который автоматически преобразуется в JSON и обратно без потери данных.

1.3.1.5 Генерация отчёта. После завершения воспроизведения одного или нескольких сценариев программа должна формировать PDF-отчёт. Отчёт должен содержать:

- общий заголовок с указанием имени сценария (или перечня сценариев для сводного отчёта);
- для каждого шага сценария – порядковый номер, тип действия, параметры в текстовом виде;
- для мышинных действий – изображение (скриншот) с визуальным маркером выполнения действия;
- для шагов верификации скриншотов – парное отображение эталонного и фактического (или разностного) изображений с подписями.

Все изображения в отчёте должны автоматически масштабироваться с сохранением пропорций, чтобы вписываться в ширину страницы. При формировании сводного отчёта по нескольким сценариям каждый следующий сценарий должен начинаться с новой страницы.

1.3.1.6 Пользовательский интерфейс. Графический интерфейс приложения должен быть реализован на базе библиотеки PyQt5. Требуются следующие основные элементы:

- главное окно, содержащее список доступных сценариев (с поддержкой множественного выбора);
- многострочный редактор для просмотра и ручного редактирования сценария в текстовом виде;
- кнопки управления: «Новый сценарий», «Удалить сценарий», «Сохранить сценарий», «Записать», «Воспроизвести», «Создать отчёт», «Вставить текст», «CLI команда», «Ожидание», а также кнопка переключения формата отображения сценария (DSL / JSON);
- строка состояния для вывода текущей информации (статус записи, результаты сохранения, сообщения об ошибках).

Приложение должно поддерживать свёртывание в системный трей при закрытии главного окна с возможностью восстановления окна по двойному клику на иконке трея. По окончании воспроизведения (независимо от результата) должно выводиться уведомление в трее. Все диалоговые окна для ввода имени сценария, текста, команд и времени ожидания должны быть реализованы с использованием стандартных средств библиотеки.

На рисунке 4 представлена UML-диаграмма вариантов использования.

1.3.2 Нефункциональные требования:

- платформа: Windows 10/11 (x64) – как основная целевая ОС;
- минимальные версии технологий: Python 3.10, PyQt5, pynput, keyboard, pyautogui, Pillow, ReportLab, jsonschema;
- приложение должно быть standalone – не требовать установки дополнительных серверов или веб-компонентов, распространяться как каталог с исполняемым файлом (упаковка через PyInstaller);
- время реакции интерфейса на действия пользователя – не более 1 секунды.



Рисунок 4 – Диаграмма вариантов использования

2 Выбор инструментов разработки

В данном разделе описываются инструменты и технологии, выбранные для реализации программного комплекса автоматизации тестирования графических интерфейсов. Анализ инструментов проводился с учётом требований к кроссплатформенности, производительности, простоте разработки и наличию необходимых библиотек для захвата изображений, эмуляции ввода и генерации отчётов.

2.1 Выбор языка программирования

Для реализации приложения требовался язык, обеспечивающий быструю разработку, богатую экосистему для работы с графикой и системными вызовами, а также наличие средств для создания исполняемых файлов (standalone-приложений). Рассмотрены три языка, наиболее часто используемых для задач автоматизации рабочего стола.

2.1.1 Язык программирования C++

C++ является компилируемым языком низкого уровня, предоставляющим максимальную производительность и прямой доступ к системным вызовам операционной системы, включая WinAPI. Это делает его привлекательным для задач, требующих минимальных задержек при эмуляции ввода.

Достоинства:

- максимальная производительность и минимальное потребление ресурсов;
- прямой доступ к WinAPI, что позволяет реализовывать низкоуровневую эмуляцию ввода на уровне драйверов;
- возможность создания полностью независимых исполняемых файлов без необходимости установки дополнительных сред выполнения;
- большой выбор библиотек для работы с изображениями (OpenCV, Boost.GIL).

Недостатки:

- высокая сложность разработки и отладки, обусловленная необходимостью ручного управления памятью и работой с указателями;
- отсутствие встроенных кроссплатформенных средств для глобального захвата событий ввода (требуется использование платформозависимых решений);
- значительный объём кода для решения типовых задач (например, парсинг JSON требует подключения сторонних библиотек и написания обёрток);
- длительный цикл разработки и тестирования по сравнению с интерпретируемыми языками.

2.1.2 Язык программирования Java

Java представляет собой объектно-ориентированный язык, работающий поверх виртуальной машины JVM. Он широко используется для создания кроссплатформенных приложений.

Достоинства:

- кроссплатформенность — приложение может работать на Windows, Linux и macOS без изменения кода;
- автоматическое управление памятью (сборщик мусора), снижающее риск утечек памяти;
- богатая стандартная библиотека, включающая средства для работы с изображениями (java.awt), эмуляции ввода (java.awt.Robot) и потоковой обработки данных;
- строгая типизация и проверка ошибок на этапе компиляции.

Недостатки:

- требует установленной Java Runtime Environment (JRE) на целевой машине, что противоречит требованию standalone-приложения;
- повышенное потребление оперативной памяти (типичное приложение на Java занимает 100–200 МБ без учёта загрузки данных);
- более медленный запуск по сравнению с нативными приложениями из-за инициализации JVM;

- сложность упаковки в единый исполняемый файл для Windows (требуется использование средств типа `jpackage` или `launch4j`).

2.1.3 Язык программирования Python

Python – интерпретируемый язык с динамической типизацией, обладающий лаконичным синтаксисом и огромной экосистемой библиотек. Он зарекомендовал себя как эффективное средство для задач автоматизации и прототипирования.

Достоинства:

- лаконичный и читаемый синтаксис, сокращающий время написания и отладки кода в 2–3 раза по сравнению с C++ и Java;
- обширная экосистема библиотек для автоматизации: `pyautogui` (эмуляция мыши и клавиатуры), `rpymp` (глобальный захват событий), `Pillow` (обработка изображений), `keyboard` (работа с горячими клавишами);
- возможность упаковки приложения в standalone-исполняемый файл с помощью `PyInstaller` без необходимости установки интерпретатора на целевой машине;
- наличие фреймворков для построения графического интерфейса (`PyQt5`, `Tkinter`, `wxPython`);
- большое сообщество и обилие документации, что упрощает поиск решений типовых задач.

Недостатки:

- более низкая скорость выполнения по сравнению с компилируемыми языками (однако для задач воспроизведения тестовых сценариев, где основное время занимают задержки между действиями и реакции приложений, этот недостаток не является критическим);
- динамическая типизация может приводить к ошибкам, проявляющимся только во время выполнения;
- более высокое потребление памяти по сравнению с C++.

На основании проведенного анализа для реализации приложения был выбран язык Python версии 3.10 и выше. Выбор обусловлен оптимальным балансом между

скоростью разработки, доступностью необходимых библиотек и возможностью создания standalone-приложения.

2.2 Выбор среды разработки

Для написания кода на Python были рассмотрены три наиболее популярные интегрированные среды разработки (IDE).

2.2.1 PyCharm

PyCharm — профессиональная IDE от компании JetBrains, существующая в двух редакциях: бесплатной Community и платной Professional.

Достоинства:

- интеллектуальный анализ кода с автодополнением и проверкой ошибок в реальном времени;
- встроенная поддержка виртуальных окружений (venv, pipenv, conda);
- мощные инструменты рефакторинга (переименование, выделение метода, изменение сигнатуры);
- интеграция с системами контроля версий (Git, SVN, Mercurial);
- встроенный отладчик с поддержкой точек останова, условных остановов и анализа переменных;
- поддержка тестовых фреймворков (pytest, unittest) с графическим интерфейсом запуска.

Недостатки:

- высокие системные требования (рекомендуется 8 ГБ ОЗУ и более), что делает её медленной на старых компьютерах;
- длительное время индексации проектов большого объёма;
- бесплатная версия Community имеет ограничения (например, отсутствует поддержка веб-фреймворков и удалённой разработки);
- более сложный интерфейс по сравнению с лёгкими редакторами.

2.2.2 Visual Studio Code (VS Code)

VS Code — бесплатный, легковесный и высокорасширяемый редактор, разработанный компанией Microsoft.

Достоинства:

- кроссплатформенность (Windows, Linux, macOS) с единым интерфейсом;
- низкое потребление ресурсов и быстрый запуск;
- огромный каталог расширений (marketplace), включая официальный пакет Python от Microsoft;
- встроенный терминал, интеграция с Git и отладчик;
- поддержка удалённой разработки через SSH и контейнеры (Dev Containers);
- бесплатность и открытый исходный код.

Недостатки:

- для полноценной работы с Python требуется установка и настройка нескольких расширений (Python, Pylance, pytest, Pyright);
- менее глубокая интеграция с виртуальными окружениями по сравнению с PyCharm;
- отсутствие встроенных инструментов рефакторинга уровня PyCharm;
- конфигурация отладчика и параметров запуска производится через JSON-файлы (launch.json, settings.json), что может быть сложно для начинающих.

2.2.3 OpenIDE

OpenIDE — это бесплатная, кроссплатформенная среда разработки с открытым исходным кодом, созданная на платформе IntelliJ IDEA. Проект стартовал в 2024 году и является совместной разработкой «Группы Астра», Axiom JDK и Haulmont. Инструмент предназначен в первую очередь для профессиональных Java-разработчиков, но также поддерживает другие языки, включая Python, через систему плагинов.

Достоинства:

- технологическая независимость: инфраструктура проекта (включая маркетплейс, серверы обновлений и телеметрии) полностью расположена на

территории России, что гарантирует стабильную работу и защиту от внешних ограничений;

- открытость и лицензионная чистота: исходный код OpenIDE доступен под лицензией AGPLv3. Среда не содержит проприетарных компонентов, что делает её привлекательной для использования в госкорпорациях и проектах, требующих полного контроля над стеком технологий;

- собственный маркетплейс: в официальном маркетплейсе доступно более 350 плагинов, включая поддержку Python и Go. При этом среда полностью совместима с существующими плагинами для IntelliJ IDEA;

- производительность: по отзывам, удаление «лишних» зависимостей и сервисов сделало OpenIDE более лёгкой и отзывчивой по сравнению с оригинальной IntelliJ IDEA;

- интеграция с российскими инструментами: в среду встроен Git-клиент, и она поддерживает работу с российским сервисом управления кодом GitFlic;

- включение в Реестр отечественного ПО: с 30 декабря 2025 года OpenIDE официально включена в реестр Минцифры, что подтверждает её соответствие требованиям для использования в государственных проектах.

Недостатки:

- ограниченная экосистема: несмотря на наличие маркетплейса, количество специализированных плагинов для Python пока уступает зрелым IDE, таким как PyCharm;

- молодость проекта: в то время как PyCharm и VS Code развиваются уже много лет, OpenIDE находится на раннем этапе (стабильный релиз состоялся в конце 2025 года). В связи с этим некоторые функции могут быть не полностью стабилизированы, а размер сообщества пока значительно меньше;

- необходимость донастройки: для полноценной разработки на Python пользователю требуется вручную установить соответствующий плагин из маркетплейса, тогда как в PyCharm поддержка языка доступна «из коробки»;

- ориентация на Java: среда в первую очередь заточена под Java-разработку, поэтому интерфейс и опыт могут быть избыточными для специалистов, занимающихся исключительно Python;

- нетривиальный процесс сборки: как отмечают разработчики, сборка OpenIDE из исходного кода — сложная инженерная задача из-за множества нативных компонентов и особенностей билд-системы JetBrains. Однако это не является недостатком для рядового пользователя, скачивающего готовую сборку.

2.2.4 Обоснование выбора OpenIDE для дипломного проекта

Для реализации задач дипломного проектирования выбор был сделан в пользу OpenIDE. Это решение обусловлено совокупностью нескольких факторов:

- соответствие целям проекта: OpenIDE предоставляет все необходимые базовые функции для разработки на Python, такие как интеллектуальное автодополнение кода, отладка и поддержка систем контроля версий, что полностью покрывает потребности проекта;

- лицензионная чистота и безопасность: использование OpenIDE, включённой в Реестр отечественного ПО, гарантирует отсутствие каких-либо правовых рисков;

- адаптация под современные условия: полная независимость от зарубежной инфраструктуры гарантирует стабильную работу инструмента в долгосрочной перспективе и отсутствие рисков, связанных с санкционными ограничениями;

- достаточность функционала: для выполнения поставленных задач не требуется использование узкоспециализированных или коммерческих функций, которые есть в других IDE, но отсутствуют в OpenIDE.

Таким образом, выбор OpenIDE в качестве среды разработки является обоснованным и рациональным решением с учётом специфики, целей и контекста выполнения данной дипломной работы.

2.3 Выбор формата хранения тестовых сценариев

Тестовые сценарии представляют собой последовательность действий пользователя (клики, ввод текста, нажатия клавиш, выполнение команд). Для их хранения был рассмотрен ряд форматов.

2.3.1 XML

Расширяемый язык разметки поддерживает вложенные структуры и схемы валидации (XSD). Однако XML избыточен по объёму, сложнее для ручного редактирования и требует больше кода для парсинга.

2.3.2 YAML

Формат, ориентированный на читаемость человеком. Поддерживает комментарии и сложные структуры. Недостатком является чувствительность к отступам и необходимость установки сторонней библиотеки (PyYAML).

2.3.3 JSON

JavaScript Object Notation — лёгкий текстовый формат обмена данными. Преимущества JSON:

- поддерживается стандартной библиотекой Python (модуль `json`), что исключает дополнительные зависимости;
- легко читается как человеком, так и машиной;
- структура (массивы, объекты, строки, числа) полностью покрывает потребности описания сценариев (список действий с параметрами);
- существует стандарт JSON Schema для валидации структуры сценариев.

На основании этого в качестве формата хранения сценариев выбран JSON. Каждый сценарий представляет собой JSON-массив, где каждый элемент — объект с полями `type` (тип действия), `parameters` (параметры) и при необходимости `screenshot_required` (флаг необходимости сравнения скриншота).

2.4 Обзор используемых библиотек

Для реализации всех функциональных требований были выбраны следующие Python-библиотеки.

2.4.1 PyAutoGUI

Библиотека для программного управления мышью и клавиатурой. Она позволяет:

- перемещать курсор и выполнять клики в заданных координатах;
- вводить текст;
- нажимать комбинации клавиш;
- делать скриншоты всего экрана или его области.

Используется в модуле воспроизведения для имитации действий пользователя.

2.4.2 Pynput и Keyboard

Библиотеки для глобального захвата событий ввода. `pynput` позволяет прослушивать нажатия клавиш и кнопок мыши в фоновом режиме, что необходимо для модуля записи сценариев. `keyboard` используется для эмуляции сложных комбинаций клавиш и ожидания определённых нажатий.

2.4.3 Pillow (PIL Fork)

Библиотека для обработки изображений. Применяется для:

- загрузки и сохранения скриншотов в различных форматах;
- сравнения эталонного и фактического скриншотов (попиксельно или с порогом допуска);
- выделения областей расхождения для последующей вставки в PDF-отчёт.

2.4.4 PyQt5

Фреймворк для создания графических интерфейсов. Выбран по следующим причинам:

- кроссплатформенность (Windows, Linux, macOS);
- наличие визуального дизайнера Qt Designer для быстрого создания форм;
- богатый набор виджетов (таблицы, текстовые поля, кнопки, вкладки);
- поддержка работы с изображениями (QLabel, QPixmap).

В приложении `PyQt5` используется для главного окна, окна просмотра отчётов и диалогов настройки.

2.4.5 ReportLab и OpenCV

Для генерации PDF-отчётов выбрана библиотека ReportLab для прямой генерации PDF в связке с библиотекой OpenCV для возможности выделения элементов на скриншотах.

2.4.6 jsonschema

Библиотека для валидации JSON-документов по заданной схеме. Используется для проверки корректности загружаемых сценариев перед их воспроизведением. Схема описывает допустимые типы действий, обязательные поля и форматы параметров.

3 Реализация приложения

В данном разделе описывается практическая реализация программного комплекса автоматизации тестирования графических интерфейсов. Приложение разработано на языке Python с использованием библиотек PyQt5 (графический интерфейс), rounput (перехват и эмуляция ввода), pyautogui (управление мышью и снятие скриншотов), Pillow (обработка изображений), OpenCV (расширенное сравнение), ReportLab (генерация PDF-отчётов), jsonschema (валидация сценариев). Архитектура построена по классическому паттерну ModelViewController, что обеспечивает чёткое разделение ответственности, упрощает сопровождение и позволяет независимо модифицировать каждый компонент.

3.1 Общая архитектура приложения

На рисунке 5 представлена диаграмма классов разработанного приложения.

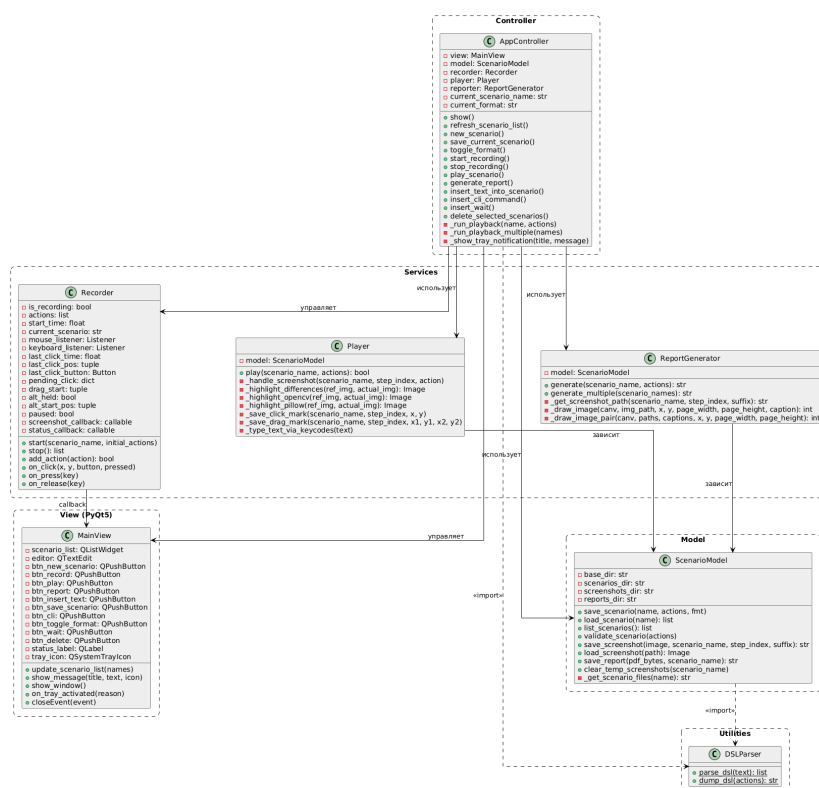


Рисунок 5 – Архитектура приложения (диаграмма классов)

Приложение разделено на четыре основных модуля, взаимодействующих через контроллер:

- модуль представления (`view.py`) — класс `MainView`, отвечающий за графический интерфейс пользователя. Обеспечивает отображение списка сценариев, редактор содержимого (поддерживающий форматы JSON и DSL), кнопки управления, строку состояния и системный трей;

- модуль управления (`controller.py`) — класс `AppController`, который связывает интерфейс с моделью и сервисами. Обработывает сигналы от кнопок, управляет записью, воспроизведением, генерацией отчётов, переключением форматов, добавлением ручных действий (текст, CLI, ожидание) и удалением сценариев;

- модуль хранения и валидации (`model.py`) — класс `ScenarioModel`. Обеспечивает работу с файловой системой: сохранение и загрузку сценариев в форматах `.test` (DSL) и `.json`, управление скриншотами (эталонными, фактическими, diff), валидацию по JSON-схеме и создание PDF-отчётов.

- Сервисные модули:

- `recorder.py` — запись действий пользователя в реальном времени;

- `player.py` — воспроизведение сценариев с эмуляцией мыши, клавиатуры, выполнением CLI-команд и сравнением скриншотов;

- `reporter.py` — генерация PDF-отчётов с визуализацией шагов и сравнений;

- `dsl_parser.py` — преобразование между DSL-текстом и внутренним JSON-представлением.

Все модули используют единую файловую структуру: папка `scenarios` для сценариев, `screenshots` для скриншотов (подпапки по имени сценария), `reports` для итоговых PDF.

3.2 Описание работы приложения с точки зрения пользователя

Взаимодействие пользователя с приложением осуществляется через графический интерфейс. Последовательность типовых сценариев использования:

3.2.1 Создание нового теста

- нажать «Новый сценарий», ввести имя – в списке появляется пустой сценарий;
- нажать «Записать» – окно скрывается, начинается перехват событий;
- выполнить необходимые действия в тестируемом приложении (клики, ввод текста, выделение области скриншота с Alt);
- нажать ту же кнопку «Записать» (теперь она называется «Остановить запись») – запись прекращается, окно восстанавливается, сценарий сохраняется в выбранном формате (по умолчанию DSL);
- при необходимости добавить CLI-команду или ожидание через соответствующие кнопки.

На рисунке 6 представлена UML-диаграмма последовательности для процесса записи.

3.2.2 Редактирование сценария

- выбрать сценарий в списке – его содержимое отображается в редакторе;
- изменить текст (в формате DSL или JSON) вручную;
- нажать «Сохранить сценарий» – произойдёт валидация и запись на диск;
- для переключения формата нажать «DSL / JSON» – содержимое преобразуется автоматически.

На рисунке 7 представлена UML-диаграмма последовательности для процесса редактирования.

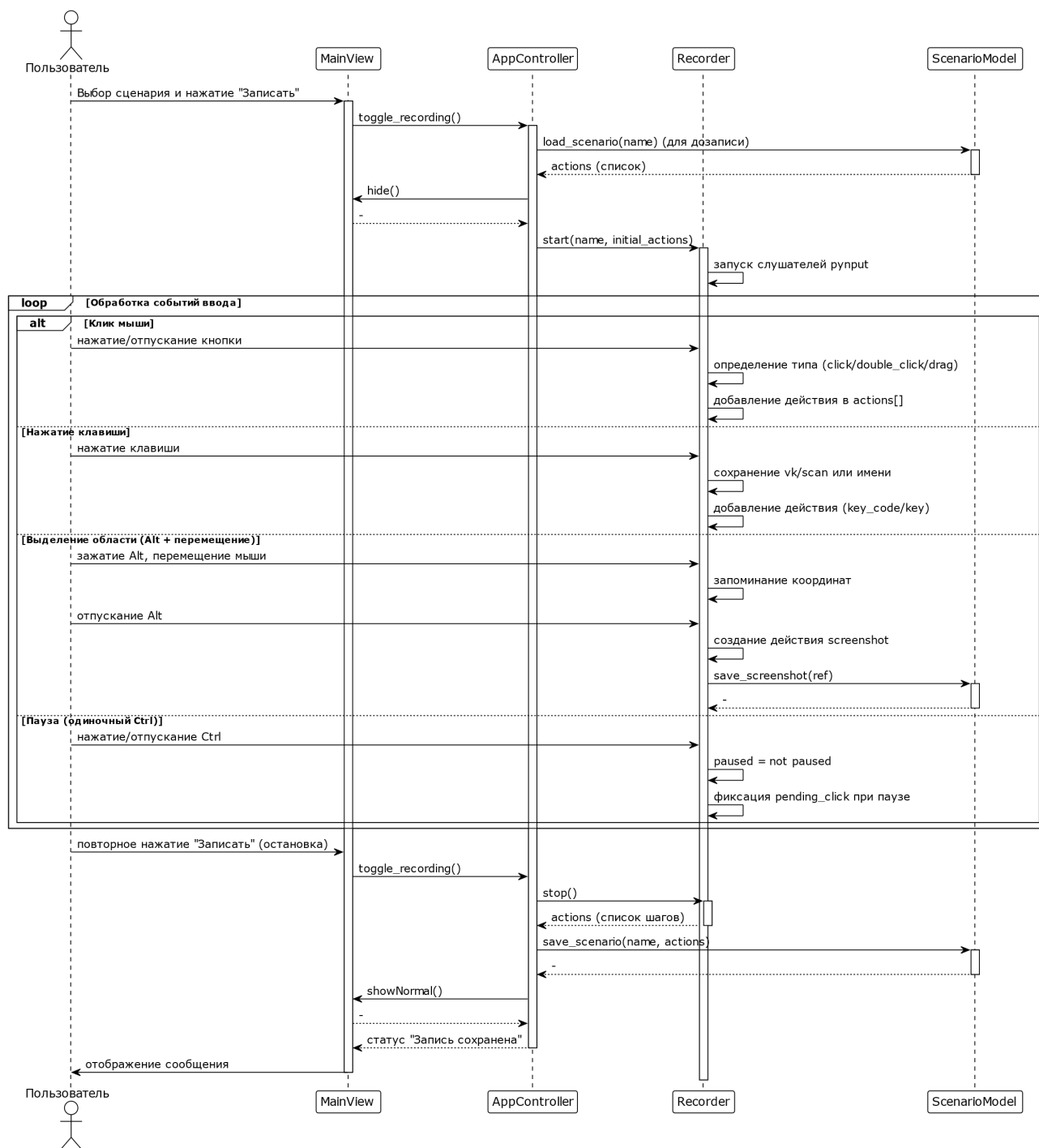


Рисунок 6 – Диаграмма последовательности записи

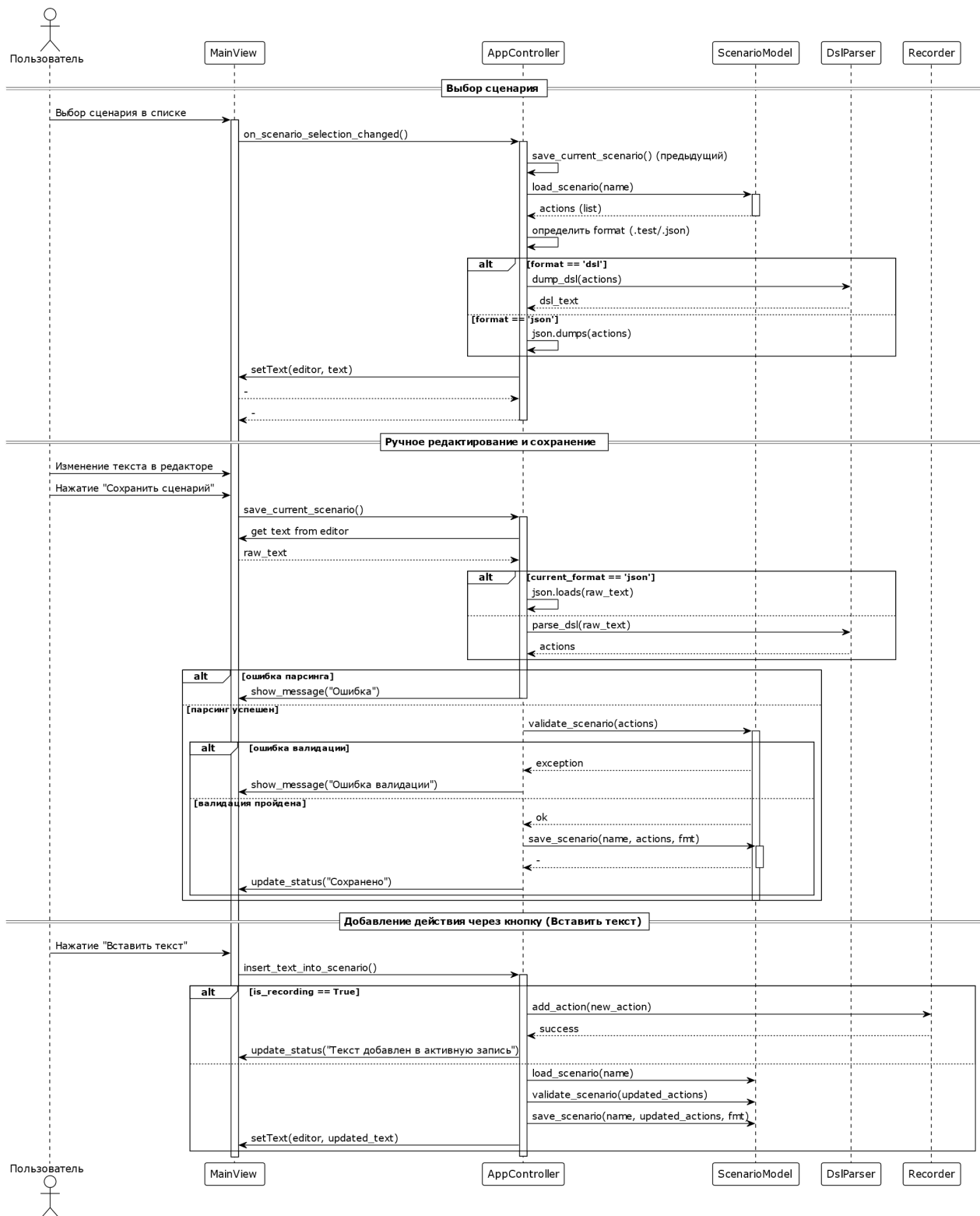


Рисунок 7 – Диаграмма последовательности редактирования

3.2.3 Воспроизведение и верификация

— выбрать один или несколько сценариев (с помощью Ctrl или Shift);

- нажать «Воспроизвести» – окно приложения скрывается, последовательно выполняются все действия;
- после завершения (или ошибки) появляется уведомление в трее;
- для получения отчёта выбрать сценарий(-и) и нажать «Создать отчёт» – формируется PDF-файл в папке reports.

UML-диаграмма последовательности для процесса воспроизведения приведена в приложении Б.

3.2.4 Удаление сценариев

- выбрать один или несколько сценариев, нажать «Удалить сценарий», подтвердить действие – удаляются файлы сценариев и соответствующие папки со скриншотами.

3.3 Реализация основных модулей

В данном подразделе описаны ключевые алгоритмы разработанного приложения. Полные тексты модулей приведены в приложении Б. Здесь же представлены только наиболее важные фрагменты кода, иллюстрирующие уникальную логику системы.

3.3.1 Модуль модели (model.py)

Класс ScenarioModel управляет хранением сценариев, скриншотов и отчётов. Основные операции – загрузка, сохранение, валидация. Наиболее важным является метод валидации по JSON-схеме (листинг 1), который проверяет корректность сценария перед воспроизведением.

Листинг 1 – Валидация сценария по JSON-схеме

```
def validate_scenario(self, actions):  
    schema_path = os.path.join(os.path.dirname(__file__),  
                                'scenario_schema.json')  
    with open(schema_path, 'r', encoding='utf-8') as f:  
        schema = json.load(f)  
        jsonschema.validate(actions, schema)
```

Остальные методы модели (сохранение, загрузка, работа со скриншотами) тривиальны и не требуют детального разбора.

3.3.2 Модуль записи (recorder.py)

Класс Recorder перехватывает события мыши и клавиатуры через rinput. Ключевой алгоритм – распознавание одиночного клика, двойного клика и перетаскивания (см. листинг 2). Порог перемещения DRAG_THRESHOLD = 5 пикселей, интервал двойного клика – 0,5 секунды.

Листинг 2 – Определение типа мышинового действия

```
dist = ((x - start_x) ** 2 + (y - start_y) ** 2) ** 0.5
if dist < DRAG_THRESHOLD:
    # одиночный клик
    self.pending_click = {
        "type": "click",
        "parameters": {"x": start_x, "y": start_y, "button":
start_button.name}
    }
else:
    # перетаскивание
    self.actions.append({
        "type": "drag",
        "parameters": {"start_x": start_x, "start_y": start_y,
                        "end_x": x, "end_y": y, "button":
start_button.name}
    })
```

Запись нажатий клавиш реализована с сохранением виртуальных кодов (vk, scan) для точного воспроизведения независимо от раскладки. Специальные клавиши преобразуются в имена через словарь SPECIAL_KEYS_MAP.

3.3.3 Модуль воспроизведения (player.py)

Класс Player выполняет сценарий. Наиболее сложный алгоритм – посимвольный ввод текста с автоматическим переключением раскладки (см. листинг 3). Для каждого символа определяется необходимая раскладка

(русская/английская), при несовпадении переключение выполняется через Win+Space. Затем символ вводится через виртуальный код; если код отсутствует – через буфер обмена.

Листинг 3 – Ввод текста с переключением раскладки

```
for ch in text:
    is_cyrillic = bool(re.match('[а-яёА-ЯЁ]', ch))
    is_latin = bool(re.match('[a-zA-Z]', ch))
    vk = CHAR_TO_VK.get(ch)
    if vk is None:
        QApplication.clipboard().setText(ch)
        pyautogui.hotkey('ctrl', 'v')
        continue
    need_layout = 'ru' if is_cyrillic else ('en' if is_latin else
None)

    if need_layout and need_layout != current_layout:
        pyautogui.hotkey('win', 'space')
        current_layout = need_layout
        time.sleep(0.2)
    if ch.isupper():
        keyboard.press(KeyCode.from_vk(0xA0)) # Left Shift
        key_obj = KeyCode.from_vk(vk)
        keyboard.press(key_obj)
        keyboard.release(key_obj)
    if ch.islower():
        keyboard.release(KeyCode.from_vk(0xA0))
        time.sleep(0.05)
```

Сравнение скриншотов (см. листинг 4) – ещё один важный механизм. После фактического снимка вычисляется разность с эталоном. При наличии отличий генерируется diff-изображение с выделением изменённых областей (через OpenCV – красные прямоугольники; через Pillow – полупрозрачное наложение).

Листинг 4 – Сравнение скриншотов

```

actual = pyautogui.screenshot(region=region)
ref = Image.open(ref_path)
diff = ImageChops.difference(ref, actual)
if diff.getbbox():
    highlighted = self._highlight_differences(ref, actual)
    highlighted.save(diff_path)

```

3.3.4 Модуль генерации отчётов (reporter.py)

Класс ReportGenerator формирует PDF-отчёты через ReportLab. Для каждого шага выводится его тип, параметры, а для мышиных действий – скриншот с маркером. Для шагов screenshot эталон и фактическое изображение размещаются рядом (см. листинг 5).

Листинг 5 – Основной цикл генерации отчёта

```

c.setFont(FONT_BOLD, 12)
c.drawString(30, y, f"Шаг {i+1} ({action['type']})")
y -= 18
c.setFont(FONT_NAME, 10)
params_text = str(action.get('parameters', {}))
c.drawString(40, y, f"Параметры: {params_text}")
y -= 18
if action["type"] in ("click", "double_click", "drag"):
    mark_path = self._get_screenshot_path(scenario_name, i,
'clickmark')
    if os.path.exists(mark_path):
        y = self._draw_image(c, mark_path, 30, y, page_width,
page_height, caption="Действие с маркером")

```

3.3.5 DSL-парсер (dsl_parser.py)

Модуль преобразует текстовый DSL в JSON и обратно. Парсинг одной команды (см. листинг 6) демонстрирует ключевую идею: использование shlex.split для корректной обработки аргументов в кавычках.

Листинг 6 – Парсинг команды DSL

```

if cmd == 'click':
    x, y = int(parts[1]), int(parts[2])
    button = parts[3] if len(parts) > 3 else 'left'
    actions.append({'type': 'click', 'parameters': {'x': x, 'y':
y, 'button': button}}))

```

Обратное преобразование (`dump_dsl`) выполняется аналогично – для каждого типа действия генерируется текстовая строка.

3.3.6 Модули представления и контроллера (`view.py`, `controller.py`)

Графический интерфейс построен на PyQt5. Контроллер `AppController` связывает интерфейс с моделью и сервисами. Ключевое архитектурное решение – групповое воспроизведение в отдельном потоке (см. листинг 7), чтобы не блокировать графический интерфейс.

Листинг 7 – Групповое воспроизведение сценариев в отдельном потоке

```

def _run_playback_multiple(self, names):
    for name in names:
        actions = self.model.load_scenario(name)
        self.model.clear_temp_screenshots(name)
        success = self.player.play(name, actions)
        if not success:
            self.view.status_label.setText(f"Ошибка при
воспроизведении '{name}'")
            self._show_tray_notification("Автоматизация GUI",
f"Ошибка при воспроизведении '{name}'")
            return
        self.view.status_label.setText("Все выбранные сценарии
успешно воспроизведены")

```

Контроллер также автоматически определяет формат сценария (JSON или DSL) по расширению файла и переключает отображение в редакторе. Ручное добавление действий (текст, CLI, ожидание) реализовано как через активную запись, так и напрямую в файл.

Теперь благодаря такой реализации приложение можно тестировать и расширять.

4 Тестирование приложения

4.1 Цели и задачи тестирования

Тестирование ПО представляет собой процесс проверки соответствия разработанного продукта предъявленным требованиям, выявления дефектов и оценки качества. Для приложения «Автоматизация тестирования GUI» тестирование решает следующие задачи:

- проверка функциональной полноты — подтверждение того, что реализованы все заявленные в техническом задании функции (запись, воспроизведение, сравнение скриншотов, генерация отчётов, управление сценариями);
- оценка корректности работы — выявление ошибок в реализации алгоритмов (распознавание двойных кликов, перетаскиваний, выделение области скриншота, эмуляция ввода);
- проверка надёжности и отказоустойчивости — анализ поведения приложения при некорректных действиях пользователя (синтаксические ошибки в сценариях, отсутствие эталонных скриншотов, недоступность файловой системы);
- оценка удобства пользовательского интерфейса — проверка эргономики, реакции на действия, работы системного трей, множественного выбора;
- верификация целостности данных — контроль корректности сохранения сценариев, скриншотов и отчётов в файловой системе;
- проверка совместимости — тестирование на целевой операционной системе Windows 10/11 с различными разрешениями экрана и масштабированием.

4.2 Виды тестирования

В процессе тестирования разработанного приложения применялись следующие виды тестирования:

- функциональное тестирование (black-box) — проверка соответствия каждой функции требованиям без анализа внутреннего кода. Тестировались: создание,

редактирование, сохранение, удаление сценариев, переключение форматов, запись и воспроизведение всех типов действий, сравнение скриншотов, генерация отчётов, работа системного трей;

- интеграционное тестирование — проверка взаимодействия между модулями: контроллер — модель, контроллер — рекордер, контроллер — плеер, плеер — модель (сохранение скриншотов), генератор отчётов — модель. Особое внимание уделялось передаче данных между слоями (например, корректности JSON-сценария при передаче из рекордера в модель);

- модульное тестирование (unit-тестирование) — изоляционная проверка отдельных методов: парсинг DSL, валидация JSON, преобразование форматов, алгоритм выделения области скриншота. Проводилось на этапе разработки с использованием отладчика и тестовых функций;

- тестирование пользовательского интерфейса (GUI-тестирование) — ручная проверка расположения элементов, реакции на нажатия кнопок, обновления списка сценариев, работы строки состояния, системного трей, диалоговых окон;

- тестирование на отказоустойчивость — имитация ошибочных ситуаций: ввод некорректного JSON, попытка воспроизведения сценария с неверным типом действия, отсутствие эталонного скриншота, прерывание записи, удаление файлов сценария во время работы приложения;

- регрессионное тестирование — повторный прогон ранее выполненных тестов после исправления ошибок для подтверждения, что новые изменения не нарушили существующую функциональность;

- тестирование удобства использования — оценка логичности интерфейса, компактности расположения кнопок, понятности сообщений об ошибках, наличия контекстной подсказки в трейе.

4.3 План тестирования

Для систематизации процесса и фиксации результатов был составлен план тестирования, включающий описание тестовых сценариев, ожидаемых и фактических результатов.

План тестирования представлен в таблице 2. Для каждого тестового сценария указаны уникальный идентификатор, наименование, шаги выполнения, ожидаемый результат, фактический результат и статус прохождения.

Таблица 2 – План тестирования приложения «Автоматизация тестирования GUI»

Идентификатор	Наименование теста	Шаги выполнения	Ожидаемый результат	Фактический результат	Статус
TC-01	Создание нового сценария	1. Нажать кнопку «Новый сценарий». 2. Ввести имя «Test1».	В списке сценариев появляется «Test1», в папке scenarios создаётся файл Test1.test нулевого размера.	Сценарий создан, файл присутствует.	+
TC-02	Сохранение и валидация корректного сценария	1. Выбрать сценарий «Test1». 2. Ввести в редакторе DSL-команду click 100 200 left. 3. Нажать «Сохранить сценарий».	Сценарий сохраняется без ошибок, содержимое файла Test1.test соответствует введённой строке.	Сохранение выполнено, содержимое совпадает.	+
TC-03	Валидация некорректного сценария	1. В редакторе ввести неполную команду click (без координат). 2. Нажать «Сохранить сценарий».	Появляется сообщение об ошибке валидации с указанием проблемной строки и типа ошибки.	Выведено сообщение «Ошибка DSL: недостаточно аргументов для click».	+

Идентификатор	Наименование теста	Шаги выполнения	Ожидаемый результат	Фактический результат	Статус
TC-04	Переключение формата DSL / JSON	1. Выбрать сценарий с DSL-содержимым. 2. Нажать кнопку «DSL / JSON».	Текст в редакторе преобразуется в JSON-формат. Повторное нажатие возвращает исходный	Преобразование выполнено без потери данных, формат соответствует.	+

			DSL.		
ТС-05	Удаление одного сценария	1. Выбрать сценарий «Test1». 2. Нажать «Удалить сценарий», подтвердить действие.	Сценарий исчезает из списка, файл Test1.test удалён из папки scenarios.	Сценарий удалён, файла нет.	+
ТС-06	Групповое удаление нескольких сценариев	1. Создать сценарии «Test2» и «Test3». 2. Выделить оба сценария. 3. Нажать «Удалить сценарий», подтвердить.	Оба сценария удалены из списка и из файловой системы.	Удаление выполнено.	+
ТС-07	Запись одиночного клика мыши	1. Создать сценарий «RecordTest». 2. Нажать «Записать». 3. В произвольной точке экрана кликнуть левой кнопкой мыши. 4. Нажать «Остановить запись».	В сценарии появилась команда click с координатами точки клика.	Команда присутствует, координаты верны.	+
ТС-08	Запись двойного клика мыши	1. В режиме записи выполнить два быстрых клика в одной точке с интервалом $\leq 0,5$ с.	В сценарии появляется команда double_click с координатами и кнопкой.	Двойной клик распознан и записан.	+
ТС-09	Запись перетаскивания	1. В режиме записи зажать левую кнопку мыши, переместить курсор на расстояние ≥ 5 пикселей, отпустить.	В сценарии появляется команда drag с параметрами start_x, start_y, end_x, end_y, button.	Команда записана корректно.	+

Идентификатор	Наименование теста	Шаги выполнения	Ожидаемый результат	Фактический результат	Статус
ТС-10	Выделение области скриншота	1. В режиме записи зажать клавишу Alt. 2. Переместить мышь, выделив прямоугольную область на экране 3. Отпустить Alt.	В сценарий добавляется команда screenshot с параметрами области. В папке screenshots/RecordTest сохраняется эталонный скриншот с суффиксом _ref.png.	Действие добавлено, скриншот сохранён.	+
ТС-11	Воспроизведение одиночного сценария	1. Выбрать сценарий «RecordTest». 2. Нажать «Воспроизвести».	Окно приложения скрывается, через 1 секунду курсор выполняет	Воспроизведение успешно.	+

			записанные действия. По окончании появляется уведомление в трее «Сценарий 'RecordTest' успешно воспроизведён».		
ТС-12	Групповое воспроизведение сценариев	1. Выбрать два сценария «RecordTest» и «ScreenshotTest». 2. Нажать «Воспроизвести».	Сценарии выполняются последовательно: после завершения первого начинается второй. В строке состояния отображается имя текущего сценария. По окончании – уведомление.	Последовательное воспроизведение подтверждено.	+

Идентификатор	Наименование теста	Шаги выполнения	Ожидаемый результат	Фактический результат	Статус
ТС-13	Генерация PDF-отчёта для одного сценария	1. Выбрать сценарий «RecordTest». 2. Нажать «Создать отчёт».	В папке reports появляется PDF-файл с именем RecordTest_<timestamp>.pdf. Отчёт содержит заголовок, список шагов, для мышинных действий – скриншот с маркером, для шага screenshot – эталонное и фактическое изображения рядом.	Отчёт сформирован, все элементы на месте.	+
ТС-14	Генерация сводного отчёта для нескольких сценариев	1. Выбрать два сценария. 2. Нажать «Создать отчёт».	Формируется PDF с именем multiple_<timestamp>.pdf. Каждый сценарий начинается с новой страницы с собственным заголовком.	Сводный отчёт создан.	+

ТС-15	Сворачивание и восстановление окна из трее.	1. Закрыть главное окно (нажать крестик). 2. Дважды кликнуть по иконке приложения в трее.	Приложение не завершается, есть иконка в трее. Главное окно восстанавливается, становится активным.	Сворачивание и восстановление работает корректно.	+
-------	---	--	---	---	---

4.4 Среда и инструменты тестирования

Тестирование проводилось в следующей среде:

- операционная система: Windows 10 IoT Корпоративная LTSC (версия 21H2), сборка 19044;
- аппаратное обеспечение: Intel Core i5-5500 (4 ядра, 3,2 ГГц), 24 ГБ ОЗУ, SSD-накопитель 1 ТБ;
- разрешение экрана: 1920×1080, масштабирование 100 %;
- тестируемые приложения: «Блокнот» (стандартный, версия 10.0.19041.1).

4.5 Критерии начала и окончания тестирования

Критерии начала тестирования:

- завершение написания всех функциональных модулей;
- успешная сборка standalone-приложения (исполняемый файл, созданный PyInstaller);
- наличие тестовой среды (компьютер с Windows 10/11, установленные тестируемые приложения);
- подготовка тестовых данных (набор сценариев для проверки различных типов действий).

Критерии окончания тестирования:

- выполнение всех тестовых сценариев из плана (ТС-01 – ТС-15);
- отсутствие критических ошибок (блокирующих выполнение основных функций);
- все обнаруженные дефекты зарегистрированы и исправлены, либо для не критических дефектов принято обоснованное решение об отложении;

— проведён один успешный прогон регрессионного тестирования после исправлений.

Данный план тестирования позволил систематически проверить все ключевые функции приложения и обеспечить высокое качество программного продукта. Результаты выполнения тестов представлены в последующих подразделах. На рисунке 8 представлено главное окно приложения.

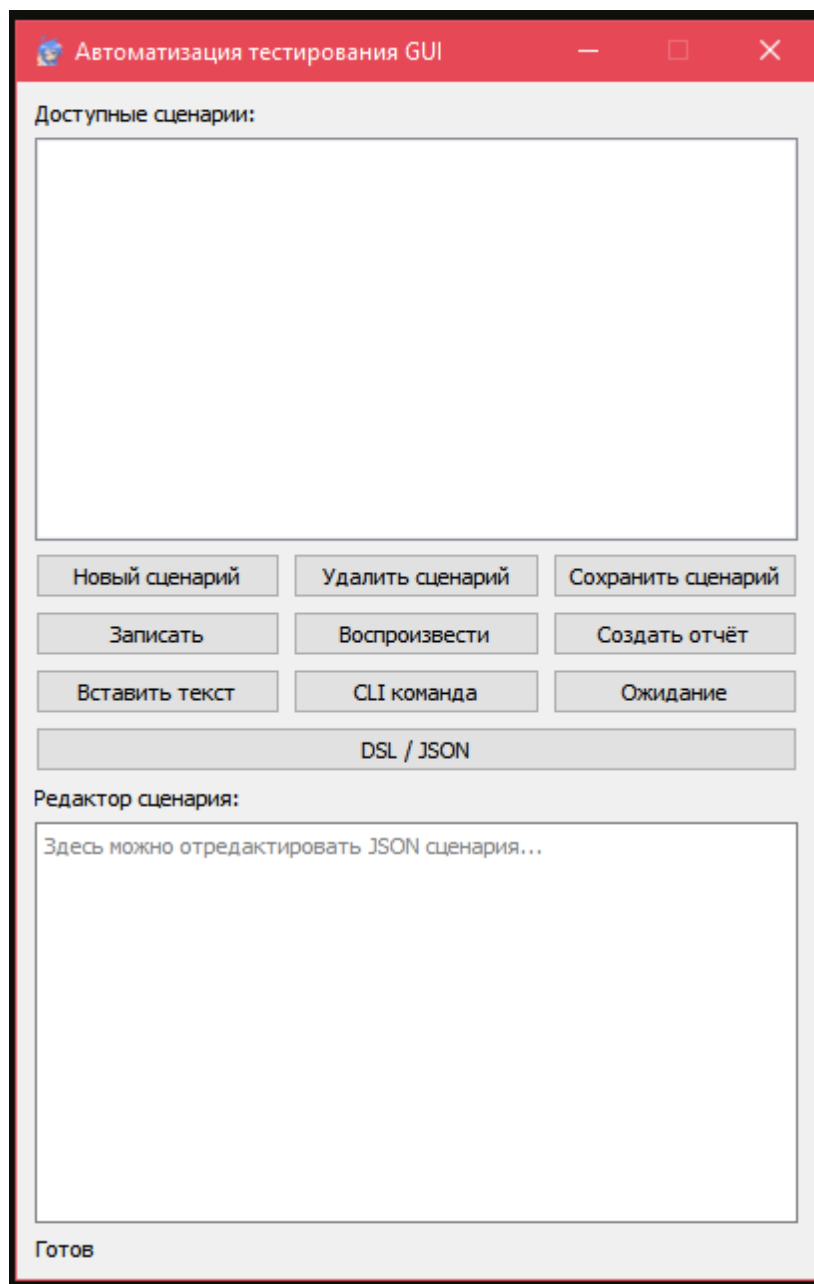


Рисунок 8 – Окно приложения «Автоматизация тестирования GUI»

4.6 Проверка тестовых сценариев по плану тестирования

4.6.1 Проверка возможностей управления сценариями

4.6.1.1 Тестовый сценарий 1. Был создан новый сценарий, нажатием кнопки «Новый сценарий» сценарию дано имя «Test1». Сценарий создан и отображается в списке доступных сценариев (см. рисунок 9).

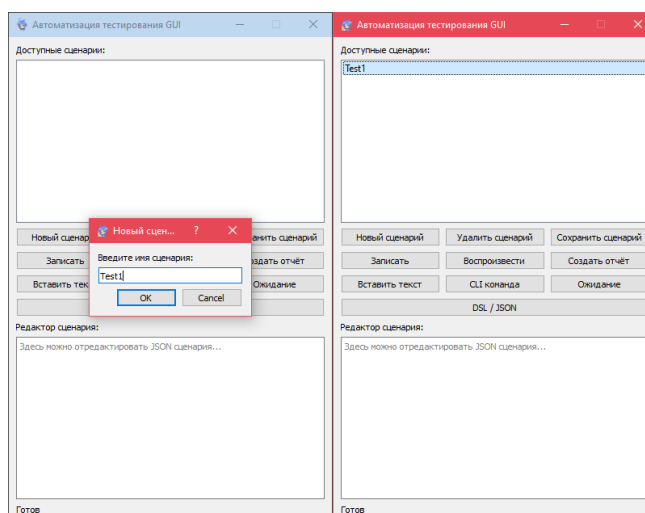


Рисунок 9 – Создание нового сценария

4.6.1.2 Тестовый сценарий 2. В существующем сценарии «Test1», в редакторе сценария, была введена команда `click 100 200 left`. После чего изменения были сохранены нажатием кнопки «Сохранить сценарий».

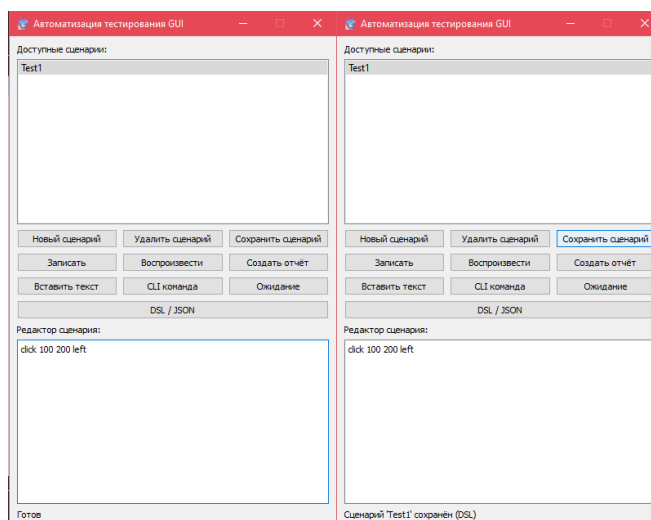


Рисунок 10 – Редактирование и сохранение сценария

4.6.1.3 Тестовый сценарий 3. Для проверки корректной работы валидации сценария была введена неполная команда `click`, не содержащая аргументов. При попытке сохранить сценарий появляется сообщение об ошибке валидации.

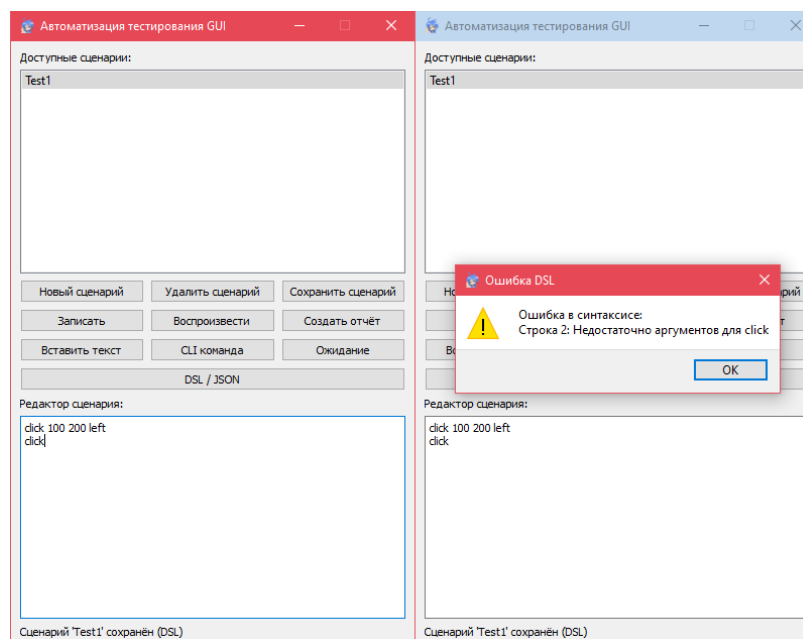


Рисунок 11 – Ошибка валидации сценария

4.6.1.4 Тестовый сценарий 4. Формат существующего сценария «Test1» в редакторе сценария был изменен нажатием кнопки «DSL / JSON». Формат изменился с DSL на JSON. После, нажатием той же кнопки, формат был изменен обратно, то есть с JSON на DSL.

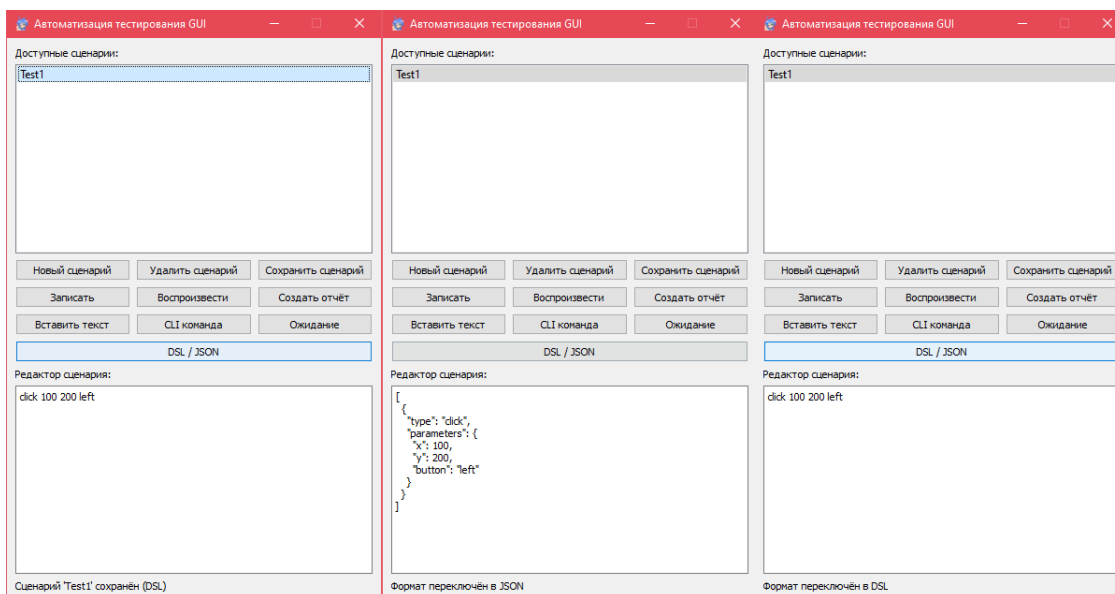


Рисунок 12 – Переключение формата сценария (DSL/JSON)

4.6.1.5 Тестовый сценарий 5. Существующий сценарий «Test1» был удален нажатием кнопки «Удалить сценарий». В высветившемся окне подтверждения удаления сценария была нажата кнопка «Yes». Сценарий удален и не отображается в списке доступных сценариев.

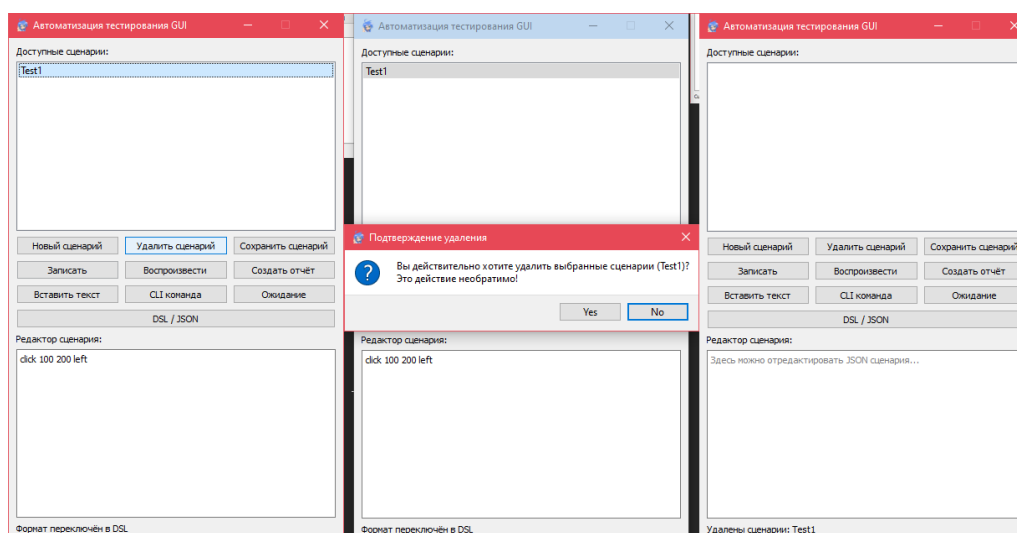


Рисунок 13 – Удаление одного сценария

4.6.1.6 Тестовый сценарий 6. Для проверки корректности работы множественного удаления сценариев, были созданы два новых сценария «Test2» и «Test3». После чего они были выбраны и удалены нажатием кнопки «Удалить сценарий». Сценарии удалены и не отображаются в списке доступных сценариев.

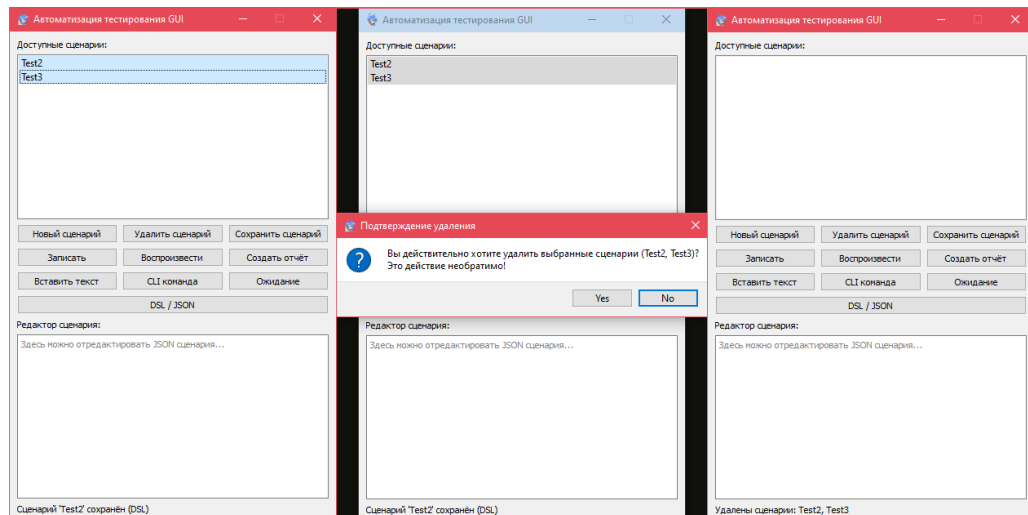


Рисунок 14 – Удаление нескольких сценариев

4.6.2 Проверка работы записи сценариев

4.6.2.1 Тестовый сценарий 7. Создан новый сценарий «RecordTest», в него было добавлено действия клика мыши. После нажатия кнопки «Записать» окно приложения свернулось, был совершен клик, после приложение было развернуто и записанное действие было сохранено нажатием кнопки «Остановить запись».

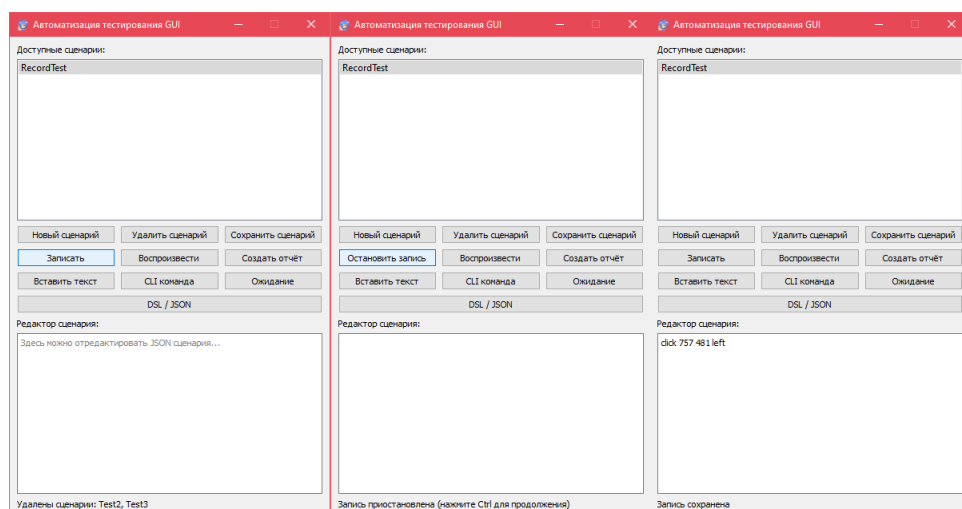


Рисунок 15 – Запись клика мыши

4.6.2.2 Тестовый сценарий 8. В существующий сценарий «RecordTest» было добавлено действие двойного клика мыши аналогично предыдущему сценарию. Действие двойного клика мыши было сохранено и отображается в редакторе сценария.

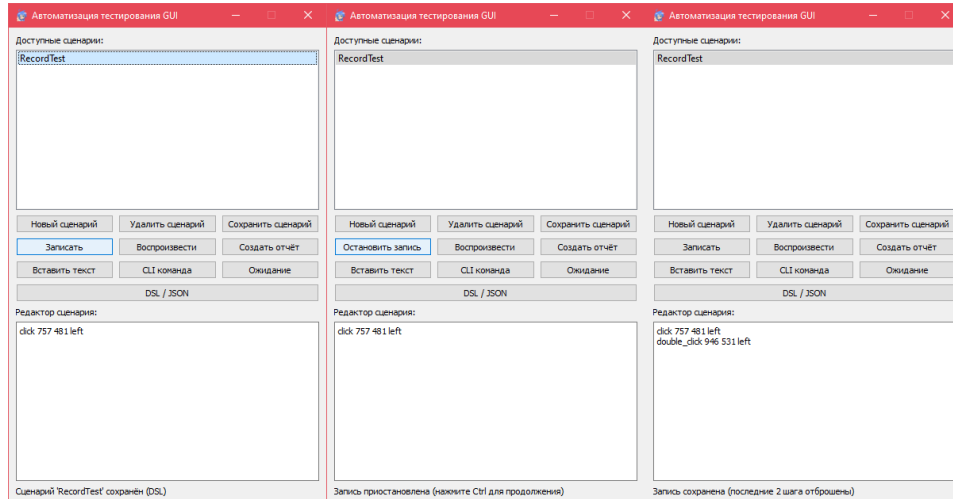


Рисунок 16 – Запись двойного клика мыши

4.6.2.3 Тестовый сценарий 9. В существующий сценарий «RecordTest» было добавлено действие перетаскивания мыши аналогично предыдущему сценарию. Действие перетаскивания мыши было сохранено и отображается в редакторе сценария.

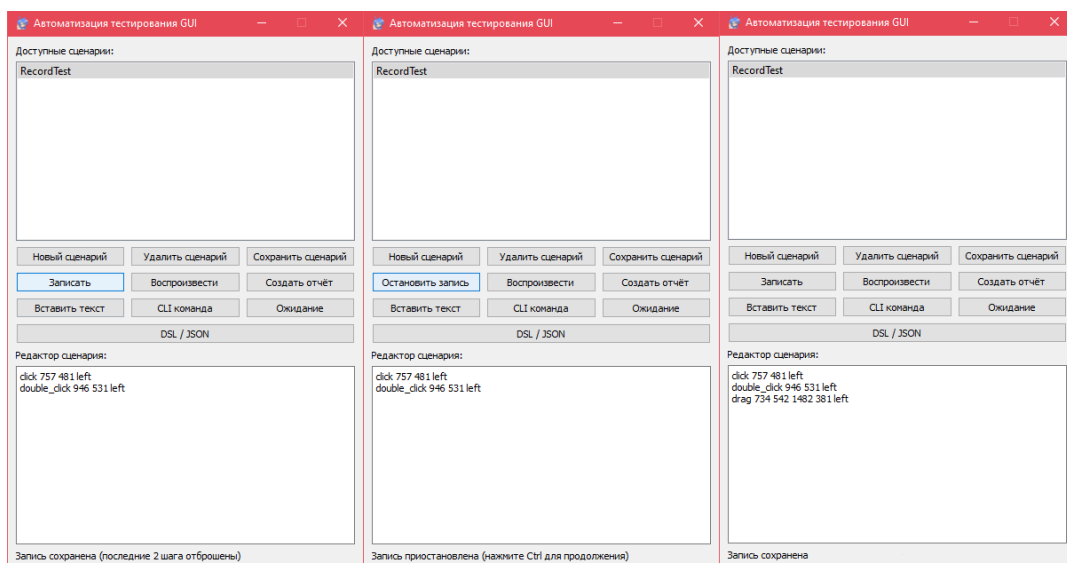


Рисунок 17 – Запись перетаскивания мыши

4.6.2.4 Тестовый сценарий 10. В режиме записи, в существующий сценарий «RecordTest» было добавлено действие скриншота, используя выделение области экрана клавишей Alt.

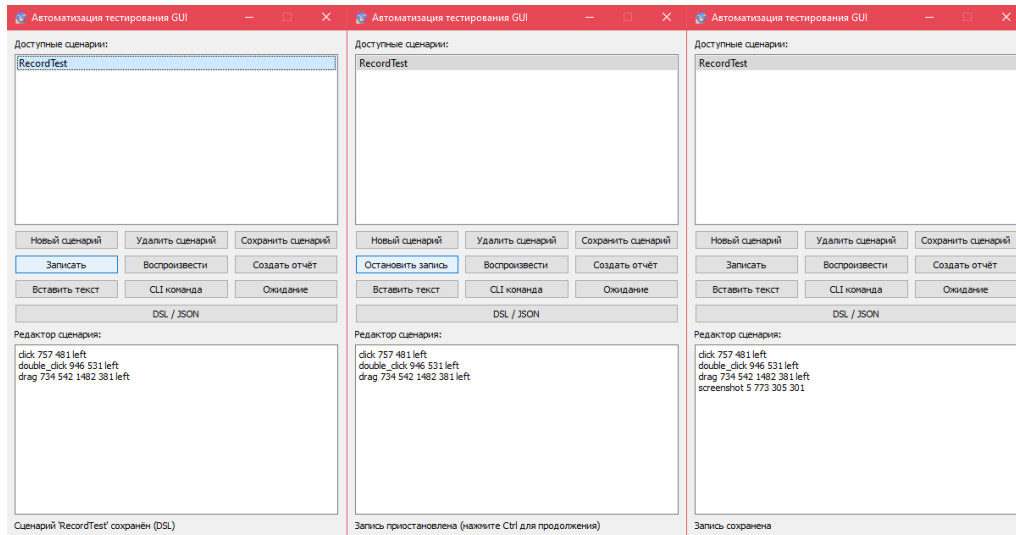


Рисунок 18 – Запись скриншота области экрана

4.6.3 Проверка работы воспроизведения сценариев и генерации отчётов

4.6.3.1 Тестовый сценарий 11. В списке доступных сценариев был выбран существующий сценарий «RecordTest», записанные в нём действия были воспроизведены нажатием кнопки «Воспроизвести». Все действия сценария были воспроизведены корректно.

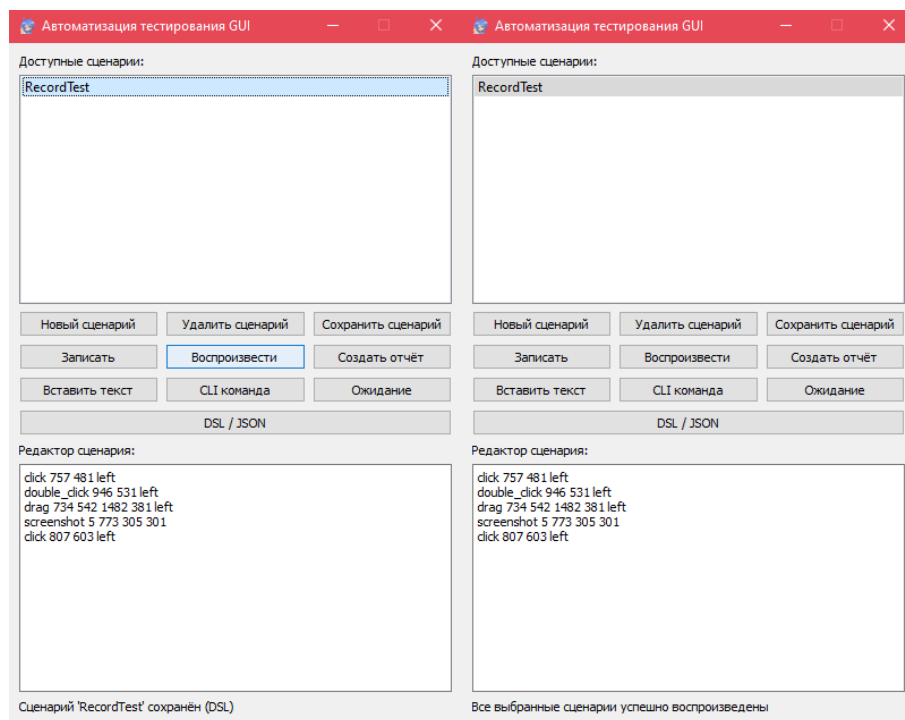


Рисунок 19 – Воспроизведения сценария

По окончании воспроизведения сценария, на экране появилось уведомление об успешном выполнении сценария.

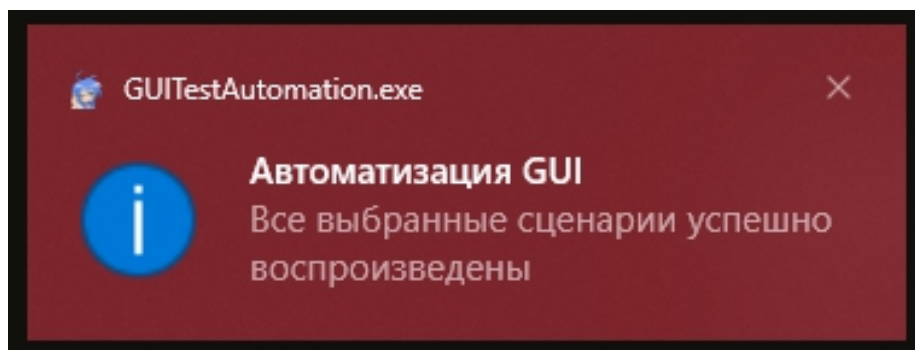


Рисунок 20 – Уведомление об успешном воспроизведении сценария

4.6.3.2 Тестовый сценарий 12. В списке доступных сценариев были выбраны существующие сценарии «RecordTest» и «ScreenshotTest», записанные в них действия были воспроизведены нажатием кнопки «Воспроизвести». Действия выбранных сценариев были воспроизведены в порядке списка и корректно.

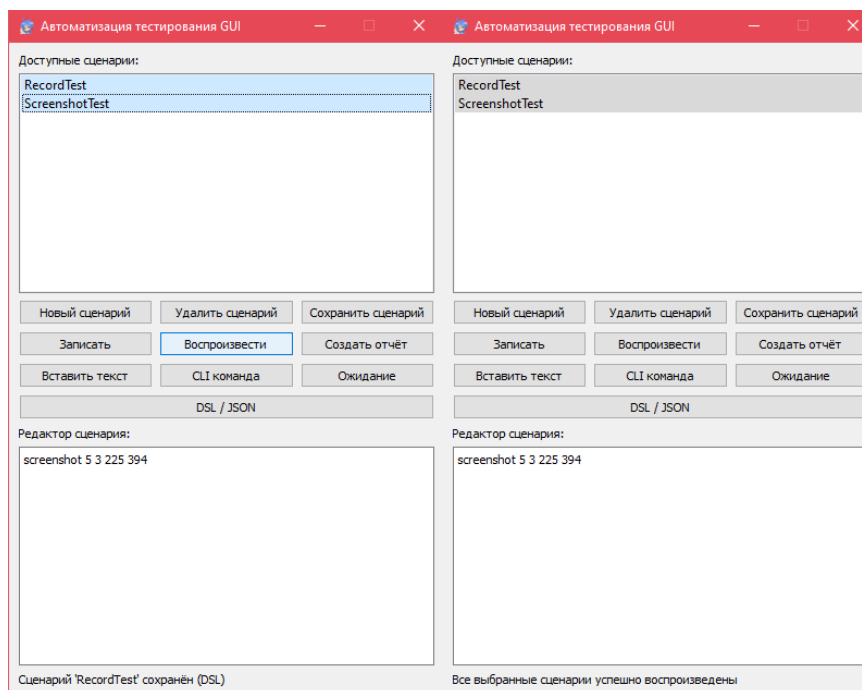


Рисунок 21 – Воспроизведение нескольких сценариев

После воспроизведения группы сценариев, на экране появилось уведомление об их успешном выполнении.

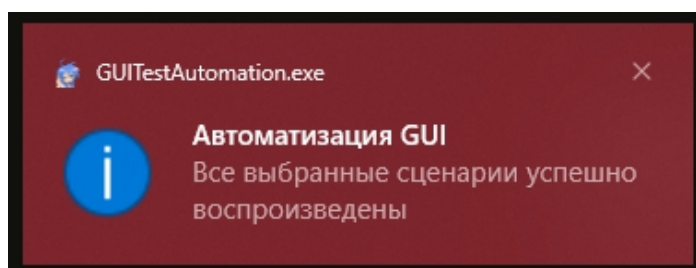


Рисунок 22 – Уведомление об успешном воспроизведении сценариев

4.6.3.3 Тестовый сценарий 13. В списке доступных сценариев был выбран существующий сценарий «RecordTest», для него был сгенерирован сводный отчёт нажатием кнопки «Создать отчёт». При завершении генерации отчёта высветилось окно, в котором указан путь до него.

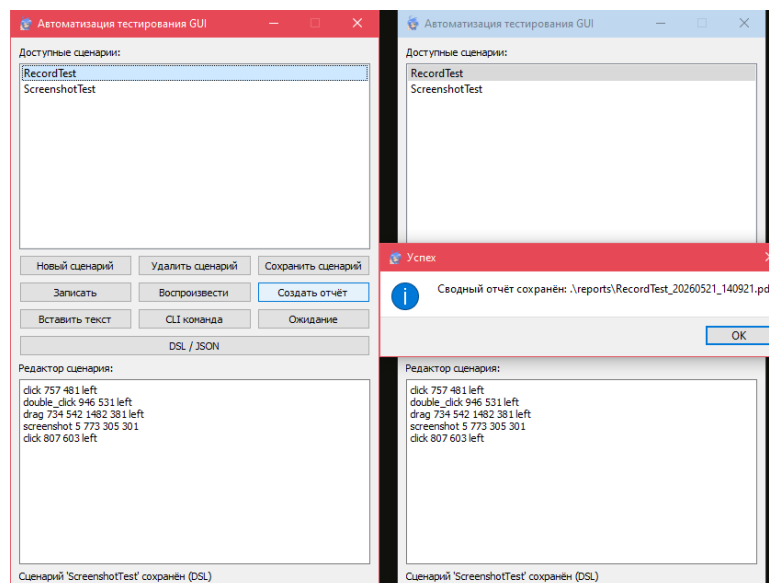


Рисунок 23 – Генерация отчета для сценария

После файл отчёта был открыт, чтобы проверить корректность форматирования внутри него. Отчёт создавался корректно, все необходимые элементы в нём отображены.

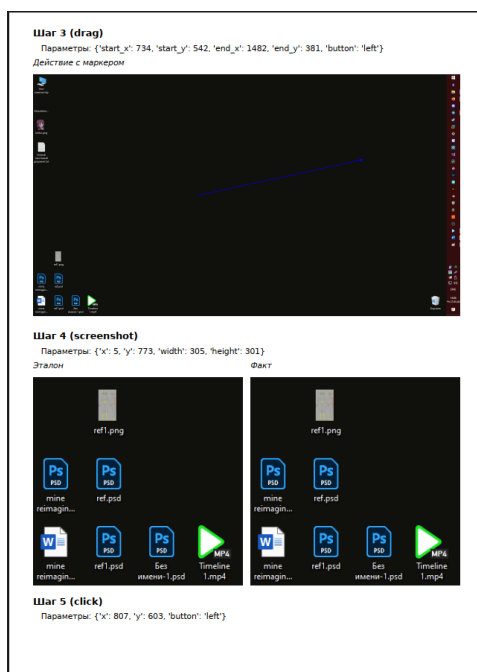


Рисунок 24 – Файл сгенерированного отчета для сценария «RecordTest»

4.6.3.4 Тестовый сценарий 14. В списке доступных сценариев были выбраны существующие сценарии «RecordTest» и «ScreenshotTest», для них был

сгенерирован сводный отчёт нажатием кнопки «Создать отчёт». При завершении генерации отчёта высветилось окно, в котором указан путь до него.

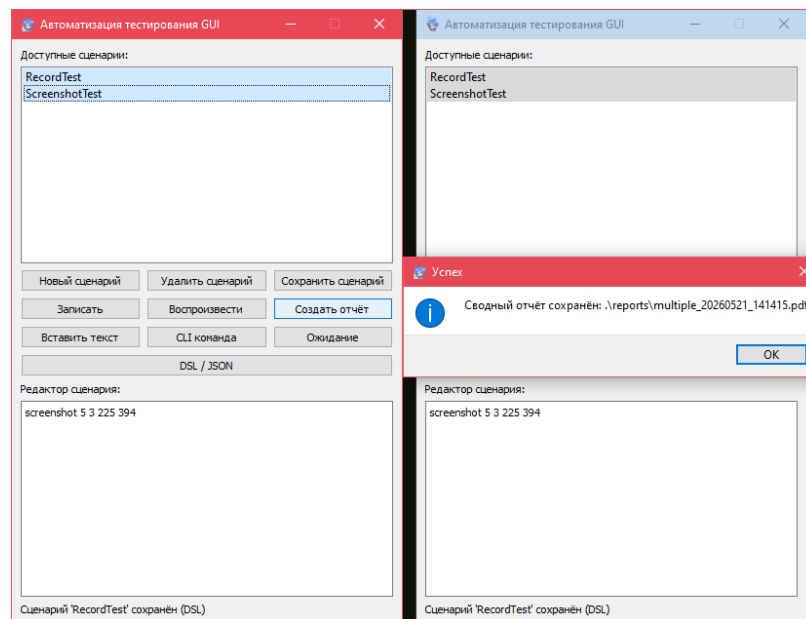


Рисунок 25 – Генерация отчета для группы сценариев

Внутри файла сводного отчёта есть отчётливое разделение на блоки разных сценариев.

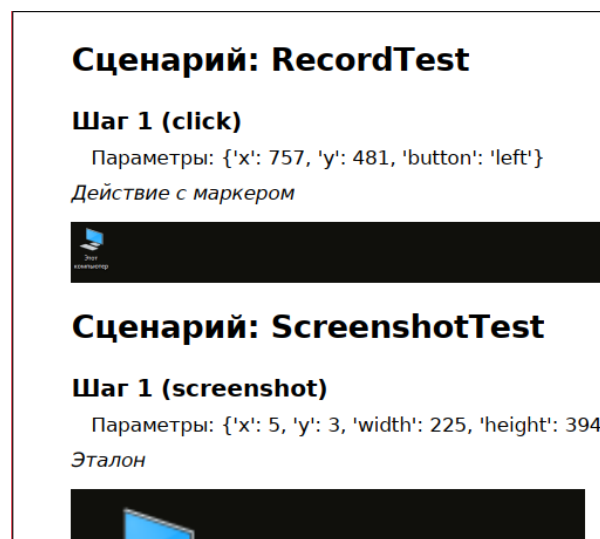


Рисунок 26 – Заголовки начала отчётов для разных сценариев

4.6.4 Тестовый сценарий 15. Для проверки корректности работы системного трея приложение было закрыто нажатием кнопки «Заккрыть». Приложение не завершило свою работу, иконка приложения отображается в трее.

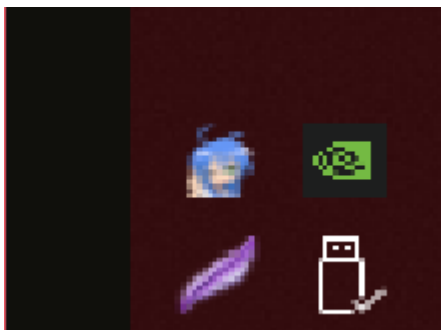


Рисунок 27 – Сворачивание приложения в системный трей

4.7 Вывод

В ходе тестирования разработанного приложения были проверены все ключевые функциональные модули:

- управление сценариями (создание, загрузка, сохранение, удаление, переключение форматов);
- запись пользовательских действий (клики, двойные клики, перетаскивание, ввод текста, нажатия клавиш, выделение области скриншота, пауза, дозапись);
- воспроизведение сценариев (эмуляция мыши и клавиатуры, выполнение CLI-команд, сравнение скриншотов, групповое воспроизведение);
- генерация PDF-отчётов (одиночные и сводные отчёты с визуализацией маркеров и различий);
- пользовательский интерфейс и системный трей.

При разработке приложения было выявлено достаточное количество ошибок. Но протестировав ПО получилось исправить и обработать большую часть из них, это позволило сделать более надежное и отказоустойчивое приложение.

5 Безопасность жизнедеятельности

При разработке и использовании программного обеспечения для автоматизации тестирования графических интерфейсов пользователь (разработчик или тестировщик) проводит значительное количество времени за персональным компьютером. Длительная работа с вычислительной техникой связана с воздействием ряда вредных и опасных факторов, которые могут негативно влиять на здоровье. Для минимизации этих рисков необходимо соблюдать установленные правила и нормативы.

В данном разделе рассмотрены основные требования безопасности при работе с ПК, эргономические рекомендации по организации рабочего места, а также режим труда и отдыха.

5.1 Правила и техника безопасности при работе с ПК

Человек при взаимодействии с вычислительной техникой может подвергаться следующим вредным и опасным воздействиям:

- поражение электрическим током;
- электромагнитное излучение;
- опасность пожара;
- повышенный уровень шума (от кулеров и систем охлаждения);
- зрительное и мышечное утомление.

Для снижения влияния этих факторов требуется соблюдать ряд правил.

Освещение рабочего места. Помещение должно иметь равномерное естественное и искусственное освещение. Свет не должен создавать бликов на экране монитора и попадать прямо в глаза. Рекомендуется использовать рассеянный свет, а для локального освещения – настольную лампу с регулируемым направлением света, установленную сбоку от монитора.

Эргономика рабочего места. Стул должен быть офисным, с регулируемой высотой сиденья и наклоном спинки. Ноги пользователя должны полностью стоять на полу, руки – естественно лежать на столе. Монитор располагается на расстоянии

вытянутой руки (примерно 50–70 см от глаз), его верхняя граница находится на уровне глаз или чуть ниже. Наклон экрана должен исключать отражения от источников света. Клавиатуру рекомендуется размещать на расстоянии 10–15 см от края стола, мышь – на том же уровне, что и клавиатура.

Электробезопасность и пожарная безопасность. Запрещается использовать компьютер при наличии повреждённых кабелей, розеток или видимых дефектов корпуса. Не следует оставлять включённое оборудование без присмотра на длительное время. Рабочие места должны быть оснащены заземлением, а электропроводка – соответствовать нормам. В помещении необходимо иметь углекислотный огнетушитель для тушения электрооборудования.

Режим труда и отдыха. При 8-часовом рабочем дне регламентированные перерывы должны составлять от 30 до 90 минут в сумме. Каждый час рекомендуется делать короткий перерыв (5–10 минут) для разминки и гимнастики для глаз. Это снижает нагрузку на зрение, предотвращает развитие туннельного синдрома и заболеваний опорно-двигательного аппарата.

5.2 Организация рабочего места и профилактика утомления

Правильная организация рабочего места способствует поддержанию высокой работоспособности и сохранению здоровья. Рекомендуется:

- располагать монитор задней стенкой к стене, чтобы уменьшить электромагнитное излучение в сторону других людей;
- использовать клавиатуру и мышь с низким профилем, чтобы кисти рук не перегибались;
- регулярно проветривать помещение и поддерживать температуру воздуха в пределах 20–24 °С, влажность – 40–60 %;
- выполнять упражнения для глаз: переводить взгляд с ближнего предмета на дальний, делать круговые движения, часто моргать.

Особое внимание следует уделить профилактике зрительного утомления. При работе с GUI-тестированием пользователь часто всматривается в мелкие элементы

интерфейса и сравнивает скриншоты. Для снижения нагрузки рекомендуется использовать тёмную тему оформления, настраивать масштаб интерфейса, а также периодически менять фокус взгляда.

Таким образом, соблюдение перечисленных правил и рекомендаций позволяет свести к минимуму негативное влияние факторов, связанных с длительной работой за компьютером, обеспечивает комфортные и безопасные условия труда как для разработчика, так и для конечного пользователя приложения.

Заключение

Результатом выполнения данной выпускной квалификационной работы можно считать успешно разработанное приложение «Автоматизация тестирования GUI», полностью соответствующее заданию. Также были выполнены следующие задачи:

- полный анализ предметной области и существующих аналогов (AutoIt, SikuliX, Katalon Studio);
- выбор языка программирования (Python), среды разработки (OpenIDE) и формата хранения сценариев (JSON и DSL);
- анализ архитектуры разрабатываемого приложения с применением паттерна ModelViewController;
- обоснование и выбор необходимых библиотек (PyQt5, rpy2, pyautogui, Pillow, OpenCV, ReportLab, jsonschema);
- непосредственная разработка приложения, включая модули записи, воспроизведения, генерации отчётов и DSL-парсер;
- тестирование программного обеспечения (управление сценариями, запись, воспроизведение, сравнение скриншотов, генерация PDF-отчётов, работа системного трей).

В результате было создано полностью протестированное standalone-приложение, написанное на языке программирования Python с использованием графического фреймворка PyQt5. Также в ходе разработки были получены навыки работы с библиотеками перехвата и эмуляции ввода (rpy2, pyautogui), обработки изображений (Pillow, OpenCV), генерации PDF-отчётов (ReportLab) и валидации JSON-сценариев (jsonschema). Таким образом результаты показали, что приложение полностью соответствует всем поставленным задачам, имеет удобный пользовательский интерфейс с поддержкой системного трей и выполняет свою работу корректно.

В перспективе приложение можно будет улучшить, добавив следующие возможности:

- автоматический поиск и клик по области скриншота на экране с использованием расширенных алгоритмов компьютерного зрения (уже заложено через OpenCV);
- поддержку удалённого запуска тестов по сети и интеграцию с CI/CD-системами (Jenkins, GitLab CI);
- расширенную аналитику и статистику прохождения тестов (графики, диаграммы);
- веб-интерфейс для просмотра результатов тестирования.

Данное приложение готово к использованию в процессе регрессионного тестирования графических интерфейсов настольных Windows-приложений и поможет значительно сократить временные затраты на ручные проверки, повысить надёжность тестирования и обеспечить наглядную отчётность о результатах.

Список использованных источников

1. Лутц М. Изучаем Python. – 5-е изд. – СПб.: Питер, 2019. – 1312 с.
2. Саммерфилд М. Программирование на Python 3. – М.: ДМК Пресс, 2018. – 608 с.
3. Sweigart A. Automate the Boring Stuff with Python. – 2nd ed. – No Starch Press, 2019. – 592 p.
4. PyAutoGUI documentation [Электронный ресурс]. – URL: <https://pyautogui.readthedocs.io/> (дата обращения: 22.04.2024).
5. Pynput documentation [Электронный ресурс]. – URL: <https://pynput.readthedocs.io/> (дата обращения: 25.04.2024).
6. Riverbank Computing. PyQt5 Reference Guide [Электронный ресурс]. – URL: <https://www.riverbankcomputing.com/static/Docs/PyQt5/> (дата обращения: 30.04.2024).
7. Куликов С. С. Тестирование программного обеспечения. – М.: Четыре четверти, 2017. – 320 с.
8. Савин Р. Тестирование Дот Ком. – М.: Дело, 2007. – 312 с.
9. ReportLab. PDF Generation User Guide [Электронный ресурс]. – URL: <https://www.reportlab.com/docs/reportlab-userguide.pdf> (дата обращения: 03.05.2024).
10. OpenCV. OpenCV-Python Tutorials [Электронный ресурс]. – URL: https://docs.opencv.org/master/d6/d00/tutorial_py_root.html (дата обращения: 06.05.2024).
11. JSON Schema Validation [Электронный ресурс]. – URL: <https://json-schema.org/> (дата обращения: 08.05.2026).

Приложение А

(Обязательное)

Приложение записи и воспроизведения сценариев тестирования графических
пользовательских интерфейсов

UML-диаграммы

Приложение А.1 – Архитектура приложения (диаграмма компонентов)

Приложение А.2 – Диаграмма вариантов использования

Приложение А.3 – Диаграмма последовательности записи

Приложение А.4 – Диаграмма последовательности редактирования

Приложение А.5 – Диаграмма последовательности воспроизведения

Приложение Б

(Обязательное)

Приложение записи и воспроизведения сценариев тестирования графических
пользовательских интерфейсов

Листинг программы

Файл main.py

```
import sys
from PyQt5.QtWidgets import QApplication
from controller import AppController

if __name__ == '__main__':
    app = QApplication(sys.argv)
    controller = AppController()
    controller.show()
    sys.exit(app.exec_())
```

Файл model.py

```
import json
import os
import glob
from datetime import datetime
import jsonschema
from PIL import Image
from dsl_parser import parse_dsl, dump_dsl

class ScenarioModel:
    """Хранение и управление сценариями, скриншотами и
отчётами."""
    def __init__(self, base_dir='.'):
        self.base_dir = base_dir
        self.scenarios_dir = os.path.join(base_dir, 'scenarios')
        self.screenshots_dir = os.path.join(base_dir,
'screenshots')
        self.reports_dir = os.path.join(base_dir, 'reports')
        os.makedirs(self.scenarios_dir, exist_ok=True)
        os.makedirs(self.screenshots_dir, exist_ok=True)
        os.makedirs(self.reports_dir, exist_ok=True)

    def save_scenario(self, name, actions, fmt='dsl'):
        """Сохраняет сценарий. fmt может быть 'dsl' или
'json'."""
        # Удаляем старые файлы с прежними расширениями
        for ext in ('.test', '.json'):
            old_path = os.path.join(self.scenarios_dir, name +
ext)
            if os.path.exists(old_path):
                os.remove(old_path)
        if fmt == 'dsl':
            path = os.path.join(self.scenarios_dir, name +
'.test')
            text = dump_dsl(actions)
        else:
            path = os.path.join(self.scenarios_dir, name +
'.json')
            text = json.dumps(actions, indent=2,
ensure_ascii=False)
```

```

        with open(path, 'w', encoding='utf-8') as f:
            f.write(text)

    def load_scenario(self, name):
        """Загружает сценарий по имени (без расширения)."""
        path = self._get_scenario_files(name)
        if not path:
            raise FileNotFoundError(f"Сценарий '{name}' не найден
(ни .test, ни .json)")
        with open(path, 'r', encoding='utf-8') as f:
            content = f.read()
        if path.endswith('.json'):
            actions = json.loads(content)
        else: # .test
            actions = parse_dsl(content)
        # Валидация (через схему) должна быть вызвана отдельно
        return actions

    def list_scenarios(self):
        """Возвращает имена сценариев (без расширений, учитывая и
.test и .json)."""
        names = set()
        for f in os.listdir(self.scenarios_dir):
            name, ext = os.path.splitext(f)
            if ext in ('.test', '.json'):
                names.add(name)
        return sorted(list(names))

    def validate_scenario(self, actions):
        """Проверяет структуру сценария по JSON Schema."""
        schema_path = os.path.join(os.path.dirname(__file__),
'scenario_schema.json')
        with open(schema_path, 'r', encoding='utf-8') as f:
            schema = json.load(f)
        jsonschema.validate(actions, schema)

    def save_screenshot(self, image, scenario_name, step_index,
suffix='actual'):
        dir_path = os.path.join(self.screenshots_dir,
scenario_name)
        os.makedirs(dir_path, exist_ok=True)
        filename = f"step_{step_index}_{suffix}.png"
        path = os.path.join(dir_path, filename)
        image.save(path)
        return path

    def load_screenshot(self, path):
        """Загружает изображение."""
        return Image.open(path)

    def save_report(self, pdf_bytes, scenario_name):
        """Сохраняет PDF-отчёт."""

```

```

        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        filename = f"{scenario_name}_{timestamp}.pdf"
        path = os.path.join(self.reports_dir, filename)
        with open(path, 'wb') as f:
            f.write(pdf_bytes)
        return path

    def clear_temp_screenshots(self, scenario_name):
        """Удаляет актуальные и diff скриншоты для указанного
сценария."""
        dir_path = os.path.join(self.screenshots_dir,
scenario_name)
        if not os.path.isdir(dir_path):
            return
        # Удаляем файлы с суффиксами _actual.png и _diff.png
        patterns = ['*_actual.png', '*_diff.png']
        for pattern in patterns:
            for file_path in glob.glob(os.path.join(dir_path,
pattern)):
                try:
                    os.remove(file_path)
                    print(f"Удалён временный скриншот:
{file_path}")
                except OSError as e:
                    print(f"Ошибка при удалении {file_path}:
{e}")

    def _get_scenario_files(self, name):
        """Ищет файл сценария с любым поддерживаемым расширением
(приоритет .test)."""
        for ext in ('.test', '.json'):
            path = os.path.join(self.scenarios_dir, name + ext)
            if os.path.exists(path):
                return path
        return None

```

Файл player.py

```

import re
import time
import os
import pyautogui
from PIL import Image, ImageChops, ImageDraw
from pynput.keyboard import Controller as KbCtrl, KeyCode

try:
    import cv2
    import numpy as np
    OPENCV_AVAILABLE = True
except ImportError:
    OPENCV_AVAILABLE = False

```

Таблица символов -> виртуальный код клавиши (VK) для стандартной клавиатуры

CHAR_TO_VK = {

Кириллица (русская раскладка)

'ё': 0xC0, 'Ё': 0xC0,
'й': 0x51, 'Й': 0x51,
'ц': 0x57, 'Ц': 0x57,
'у': 0x45, 'У': 0x45,
'к': 0x52, 'К': 0x52,
'е': 0x54, 'Е': 0x54,
'н': 0x59, 'Н': 0x59,
'г': 0x55, 'Г': 0x55,
'ш': 0x49, 'Ш': 0x49,
'щ': 0x4F, 'Щ': 0x4F,
'з': 0x50, 'З': 0x50,
'х': 0x5B, 'Х': 0x5B,
'ъ': 0x5D, 'Ъ': 0x5D,
'ф': 0x41, 'Ф': 0x41,
'ы': 0x53, 'Ы': 0x53,
'в': 0x44, 'В': 0x44,
'а': 0x46, 'А': 0x46,
'п': 0x47, 'П': 0x47,
'р': 0x48, 'Р': 0x48,
'о': 0x4A, 'О': 0x4A,
'л': 0x4B, 'Л': 0x4B,
'д': 0x4C, 'Д': 0x4C,
'ж': 0x3B, 'Ж': 0x3B,
'э': 0x27, 'Э': 0x27,
'я': 0x5A, 'Я': 0x5A,
'ч': 0x58, 'Ч': 0x58,
'с': 0x43, 'С': 0x43,
'м': 0x56, 'М': 0x56,
'и': 0x42, 'И': 0x42,
'т': 0x4E, 'Т': 0x4E,
'ь': 0x4D, 'Ь': 0x4D,
'б': 0x2C, 'Б': 0x2C,
'ю': 0x2E, 'Ю': 0x2E,

Латиница (английская раскладка) - заглавные и строчные

'a': 0x41, 'A': 0x41,
'b': 0x42, 'B': 0x42,
'c': 0x43, 'C': 0x43,
'd': 0x44, 'D': 0x44,
'e': 0x45, 'E': 0x45,
'f': 0x46, 'F': 0x46,
'g': 0x47, 'G': 0x47,
'h': 0x48, 'H': 0x48,
'i': 0x49, 'I': 0x49,
'j': 0x4A, 'J': 0x4A,
'k': 0x4B, 'K': 0x4B,
'l': 0x4C, 'L': 0x4C,
'm': 0x4D, 'M': 0x4D,

```

'n': 0x4E, 'N': 0x4E,
'o': 0x4F, 'O': 0x4F,
'p': 0x50, 'P': 0x50,
'q': 0x51, 'Q': 0x51,
'r': 0x52, 'R': 0x52,
's': 0x53, 'S': 0x53,
't': 0x54, 'T': 0x54,
'u': 0x55, 'U': 0x55,
'v': 0x56, 'V': 0x56,
'w': 0x57, 'W': 0x57,
'x': 0x58, 'X': 0x58,
'y': 0x59, 'Y': 0x59,
'z': 0x5A, 'Z': 0x5A,

```

```
# Цифры
```

```

'0': 0x30, '1': 0x31, '2': 0x32, '3': 0x33, '4': 0x34,
'5': 0x35, '6': 0x36, '7': 0x37, '8': 0x38, '9': 0x39,

```

```
# Основные знаки (общие для раскладок)
```

```

' ': 0x20,          # пробел
',': 0xBC,          # запятая
'.': 0xBE,          # точка
'-': 0xBD,          # дефис/минус
'=': 0xBB,          # равно
';': 0xBA,          # точка с запятой
"'": 0xDE,          # апостроф
'/': 0xBF,          # слеш
'\\': 0xDC,         # обратный слеш
 '[': 0xDB,          # открывающая скобка
 ']': 0xDD,          # закрывающая скобка
 '`': 0xC0,          # гравис (тильда)

```

```
}
```

```
class Player:
```

```

    def __init__(self, model):
        self.model = model

```

```
    def play(self, scenario_name, actions):
```

```
        pyautogui.FAILSAFE = False
```

```
        print(f"Начало воспроизведения: {scenario_name}")
```

```
        for i, action in enumerate(actions):
```

```
            print(f"Шаг {i+1}: {action['type']}")
```

```
        {action.get('parameters', '')})")
```

```
            if action["type"] == "click":
```

```
                p = action["parameters"]
```

```
                button = p.get("button", "left")
```

```
                self._save_click_mark(scenario_name, i, p["x"],
```

```
p["y"])
```

```
                pyautogui.click(p["x"], p["y"], button=button)
```

```
            elif action["type"] == "double_click":
```

```
                p = action["parameters"]
```

```
                button = p.get("button", "left")
```

```

        self._save_click_mark(scenario_name, i, p["x"],
p["y"])
        pyautogui.doubleClick(p["x"], p["y"],
button=button)
    elif action["type"] == "drag":
        p = action["parameters"]
        start_x, start_y = p["start_x"], p["start_y"]
        end_x, end_y = p["end_x"], p["end_y"]
        button = p.get("button", "left")
        pyautogui.moveTo(start_x, start_y)
        time.sleep(0.1) # небольшая пауза для
стабилизации
        self._save_click_mark(scenario_name, i, start_x,
start_y)
        pyautogui.mouseDown(button=button)
        pyautogui.moveTo(end_x, end_y, duration=0.5)
        pyautogui.mouseUp(button=button)
        self._save_drag_mark(scenario_name, i, start_x,
start_y, end_x, end_y)
    elif action["type"] == "key_code":
        from pynput.keyboard import Controller as KbCtrl,
KeyCode
        kbd = KbCtrl()
        p = action["parameters"]
        vk = p.get("vk")
        scan = p.get("scan")
        if vk is not None:
            key_obj = KeyCode.from_vk(vk)
        elif scan is not None:
            key_obj = KeyCode.from_scan(scan)
        else:
            key_obj = KeyCode.from_vk(0)
        kbd.press(key_obj)
        time.sleep(0.02)
        kbd.release(key_obj)
    elif action["type"] == "key":
        pyautogui.press(action["parameters"]["key"])
    elif action["type"] == "text":
        self._type_text_via_keycodes(action["parameters"]
["char"])
    elif action["type"] == "screenshot":
        self._handle_screenshot(scenario_name, i, action)
    elif action["type"] == "cli":
        import subprocess
        cmd = action["parameters"]["command"]
        try:
            # Запускаем процесс без ожидания завершения
            subprocess.Popen(cmd, shell=True)
            # Небольшая пауза, чтобы система успела
создать процесс
            time.sleep(3)
        except Exception as e:

```

```

        print(f"Ошибка выполнения CLI: {e}")
    elif action["type"] == "wait":
        sec = float(action["parameters"]["seconds"])
        print(f"Ожидание {sec} с...")
        time.sleep(sec)
    else:
        print(f"Неизвестное действие: {action['type']}")
        time.sleep(0.3)
    print("Воспроизведение завершено")
    return True

def _handle_screenshot(self, scenario_name, step_index,
action):
    p = action.get("parameters", {})
    region = None
    if all(k in p for k in ("x", "y", "width", "height")):
        region = (int(p["x"]), int(p["y"]), int(p["width"]),
int(p["height"]))
        print(f"Скриншот области: {region}")
        time.sleep(0.5)
        actual = pyautogui.screenshot(region=region)
        self.model.save_screenshot(actual, scenario_name,
step_index, suffix='actual')

        ref_dir = os.path.join(self.model.screenshots_dir,
scenario_name)
        ref_path = os.path.join(ref_dir,
f"step_{step_index}_ref.png")
        if os.path.exists(ref_path):
            ref = Image.open(ref_path)
            diff = ImageChops.difference(ref, actual)
            if diff.getbbox():
                print(f"Шаг {step_index}: скриншоты различаются")
                # Генерируем изображение с выделенными отличиями
                highlighted = self._highlight_differences(ref,
actual)
                diff_path = os.path.join(ref_dir,
f"step_{step_index}_diff.png")
                highlighted.save(diff_path)
                print(f"Выделение сохранено: {diff_path}")
            else:
                print(f"Шаг {step_index}: скриншоты совпадают")
        else:
            print(f"Шаг {step_index}: эталонный скриншот не
найден")

def _highlight_differences(self, ref_img, actual_img):
    """Обводит отличающиеся области на фактическом
скриншоте"""
    if OPENCV_AVAILABLE:
        return self._highlight_opencv(ref_img, actual_img)
    else:

```

```

        return self._highlight_pillow(ref_img, actual_img)

    def _highlight_opencv(self, ref_img, actual_img):
        # Преобразуем PIL в OpenCV BGR
        ref_np = cv2.cvtColor(np.array(ref_img),
cv2.COLOR_RGB2BGR)
        actual_np = cv2.cvtColor(np.array(actual_img),
cv2.COLOR_RGB2BGR)
        diff = cv2.absdiff(ref_np, actual_np)
        gray = cv2.cvtColor(diff, cv2.COLOR_BGR2GRAY)
        _, thresh = cv2.threshold(gray, 30, 255,
cv2.THRESH_BINARY)
        contours, _ = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
        result = actual_np.copy()
        for cnt in contours:
            x, y, w, h = cv2.boundingRect(cnt)
            if w > 5 and h > 5: # отсеиваем мелкий шум
                cv2.rectangle(result, (x, y), (x+w, y+h), (0, 0,
255), 2)

        result_rgb = cv2.cvtColor(result, cv2.COLOR_BGR2RGB)
        return Image.fromarray(result_rgb)

    def _highlight_pillow(self, ref_img, actual_img):
        """Простой режим: полупрозрачное наложение разницы"""
        diff = ImageChops.difference(ref_img, actual_img)
        # Накладываем разницу с прозрачностью 50%
        actual_rgba = actual_img.copy().convert("RGBA")
        diff_rgba = diff.convert("RGBA")
        # Создаём красный оттенок для различий
        diff_red = Image.new("RGBA", diff.size, (255, 0, 0, 128))
        mask = diff.convert("L").point(lambda p: 255 if p > 30
else 0) # порог
        actual_rgba.paste(diff_red, mask=mask)
        return actual_rgba.convert("RGB")

    def _save_click_mark(self, scenario_name, step_index, x, y):
        """Делает скриншот и рисует зелёную рамку вокруг точки
(x, y)."""
        try:
            screen = pyautogui.screenshot()
            draw = ImageDraw.Draw(screen)
            half = 20 # размер рамки 40x40
            left = x - half
            top = y - half
            right = x + half
            bottom = y + half
            draw.rectangle([left, top, right, bottom],
outline='green', width=3)
            self.model.save_screenshot(screen, scenario_name,
step_index, suffix='clickmark')
        except Exception as e:

```

```

        print(f"Ошибка сохранения маркера клика: {e}")

    def _save_drag_mark(self, scenario_name, step_index, x1, y1,
x2, y2):
        """Делает скриншот и рисует стрелку от (x1,y1) к
(x2,y2)."""
        try:
            screen = pyautogui.screenshot()
            draw = ImageDraw.Draw(screen)
            # Рисуем линию
            draw.line([(x1, y1), (x2, y2)], fill='blue', width=3)
            # Рисуем треугольник на конце стрелки (в (x2,y2))
            import math
            angle = math.atan2(y2 - y1, x2 - x1)
            arrow_len = 15
            # Три вершины треугольника: острière и два угла
            tip = (x2, y2)
            left = (x2 - arrow_len * math.cos(angle - 0.5),
                    y2 - arrow_len * math.sin(angle - 0.5))
            right = (x2 - arrow_len * math.cos(angle + 0.5),
                    y2 - arrow_len * math.sin(angle + 0.5))
            draw.polygon([tip, left, right], fill='blue')
            self.model.save_screenshot(screen, scenario_name,
step_index, suffix='clickmark')
        except Exception as e:
            print(f"Ошибка сохранения стрелки drag: {e}")

    def _type_text_via_keycodes(self, text):
        """Посимвольный ввод с умным переключением раскладки."""
        import re
        from pynput.keyboard import Controller as KbCtrl, KeyCode
        from PyQt5.QtWidgets import QApplication

        keyboard = KbCtrl()
        current_layout = 'ru' # 'ru', 'en' или None

        for ch in text:
            # Определяем, буква ли это
            is_cyrillic = bool(re.match('[а-яёА-ЯЁ]', ch))
            is_latin = bool(re.match('[a-zA-Z]', ch))

            # Получаем VK из словаря
            vk = CHAR_TO_VK.get(ch)

            # Если символ не найден вообще, вставляем через буфер
            if vk is None:
                clipboard = QApplication.clipboard()
                clipboard.setText(ch)
                time.sleep(0.05)
                pyautogui.hotkey('ctrl', 'v')
                time.sleep(0.1)
                continue

```

```

# Определяем требуемую раскладку (только для букв)
if is_cyrillic:
    need_layout = 'ru'
elif is_latin:
    need_layout = 'en'
else:
    need_layout = None # символ не требует смены
раскладки

# Переключаем раскладку при необходимости
if need_layout and need_layout != current_layout:
    if need_layout == 'ru':
        pyautogui.hotkey('win', 'space') # или
Alt+Shift

        time.sleep(0.2)
        current_layout = 'ru'
    elif need_layout == 'en':
        pyautogui.hotkey('win', 'space')
        time.sleep(0.2)
        current_layout = 'en'

# Эмулируем нажатие клавиши
# Для букв учитываем Shift, если нужна заглавная
# Для не-букв (цифры, знаки) Shift не трогаем, кроме
случаев, когда символ верхний (например, '!')
is_upper = ch.isupper() if ch.isalpha() else False
if is_upper:
    keyboard.press(KeyCode.from_vk(0xA0)) # Left
Shift

key_obj = KeyCode.from_vk(vk)
keyboard.press(key_obj)
time.sleep(0.02)
keyboard.release(key_obj)
if is_upper:
    keyboard.release(KeyCode.from_vk(0xA0))
time.sleep(0.05)

```

Файл recorder.py

```

import time
import threading
import pyautogui
from pynput import mouse, keyboard

DOUBLE_CLICK_INTERVAL = 0.5
DRAG_THRESHOLD = 5

SPECIAL_KEYS_MAP = {
    keyboard.Key.enter: 'enter',
    keyboard.Key.tab: 'tab',
    keyboard.Key.space: 'space',
    keyboard.Key.backspace: 'backspace',

```

```

keyboard.Key.delete:      'delete',
keyboard.Key.esc:         'esc',
keyboard.Key.up:          'up',
keyboard.Key.down:        'down',
keyboard.Key.left:        'left',
keyboard.Key.right:       'right',
keyboard.Key.home:        'home',
keyboard.Key.end:         'end',
keyboard.Key.page_up:     'pageup',
keyboard.Key.page_down:   'pagedown',
keyboard.Key.f1:          'f1',
keyboard.Key.f2:          'f2',
keyboard.Key.f3:          'f3',
keyboard.Key.f4:          'f4',
keyboard.Key.f5:          'f5',
keyboard.Key.f6:          'f6',
keyboard.Key.f7:          'f7',
keyboard.Key.f8:          'f8',
keyboard.Key.f9:          'f9',
keyboard.Key.f10:         'f10',
keyboard.Key.f11:         'f11',
keyboard.Key.f12:         'f12',
keyboard.Key.print_screen: 'printscreen',
keyboard.Key.insert:      'insert',
keyboard.Key.pause:       'pause',
keyboard.Key.caps_lock:   'capslock',
keyboard.Key.num_lock:    'numlock',
keyboard.Key.scroll_lock: 'scrolllock',
keyboard.Key.ctrl_l:      'ctrl',
keyboard.Key.ctrl_r:      'ctrl',
keyboard.Key.alt_l:       'alt',
keyboard.Key.alt_r:       'alt',
keyboard.Key.shift_l:     'shift',
keyboard.Key.shift_r:     'shift',
keyboard.Key.cmd:         'win',
}

```

```

class Recorder:
    def __init__(self, screenshot_callback=None,
status_callback=None):
        self.is_recording = False
        self.actions = []
        self.start_time = None
        self.current_scenario = None
        self.mouse_listener = None
        self.keyboard_listener = None
        # Двойные клики
        self.last_click_time = 0
        self.last_click_pos = (0, 0)
        self.last_click_button = None
        self.pending_click = None
        # Перетаскивание

```

```

self.drag_start = None
# Выделение области по Alt
self.alt_held = False
self.alt_start_pos = None
# Модификаторы, которые не записываются отдельно
self.modifier_keys = {
    keyboard.Key.ctrl_l, keyboard.Key.ctrl_r,
    keyboard.Key.alt_l, keyboard.Key.alt_r,
    keyboard.Key.shift_l, keyboard.Key.shift_r,
    keyboard.Key.cmd
}
self.screenshot_callback = screenshot_callback
self.paused = False
self.ctrl_pressed = False
self.ctrl_alone = False
self.status_callback = status_callback

def start(self, scenario_name, initial_actions=None):
    if initial_actions is not None:
        self.actions = list(initial_actions) # создаём копию
    else:
        self.actions = []
    self.is_recording = True
    self.current_scenario = scenario_name
    self.start_time = time.time()
    self.last_click_time = 0
    self.pending_click = None
    self.drag_start = None
    self.alt_held = False
    self.alt_start_pos = None
    self.mouse_listener =
mouse.Listener(on_click=self.on_click, on_move=self.on_move)
    self.keyboard_listener = keyboard.Listener(
        on_press=self.on_press,
        on_release=self.on_release
    )
    self.mouse_listener.start()
    self.keyboard_listener.start()
    self.paused = False
    self.ctrl_pressed = False
    self.ctrl_alone = False

def stop(self):
    if self.is_recording:
        self.paused = False
        self.is_recording = False
        self.mouse_listener.stop()
        self.keyboard_listener.stop()
        if self.pending_click:
            self.actions.append(self.pending_click)
            self.pending_click = None
        self.alt_held = False

```

```

        self.alt_start_pos = None
        return self.actions
    return None

def on_move(self, x, y):
    # Не используется, но требуется для mouse.Listener
    pass

def on_click(self, x, y, button, pressed):
    if not self.is_recording or self.paused:
        return

    # Если зажат Alt, клики мыши не должны влиять на запись
    if self.alt_held:
        return

    if pressed:
        self.drag_start = (x, y, button)
        now = time.time()
        timestamp = now - self.start_time
        # Проверка двойного клика
        if (button == self.last_click_button and
            (x, y) == self.last_click_pos and
            (now - self.last_click_time) <
DOUBLE_CLICK_INTERVAL):
            if self.pending_click and
self.pending_click["type"] == "click":
                self.pending_click = None
                self.actions.append({
                    "type": "double_click",
                    "parameters": {"x": x, "y": y, "button":
button.name},
                    "timestamp": timestamp
                })
                self.last_click_time = 0
                self.pending_click = None
                self.drag_start = None
                return
        else: # отпускание кнопки мыши
            if self.drag_start is None:
                return
            start_x, start_y, start_button = self.drag_start
            self.drag_start = None
            if button != start_button:
                return
            dist = ((x - start_x) ** 2 + (y - start_y) ** 2) **
0.5

            now = time.time()
            timestamp = now - self.start_time
            if dist < DRAG_THRESHOLD:
                # Одиночный клик
                if self.pending_click:

```

```

        self.actions.append(self.pending_click)
        self.pending_click = {
            "type": "click",
            "parameters": {"x": start_x, "y": start_y,
"button": start_button.name},
            "timestamp": timestamp
        }
        self.last_click_time = now
        self.last_click_pos = (start_x, start_y)
        self.last_click_button = start_button
    else:
        # Перетаскивание
        if self.pending_click:
            self.pending_click = None
        self.actions.append({
            "type": "drag",
            "parameters": {
                "start_x": start_x,
                "start_y": start_y,
                "end_x": x,
                "end_y": y,
                "button": start_button.name
            },
            "timestamp": timestamp
        })
        self.last_click_time = 0
        self.last_click_pos = (0, 0)
        self.last_click_button = None

def on_press(self, key):
    if not self.is_recording:
        return

    # Ctrl всегда обрабатываем, даже на паузе
    if key in (keyboard.Key.ctrl_l, keyboard.Key.ctrl_r):
        self.ctrl_pressed = True
        self.ctrl_alone = True    # пока считаем, что
одиночный

        return

    # Если нажата любая другая клавиша при зажатом Ctrl —
сбрасываем флаг одиночности
    if self.ctrl_pressed:
        self.ctrl_alone = False

    if self.paused:
        # Исключение: Ctrl может обрабатываться даже на
паузе?

        # Лучше разрешить переключение паузы всегда.
        if key in (keyboard.Key.ctrl_l, keyboard.Key.ctrl_r):
            pass    # обработка Ctrl пойдёт ниже
        else:

```

```

        return

# --- Обработка Alt для выделения области скриншота ---
if key in (keyboard.Key.alt_l, keyboard.Key.alt_r):
    # Если Alt уже зажат, игнорируем повтор
    if not self.alt_held:
        self.alt_held = True
        self.alt_start_pos = pyautogui.position()
        return # не записываем Alt как действие

# Если была нажата любая другая клавиша во время
удержания Alt,
# значит Alt используется как модификатор — отменяем
выделение

if self.alt_held:
    self.alt_held = False
    self.alt_start_pos = None

# Пропускаем остальные модификаторы, чтобы не засорять
сценарий

if key in self.modifier_keys:
    return

# Фиксируем отложенный клик перед нажатием клавиши
if self.pending_click:
    self.actions.append(self.pending_click)
    self.pending_click = None

timestamp = time.time() - self.start_time
vk = getattr(key, 'vk', None)
scan = getattr(key, 'scan_code', None)

if vk is not None:
    self.actions.append({
        "type": "key_code",
        "parameters": {"vk": vk, "scan": scan},
        "timestamp": timestamp
    })
else:
    key_name = SPECIAL_KEYS_MAP.get(key)
    if key_name is None:
        if hasattr(key, 'name') and key.name:
            key_name = key.name
        else:
            key_name = str(key).replace('Key.', '')
    self.actions.append({
        "type": "key",
        "parameters": {"key": key_name},
        "timestamp": timestamp
    })

def on_release(self, key):

```

```

if not self.is_recording:
    return

# Обработка отпускания Alt — завершение выделения
if key in (keyboard.Key.alt_l, keyboard.Key.alt_r):
    if self.alt_held:
        self.alt_held = False
        end_pos = pyautogui.position()
        start_pos = self.alt_start_pos
        self.alt_start_pos = None

        # Фиксируем отложенный клик ПЕРЕД скриншотом
        if self.pending_click:
            self.actions.append(self.pending_click)
            self.pending_click = None

        if start_pos is None:
            return

        x1, y1 = start_pos
        x2, y2 = end_pos
        # Нормализуем координаты
        left = min(x1, x2)
        top = min(y1, y2)
        width = abs(x2 - x1)
        height = abs(y2 - y1)

        if width > 5 and height > 5:
            timestamp = time.time() - self.start_time
            self.actions.append({
                "type": "screenshot",
                "parameters": {
                    "x": left,
                    "y": top,
                    "width": width,
                    "height": height
                },
                "timestamp": timestamp,
                "screenshot_required": True
            })
            # Сохраняем эталонный скриншот
            if self.screenshot_callback:
                step_index = len(self.actions) - 1
                screenshot =
pyautogui.screenshot(region=(left, top, width, height))

self.screenshot_callback(self.current_scenario, step_index,
screenshot)

# Ctrl — переключение паузы
if key in (keyboard.Key.ctrl_l, keyboard.Key.ctrl_r):
    if self.ctrl_pressed and self.ctrl_alone:

```

```

        self.paused = not self.paused
        if self.paused:
            # Перед входом в паузу фиксируем отложенный
клик (если есть)
            if self.pending_click:
                self.actions.append(self.pending_click)
                self.pending_click = None
            self.drag_start = None # незавершённое
перетаскивание сбрасываем
            if self.status_callback:
                self.status_callback("Запись
приостановлена (нажмите Ctrl для продолжения)")
            else:
                if self.status_callback:
                    self.status_callback("Запись
возобновлена")
        self.ctrl_pressed = False
        self.ctrl_alone = False
        return

    def add_action(self, action):
        """Добавляет действие в текущую сессию записи (даже на
паузе)."""
        if not self.is_recording:
            return False
        # Фиксируем отложенный клик, если есть
        if self.pending_click:
            self.actions.append(self.pending_click)
            self.pending_click = None
        self.actions.append(action)
        return True

```

Файл reporter.py

```

import os
import io

from reportlab.pdfbase import pdfmetrics
from reportlab.pdfbase.ttfonts import TTFont
from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from PIL import Image

# ——— Настройка шрифтов

```

```

FONT_NAME = 'DejavuSans'
FONT_BOLD = 'DejavuSansBold'
FONT_OBLIQUE = 'DejavuSansOblique'

DEJAVUSANS_PATH = 'res/fonts/DejavuSans-MdPB.ttf'
DEJAVUSANSBD_PATH = 'res/fonts/DejavuSansBold-KEmZ.ttf'
DEJAVUSANSO_PATH = 'res/fonts/DejavuSansOblique-dvzg.ttf'

```

```

if os.path.exists(DEJAVUSANS_PATH):
    pdfmetrics.registerFont(TTFont(FONT_NAME, DEJAVUSANS_PATH))
if os.path.exists(DEJAVUSANSBD_PATH):
    pdfmetrics.registerFont(TTFont(FONT_BOLD, DEJAVUSANSBD_PATH))
if os.path.exists(DEJAVUSANSO_PATH):
    pdfmetrics.registerFont(TTFont(FONT_OBLIQUE,
DEJAVUSANSO_PATH))

```

```

if FONT_NAME not in pdfmetrics._fonts:
    FONT_NAME = 'Helvetica'
    FONT_BOLD = 'Helvetica-Bold'
    FONT_OBLIQUE = 'Helvetica-Oblique'
    print("ВНИМАНИЕ: шрифт DejaVuSans не найден, кириллица будет
отображаться некорректно.")

```

— Генератор отчётов

```

class ReportGenerator:
    def __init__(self, model):
        self.model = model

    def generate(self, scenario_name, actions):
        buffer = io.BytesIO()
        c = canvas.Canvas(buffer, pagesize=A4)
        page_width, page_height = A4

        c.setFont(FONT_BOLD, 16)
        c.drawString(30, page_height - 30, f"Отчёт выполнения
сценария: {scenario_name}")
        c.setFont(FONT_NAME, 12)
        y = page_height - 60

        for i, action in enumerate(actions):
            if y < 60:
                c.showPage()
                y = page_height - 30

            c.setFont(FONT_BOLD, 12)
            c.drawString(30, y, f"Шаг {i+1} ({action['type']})")
            y -= 18
            c.setFont(FONT_NAME, 10)
            params_text = str(action.get('parameters', {}))
            c.drawString(40, y, f"Параметры: {params_text}")
            y -= 18

            # Маркер для мышинных действий
            if action["type"] in ("click", "double_click",
"drag"):
                mark_path =
self._get_screenshot_path(scenario_name, i, 'clickmark')
                if os.path.exists(mark_path):

```

```

        y = self._draw_image(c, mark_path, 30, y,
page_width, page_height,
                                caption="Действие с
маркером")
        y -= 5

        # Скриншоты сравнения
        if action["type"] == "screenshot" or
action.get("screenshot_required"):
            ref_path =
self._get_screenshot_path(scenario_name, i, 'ref')
            actual_path =
self._get_screenshot_path(scenario_name, i, 'actual')
            diff_path =
self._get_screenshot_path(scenario_name, i, 'diff')

            # Определяем второе изображение (приоритет: diff,
затем actual)
            second_path = None
            second_caption = None
            if diff_path and os.path.exists(diff_path):
                second_path = diff_path
                second_caption = "Различия"
            elif actual_path and os.path.exists(actual_path):
                second_path = actual_path
                second_caption = "Факт"

            if ref_path and os.path.exists(ref_path) and
second_path:
                # Рядом: эталон + второе изображение
                y = self._draw_image_pair(
                    c, [ref_path, second_path],
                    ["Эталон", second_caption],
                    30, y, page_width, page_height
                )
            elif ref_path and os.path.exists(ref_path):
                # Только эталон
                y = self._draw_image(c, ref_path, 30, y,
page_width, page_height,
                                caption="Эталонный
скриншот")
            elif second_path:
                # Только diff/actual (на случай, если эталон
удалён)
                y = self._draw_image(c, second_path, 30, y,
page_width, page_height,
                                caption=second_caption)
            else:
                c.drawString(40, y, "Скриншоты отсутствуют")
                y -= 15
        y -= 10

```

```

        c.save()
        pdf_bytes = buffer.getvalue()
        buffer.close()
        return self.model.save_report(pdf_bytes, scenario_name)

def generate_multiple(self, scenario_names):
    buffer = io.BytesIO()
    c = canvas.Canvas(buffer, pagesize=A4)
    page_width, page_height = A4

    for scenario_name in scenario_names:
        try:
            actions = self.model.load_scenario(scenario_name)
        except Exception:
            c.setFont(FONT_BOLD, 14)
            c.drawString(30, page_height - 30, f"Сценарий '{scenario_name}' не найден")
            c.showPage()
            continue

        c.setFont(FONT_BOLD, 16)
        c.drawString(30, page_height - 30, f"Сценарий: {scenario_name}")
        c.setFont(FONT_NAME, 12)
        y = page_height - 60

        for i, action in enumerate(actions):
            if y < 60:
                c.showPage()
                y = page_height - 30

            c.setFont(FONT_BOLD, 12)
            c.drawString(30, y, f"Шаг {i+1} ({action['type']})")
            y -= 18
            c.setFont(FONT_NAME, 10)
            params_text = str(action.get('parameters', {}))
            c.drawString(40, y, f"Параметры: {params_text}")
            y -= 18

            if action["type"] in ("click", "double_click", "drag"):
                mark_path =
self._get_screenshot_path(scenario_name, i, 'clickmark')
                if os.path.exists(mark_path):
                    y = self._draw_image(c, mark_path, 30, y,
page_width, page_height,
caption="Действие с маркером")
                    y -= 5

```

```

        if action["type"] == "screenshot" or
action.get("screenshot_required"):
            ref_path =
self._get_screenshot_path(scenario_name, i, 'ref')
            actual_path =
self._get_screenshot_path(scenario_name, i, 'actual')
            diff_path =
self._get_screenshot_path(scenario_name, i, 'diff')

            second_path = None
            second_caption = None
            if (diff_path and os.path.exists(diff_path))
or (actual_path and os.path.exists(actual_path)):
                second_path = actual_path
                second_caption = "Факт"

            if ref_path and os.path.exists(ref_path) and
second_path:

                y = self._draw_image_pair(
                    c, [ref_path, second_path],
                    ["Эталон", second_caption],
                    30, y, page_width, page_height
                )
            elif ref_path and os.path.exists(ref_path):
                y = self._draw_image(c, ref_path, 30, y,
page_width, page_height,
                                caption="Эталонный
скриншот")
            elif second_path:
                y = self._draw_image(c, second_path, 30,
y, page_width, page_height,
                                caption=second_caption)
            else:
                c.drawString(40, y, "Скриншоты
отсутствуют")
                y -= 15
                y -= 10

            c.showPage()

            c.save()
            pdf_bytes = buffer.getvalue()
            buffer.close()
            return self.model.save_report(pdf_bytes, "multiple")

# — Вспомогательные методы


---


def _get_screenshot_path(self, scenario_name, step_index,
suffix):
    filename = f"step_{step_index}_{suffix}.png"

```

```

        return os.path.join(self.model.screenshots_dir,
scenario_name, filename)

    def _draw_image(self, canv, img_path, x, y, page_width,
page_height, caption=""):
        """Одиночное изображение на всю доступную ширину."""
        try:
            img = Image.open(img_path)
            img_w, img_h = img.size
            available_w = page_width - 2 * x
            ratio = available_w / img_w
            new_w = img_w * ratio
            new_h = img_h * ratio

            if y - new_h < 40:
                canv.showPage()
                y = page_height - 30

            if caption:
                canv.setFont(FONT_OBLIQUE, 10)
                canv.drawString(x, y, caption)
                y -= 12

            canv.drawImage(img_path, x, y - new_h, width=new_w,
height=new_h)
            y -= (new_h + 15)
        except Exception as e:
            canv.setFont(FONT_NAME, 10)
            canv.drawString(x, y, f"Ошибка загрузки изображения:
{e}")
            y -= 15
        return y

    def _draw_image_pair(self, canv, paths, captions, x, y,
page_width, page_height):
        """Два изображения рядом (слева и справа) с подписями."""
        GAP = 10
        available_w = page_width - 2 * x
        half_w = (available_w - GAP) / 2

        # Загружаем и рассчитываем размеры
        sizes = []
        for path in paths:
            img = Image.open(path)
            w, h = img.size
            ratio = half_w / w
            sizes.append((half_w, h * ratio))

        row_h = max(h for _, h in sizes)

        # Проверка места на странице
        if y - row_h < 40:

```

```

        canv.showPage()
        y = page_height - 30

    # Подписи
    canv.setFont(FONT_OBLIQUE, 10)
    canv.drawString(x, y, captions[0])
    canv.drawString(x + half_w + GAP, y, captions[1])
    y -= 12

    # Рисуем изображения, выравнивая по верхнему краю
    for idx, (img_path, (w, h)) in enumerate(zip(paths,
sizes)):
        img_x = x if idx == 0 else x + half_w + GAP
        canv.drawImage(img_path, img_x, y - h, width=w,
height=h)

        y -= (row_h + 15) # отступ после пары
    return y

```

Файл view.py

```

from PyQt5.QtWidgets import (
    QMainWindow, QWidget, QVBoxLayout, QHBoxLayout,
    QPushButton, QListWidget, QTextEdit, QLabel, QMessageBox,
    QSystemTrayIcon, QMenu, QAction, QApplication,
QAbstractItemView
)
from PyQt5.QtGui import QIcon
from PyQt5.QtCore import Qt

class MainView(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Автоматизация тестирования GUI")
        self.setGeometry(100, 100, 400, 600)
        self.setFixedWidth(400)

        # Иконка приложения
        #app_icon =
QApplication.style().standardIcon(QApplication.style().SP_ComputerIcon
)

        #self.setWindowIcon(app_icon)
        app_icon = QIcon("res/icon/app_icon.png")
        self.setWindowIcon(app_icon)
        self.tray_icon = QSystemTrayIcon(app_icon, self)

        main_widget = QWidget()
        self.setCentralWidget(main_widget)
        layout = QVBoxLayout(main_widget)

        # Список сценариев
        layout.addWidget(QLabel("Доступные сценарии:"))
        self.scenario_list = QListWidget()

```

```

self.scenario_list.setSelectionMode(QAbstractItemView.ExtendedSelectio
n)

    layout.addWidget(self.scenario_list)

    # Кнопки управления
    self.btn_new_scenario = QPushButton("Новый сценарий")
    self.btn_record = QPushButton("Записать")
    self.btn_play = QPushButton("Воспроизвести")
    self.btn_report = QPushButton("Создать отчёт")
    self.btn_insert_text = QPushButton("Вставить текст")
    self.btn_save_scenario = QPushButton("Сохранить
сценарий")

    self.btn_cli = QPushButton("CLI команда")
    self.btn_toggle_format = QPushButton("DSL / JSON")
    self.btn_wait = QPushButton("Ожидание")
    self.btn_delete = QPushButton("Удалить сценарий")

    # 1-я строка
    row1 = QHBoxLayout()
    row1.addWidget(self.btn_new_scenario)
    row1.addWidget(self.btn_delete)
    row1.addWidget(self.btn_save_scenario)
    layout.addLayout(row1)

    # 2-я строка
    row2 = QHBoxLayout()
    row2.addWidget(self.btn_record)
    row2.addWidget(self.btn_play)
    row2.addWidget(self.btn_report)
    layout.addLayout(row2)

    # 3-я строка
    row3 = QHBoxLayout()
    row3.addWidget(self.btn_insert_text)
    row3.addWidget(self.btn_cli)
    row3.addWidget(self.btn_wait)
    layout.addLayout(row3)

    # 4-я строка
    row4 = QHBoxLayout()
    row4.addWidget(self.btn_toggle_format)
    layout.addLayout(row4)

    # Редактор сценария (JSON)
    layout.addWidget(QLabel("Редактор сценария:"))
    self.editor = QTextEdit()
    self.editor.setPlaceholderText("Здесь можно
отредактировать JSON сценария...")
    layout.addWidget(self.editor)

    # Статус

```

```

self.status_label = QLabel("Готов")
layout.addWidget(self.status_label)

# Настройка системного трея
self.tray_icon = QSystemTrayIcon(app_icon, self)
self.tray_icon.setToolTip("Автоматизация тестирования
GUI")

# Контекстное меню трея
tray_menu = QMenu()
show_action = QAction("Показать окно", self)
show_action.triggered.connect(self.show_window)
quit_action = QAction("Выход", self)

quit_action.triggered.connect(QApplication.instance().quit)
tray_menu.addAction(show_action)
tray_menu.addSeparator()
tray_menu.addAction(quit_action)
self.tray_icon.setContextMenu(tray_menu)

# По двойному клику на иконке - показать окно
self.tray_icon.activated.connect(self.on_tray_activated)
self.tray_icon.show()

def show_window(self):
    """Показать окно и развернуть его."""
    self.showNormal()
    self.activateWindow()

def on_tray_activated(self, reason):
    """Обработка активации иконки в трее."""
    if reason == QSystemTrayIcon.DoubleClick:
        self.show_window()

def closeEvent(self, event):
    """При закрытии окна - сворачиваем в трей, а не завершаем
приложение."""
    event.ignore()
    self.hide()
    #self.tray_icon.showMessage(
    #    "Автоматизация тестирования GUI",
    #    "Приложение свёрнуто в трей. Для выхода используйте
контекстное меню иконки.",
    #    QSystemTrayIcon.Information,
    #    2000
    #)

def update_scenario_list(self, names):
    self.scenario_list.clear()
    self.scenario_list.addItem(names)

```

```

        def show_message(self, title, text,
icon=QMessageBox.Information):
            msg = QMessageBox(self)
            msg.setIcon(icon)
            msg.setWindowTitle(title)
            msg.setText(text)
            msg.exec_()

```

Файл controller.py

```

import json
import threading
import os
from time import sleep

from PyQt5.QtCore import QTimer, Qt
from PyQt5.QtWidgets import QFileDialog, QInputDialog,
QMessageBox, QSystemTrayIcon
import keyboard

from view import MainView
from model import ScenarioModel
from recorder import Recorder
from player import Player
from reporter import ReportGenerator
# Импортируем функции преобразования DSL
from dsl_parser import parse_dsl, dump_dsl

class AppController:
    def __init__(self):
        self.view = MainView()
        self.model = ScenarioModel()
        self.recorder = Recorder(
            screenshot_callback=self.save_reference_screenshot,
            status_callback=self.update_recording_status
        )
        self.player = Player(self.model)
        self.reporter = ReportGenerator(self.model)

        self.current_scenario_name = None
        self.current_format = 'dsl' # по умолчанию DSL

        # Подключение сигналов

self.view.btn_new_scenario.clicked.connect(self.new_scenario)

self.view.btn_record.clicked.connect(self.toggle_recording)
        self.view.btn_play.clicked.connect(self.play_scenario)

self.view.btn_report.clicked.connect(self.generate_report)

self.view.btn_insert_text.clicked.connect(self.insert_text_into_scenar
io)

```

```

self.view.scenario_list.itemSelectionChanged.connect(self.on_scenario_
selection_changed)

self.view.btn_save_scenario.clicked.connect(self.save_current_scenario
)

self.view.btn_cli.clicked.connect(self.insert_cli_command)

self.view.btn_toggle_format.clicked.connect(self.toggle_format)
    self.view.btn_wait.clicked.connect(self.insert_wait)

self.view.btn_delete.clicked.connect(self.delete_selected_scenarios)

    # Обновление списка
    self.refresh_scenario_list()

    self.initial_actions_count = 0

def show(self):
    self.view.show()

def refresh_scenario_list(self):
    scenarios = self.model.list_scenarios()
    current_item = self.view.scenario_list.currentItem()
    current_name = current_item.text() if current_item else
None

    self.view.scenario_list.blockSignals(True)
    self.view.update_scenario_list(scenarios)

    if current_name and current_name in scenarios:
        items =
self.view.scenario_list.findItems(current_name, Qt.MatchExactly)
        if items:
            self.view.scenario_list.setCurrentItem(items[0])
        else:
            self.view.scenario_list.setCurrentRow(0)
    else:
        self.view.scenario_list.setCurrentRow(0)

    self.view.scenario_list.blockSignals(False)

    new_item = self.view.scenario_list.currentItem()
    new_name = new_item.text() if new_item else None
    if new_name != self.current_scenario_name:
        self.current_scenario_name = new_name
        if new_name:
            # Загружаем действия и определяем формат файла
            actions = self.model.load_scenario(new_name)
            path = self.model._get_scenario_files(new_name)
            if path and path.endswith('.json'):

```

```

        self.current_format = 'json'
    else:
        self.current_format = 'dsl'
        # Отображаем в редакторе согласно формату
        if self.current_format == 'json':
            self.view.editor.setText(json.dumps(actions,
indent=2, ensure_ascii=False))
        else:
            self.view.editor.setText(dump_dsl(actions))
    else:
        self.view.editor.clear()

    def new_scenario(self):
        name, ok = QInputDialog.getText(self.view, "Новый
сценарий", "Введите имя сценария:")
        if ok and name:
            empty_actions = []
            # Сохраняем в DSL по умолчанию
            self.model.save_scenario(name, empty_actions,
fmt='dsl')

            self.current_format = 'dsl'
            self.refresh_scenario_list()

    def on_scenario_selection_changed(self):
        """Автоматически сохраняет текущий сценарий и загружает
        новый."""
        if self.current_scenario_name:
            self.save_current_scenario()
            item = self.view.scenario_list.currentItem()
            if item:
                name = item.text()
                self.current_scenario_name = name
                actions = self.model.load_scenario(name)
                path = self.model._get_scenario_files(name)
                if path and path.endswith('.json'):
                    self.current_format = 'json'
                else:
                    self.current_format = 'dsl'
                if self.current_format == 'json':
                    self.view.editor.setText(json.dumps(actions,
indent=2, ensure_ascii=False))
                else:
                    self.view.editor.setText(dump_dsl(actions))
            else:
                self.current_scenario_name = None

    def save_current_scenario(self):
        """Сохраняет содержимое редактора в текущий сценарий,
учитывая формат."""
        name = self.current_scenario_name
        if not name:
            return

```

```

try:
    text = self.view.editor.toPlainText()
    if self.current_format == 'json':
        actions = json.loads(text)
    else:
        actions = parse_dsl(text)
    self.model.validate_scenario(actions)
    self.model.save_scenario(name, actions,
fmt=self.current_format)
    self.view.status_label.setText(f"Сценарий '{name}'
сохранён ({self.current_format.upper()})")
except json.JSONDecodeError as e:
    self.view.show_message("Ошибка JSON", f"Некорректный
JSON:\n{e}", icon=QMessageBox.Warning)
except ValueError as e:  # ошибки парсинга DSL
    self.view.show_message("Ошибка DSL", f"Ошибка в
синтаксисе:\n{e}", icon=QMessageBox.Warning)
except Exception as e:
    self.view.show_message("Ошибка", f"Не удалось
сохранить сценарий:\n{e}")

def toggle_format(self):
    """Переключает отображение сценария между DSL и JSON."""
    item = self.view.scenario_list.currentItem()
    if not item:
        return
    name = item.text()
    text = self.view.editor.toPlainText()
    try:
        if self.current_format == 'dsl':
            # Преобразуем DSL → JSON
            actions = parse_dsl(text)
            new_text = json.dumps(actions, indent=2,
ensure_ascii=False)
            self.current_format = 'json'
        else:
            # JSON → DSL
            actions = json.loads(text)
            new_text = dump_dsl(actions)
            self.current_format = 'dsl'
        self.view.editor.setText(new_text)
        self.view.status_label.setText(f"Формат переключён в
{self.current_format.upper()})")
    except Exception as e:
        self.view.show_message("Ошибка преобразования",
str(e))

def toggle_recording(self):
    if self.recorder.is_recording:
        self.stop_recording()
    else:
        self.start_recording()

```

```

def start_recording(self):
    item = self.view.scenario_list.currentItem()
    if not item:
        self.view.show_message("Ошибка", "Выберите сценарий
для записи")
        return
    name = item.text()
    # Сохраняем текущий редактор, чтобы не потерять ручные
правки
    self.save_current_scenario()
    # Загружаем существующие действия
    try:
        existing_actions = self.model.load_scenario(name)
    except Exception:
        existing_actions = [] # если файл не найден или пуст
    self.view.status_label.setText("Запись... Нажмите
'Записать' ещё раз для остановки")
    self.view.btn_record.setText("Остановить запись")
    self.view.hide()
    self.recorder.start(name,
initial_actions=existing_actions)

def stop_recording(self):
    actions = self.recorder.stop()
    self.view.showNormal()
    if actions:
        # Удаляем два последних шага только если они были
записаны в текущей сессии
        # (т.е. общее количество действий больше, чем было до
начала записи)
        if len(actions) > self.initial_actions_count:
            # Удаляем не более двух последних действий
            num_to_remove = min(2, len(actions) -
self.initial_actions_count)
            actions = actions[:-num_to_remove]
            name = self.recorder.current_scenario
            # Сохраняем в том формате, который активен (из
current_format)
            self.model.save_scenario(name, actions,
fmt=self.current_format)
            self.refresh_scenario_list()
            # Обновляем редактор соответственно формату
            if self.current_format == 'json':
                self.view.editor.setText(json.dumps(actions,
indent=2, ensure_ascii=False))
            else:
                self.view.editor.setText(dump_dsl(actions))
            self.view.status_label.setText("Запись сохранена
(последние 2 шага отброшены)")
        else:

```

```

        self.view.status_label.setText("Запись не содержала
действий")
        self.view.btn_record.setText("Записать")

    def save_reference_screenshot(self, scenario_name,
step_index, image):
        self.model.save_screenshot(image, scenario_name,
step_index, suffix='ref')

    def play_scenario(self):
        """Запускает воспроизведение всех выбранных сценариев
последовательно."""
        selected_items = self.view.scenario_list.selectedItems()
        if not selected_items:
            self.view.show_message("Ошибка", "Выберите один или
несколько сценариев")
            return
        names = [item.text() for item in selected_items]

        for name in names:
            try:
                actions = self.model.load_scenario(name)
                self.model.validate_scenario(actions)
            except Exception as e:
                self.view.show_message("Ошибка валидации",
f"Сценарий '{name}'
некорректен:\n{e}",
icon=QMessageBox.Warning)

        return

        self.view.status_label.setText(f"Воспроизведение
{len(names)} сценариев...")
        self.view.hide()
        sleep(1)
        threading.Thread(target=self._run_playback_multiple,
args=(names,), daemon=True).start()

    def _run_playback(self, name, actions):
        self.model.clear_temp_screenshots(name)
        success = self.player.play(name, actions)
        if success:
            self.view.status_label.setText("Воспроизведение
завершено успешно")
            self._show_tray_notification("Автоматизация GUI",
f"Сценарий '{name}' успешно воспроизведены")
        else:
            self.view.status_label.setText("Воспроизведение
завершено с ошибками")
            self._show_tray_notification("Автоматизация GUI",
f"Ошибка при воспроизведении сценария '{name}'")

    def _run_playback_multiple(self, names):

```

```

        for name in names:
            self.view.status_label.setText(f"Воспроизведение:
{name}")

            actions = self.model.load_scenario(name)
            self.model.clear_temp_screenshots(name)
            success = self.player.play(name, actions)
            if not success:
                self.view.status_label.setText(f"Ошибка при
воспроизведении '{name}'")
                self._show_tray_notification("Автоматизация GUI",
f"Ошибка при воспроизведении сценария '{name}'")
                return
            self.view.status_label.setText("Все выбранные сценарии
успешно воспроизведены")
            self._show_tray_notification("Автоматизация GUI", "Все
выбранные сценарии успешно воспроизведены")

    def generate_report(self):
        selected_items = self.view.scenario_list.selectedItems()
        if not selected_items:
            self.view.show_message("Ошибка", "Выберите один или
несколько сценариев")
            return
        names = [item.text() for item in selected_items]
        try:
            if len(names) == 1:
                actions = self.model.load_scenario(names[0])
                pdf_path = self.reporter.generate(names[0],
actions)

            else:
                pdf_path = self.reporter.generate_multiple(names)
                self.view.show_message("Успех", f"Сводный отчёт
сохранён: {pdf_path}")
        except Exception as e:
            self.view.show_message("Ошибка", f"Не удалось создать
отчёт:\n{e}")

    def update_recording_status(self, message):
        self.view.status_label.setText(message)

    def insert_text_into_scenario(self):
        item = self.view.scenario_list.currentItem()
        if not item:
            self.view.show_message("Ошибка", "Сначала выберите
сценарий для редактирования.")
            return
        name = item.text()
        text, ok = QInputDialog.getText(
            self.view, "Вставить текст", "Введите текст для
вставки в сценарий:"
        )
        if ok and text:

```

```

        new_action = {
            "type": "text",
            "parameters": {"char": text},
            "timestamp": 0
        }
        if self.recorder.is_recording:
            success = self.recorder.add_action(new_action)
            if success:
                self.view.status_label.setText(f"Текст
добавлен в активную запись '{name}'")
            else:
                self.view.show_message("Ошибка", "Не удалось
добавить действие.")
        else:
            try:
                actions = self.model.load_scenario(name)
                actions.append(new_action)
                self.model.validate_scenario(actions)
                # Сохраняем в том же формате, который
используется для этого сценария
                self.model.save_scenario(name, actions,
fmt=self.current_format)
                # Обновляем редактор
                if self.current_format == 'json':

self.view.editor.setText(json.dumps(actions, indent=2,
ensure_ascii=False))
            else:

self.view.editor.setText(dump_dsl(actions))
                self.view.status_label.setText(f"Текст
добавлен в сценарий '{name}'")
            except Exception as e:
                self.view.show_message("Ошибка", f"Не удалось
добавить текст:\n{e}")
            elif ok and not text:
                self.view.show_message("Внимание", "Пустой текст не
будет добавлен.")

    def insert_cli_command(self):
        item = self.view.scenario_list.currentItem()
        if not item:
            self.view.show_message("Ошибка", "Сначала выберите
сценарий для редактирования.")
            return
        name = item.text()
        command, ok = QInputDialog.getText(
            self.view, "Добавить CLI команду", "Введите команду
для выполнения:")
        )
        if ok and command:
            new_action = {

```

```

        "type": "cli",
        "parameters": {"command": command},
        "timestamp": 0
    }
    if self.recorder.is_recording:
        success = self.recorder.add_action(new_action)
        if success:
            self.view.status_label.setText(f"Команда CLI
добавлена в активную запись '{name}')"
        else:
            self.view.show_message("Ошибка", "Не удалось
добавить команду.")
    else:
        try:
            actions = self.model.load_scenario(name)
            actions.append(new_action)
            self.model.validate_scenario(actions)
            self.model.save_scenario(name, actions,
fmt=self.current_format)
            if self.current_format == 'json':

self.view.editor.setText(json.dumps(actions, indent=2,
ensure_ascii=False))
        else:

self.view.editor.setText(dump_dsl(actions))
            self.view.status_label.setText(f"Команда CLI
добавлена в сценарий '{name}')"
        except Exception as e:
            self.view.show_message("Ошибка", f"Не удалось
добавить команду:\n{e}")
            elif ok and not command:
                self.view.show_message("Внимание", "Пустая команда не
будет добавлена.")

    def insert_wait(self):
        """Открывает диалог ввода времени ожидания и добавляет
шаг wait в сценарий."""
        item = self.view.scenario_list.currentItem()
        if not item:
            self.view.show_message("Ошибка", "Сначала выберите
сценарий для редактирования.")
            return
        name = item.text()
        seconds, ok = QInputDialog.getDouble(
            self.view, "Добавить ожидание",
            "Введите время ожидания (секунды):",
            value=1.0, min=0.0, max=3600.0, decimals=1
        )
        if ok and seconds > 0:
            new_action = {
                "type": "wait",

```

```

        "parameters": {"seconds": seconds},
        "timestamp": 0
    }
    if self.recorder.is_recording:
        success = self.recorder.add_action(new_action)
        if success:
            self.view.status_label.setText(f"Ожидание
{seconds}с добавлено в активную запись '{name}'")
        else:
            self.view.show_message("Ошибка", "Не удалось
добавить ожидание.")
    else:
        try:
            actions = self.model.load_scenario(name)
            actions.append(new_action)
            self.model.validate_scenario(actions)
            self.model.save_scenario(name, actions,
fmt=self.current_format)
            if self.current_format == 'json':

self.view.editor.setText(json.dumps(actions, indent=2,
ensure_ascii=False))

            else:

self.view.editor.setText(dump_dsl(actions))
            self.view.status_label.setText(f"Ожидание
{seconds}с добавлено в сценарий '{name}'")
        except Exception as e:
            self.view.show_message("Ошибка", f"Не удалось
добавить ожидание:\n{e}")
        elif ok:
            self.view.show_message("Внимание", "Время должно быть
больше 0.")

    def delete_selected_scenarios(self):
        """Удаляет все выбранные сценарии после подтверждения."""
        selected_items = self.view.scenario_list.selectedItems()
        if not selected_items:
            self.view.show_message("Ошибка", "Выберите сценарии
для удаления.")
            return

        names = [item.text() for item in selected_items]
        # Подтверждение
        reply = QMessageBox.question(
            self.view,
            "Подтверждение удаления",
            f"Вы действительно хотите удалить выбранные сценарии
({'', '.join(names))}? \n"
            "Это действие необратимо!",
            QMessageBox.Yes | QMessageBox.No,
            QMessageBox.No

```

```

    )
    if reply != QMessageBox.Yes:
        return

    # Удаляем файлы сценариев и их папки со скриншотами
    for name in names:
        # Удаляем файлы сценариев (оба расширения)
        for ext in ('.test', '.json'):
            path = os.path.join(self.model.scenarios_dir,
name + ext)

            if os.path.exists(path):
                try:
                    os.remove(path)
                except Exception as e:
                    print(f"Ошибка удаления {path}: {e}")
            # Удаляем папку со скриншотами, если есть
            screenshots_dir =
os.path.join(self.model.screenshots_dir, name)
            if os.path.isdir(screenshots_dir):
                import shutil
                try:
                    shutil.rmtree(screenshots_dir)
                except Exception as e:
                    print(f"Ошибка удаления папки скриншотов
{screenshots_dir}: {e}")

            # Обновляем интерфейс
            self.current_scenario_name = None
            self.view.editor.clear()
            self.refresh_scenario_list()
            self.view.status_label.setText(f"Удалены сценарии: {'',
'.join(names)}")

    def _show_tray_notification(self, title, message,
duration=3000):
        """Показывает уведомление через иконку в трее, если она
доступна."""
        if hasattr(self.view, 'tray_icon') and
self.view.tray_icon and self.view.tray_icon.supportsMessages():
            self.view.tray_icon.showMessage(title, message,
QSystemTrayIcon.Information, duration)

```

Файл dsl_parser.py

```

import shlex
import json

```

```

def parse_dsl(text):
    """Преобразует текст DSL в список действий (как в JSON)."""
    actions = []
    for line_no, line in enumerate(text.split('\n'), 1):
        line = line.strip()
        if not line or line.startswith('#'):

```

```

        continue
    try:
        parts = shlex.split(line)
    except ValueError as e:
        raise ValueError(f"Строка {line_no}: ошибка разбора:
{e}")

    if not parts:
        continue
    cmd = parts[0].lower()
    try:
        if cmd == 'click':
            # click x y [button]
            if len(parts) < 3:
                raise ValueError(f"Недостаточно аргументов
для click")

            x, y = int(parts[1]), int(parts[2])
            button = parts[3] if len(parts) > 3 else 'left'
            actions.append({'type': 'click', 'parameters':
{'x': x, 'y': y, 'button': button}})
        elif cmd == 'double_click':
            if len(parts) < 3:
                raise ValueError(f"Недостаточно аргументов
для double_click")

            x, y = int(parts[1]), int(parts[2])
            button = parts[3] if len(parts) > 3 else 'left'
            actions.append({'type': 'double_click',
'parameters': {'x': x, 'y': y, 'button': button}})
        elif cmd == 'drag':
            if len(parts) < 5:
                raise ValueError(f"Недостаточно аргументов
для drag")

            x1, y1, x2, y2 = int(parts[1]), int(parts[2]),
int(parts[3]), int(parts[4])
            button = parts[5] if len(parts) > 5 else 'left'
            actions.append({'type': 'drag',
'parameters': {'start_x': x1,
'start_y': y1, 'end_x': x2, 'end_y': y2,
'button':
button}})
        elif cmd == 'text':
            if len(parts) < 2:
                raise ValueError(f"Недостаточно аргументов
для text")

            text_content = parts[1]
            actions.append({'type': 'text', 'parameters':
{'char': text_content}})
        elif cmd == 'key':
            if len(parts) < 2:
                raise ValueError(f"Недостаточно аргументов
для key")

            key_name = parts[1]

```

```

        actions.append({'type': 'key', 'parameters':
{'key': key_name}})
        elif cmd == 'screenshot':
            params = {}
            if len(parts) >= 5:
                params = {'x': int(parts[1]), 'y':
int(parts[2]),
                        'width': int(parts[3]), 'height':
int(parts[4])}
            actions.append({'type': 'screenshot',
'parameters': params, 'screenshot_required': True})
        elif cmd == 'wait':
            if len(parts) < 2:
                raise ValueError(f"Недостаточно аргументов
для wait")
            sec = float(parts[1])
            actions.append({'type': 'wait', 'parameters':
{'seconds': sec}})
        elif cmd == 'cli':
            if len(parts) < 2:
                raise ValueError(f"Недостаточно аргументов
для cli")
            command = ' '.join(parts[1:])
            actions.append({'type': 'cli', 'parameters':
{'command': command}})
        else:
            raise ValueError(f"Неизвестная команда '{cmd}'")
    except Exception as e:
        raise ValueError(f"Строка {line_no}: {e}")
    return actions

def dump_dsl(actions):
    """Преобразует список действий обратно в DSL-текст."""
    lines = []
    for action in actions:
        act_type = action['type']
        params = action.get('parameters', {})
        if act_type == 'click':
            btn = params.get('button', 'left')
            lines.append(f"click {params['x']} {params['y']}
{btn}")
        elif act_type == 'double_click':
            btn = params.get('button', 'left')
            lines.append(f"double_click {params['x']}
{params['y']} {btn}")
        elif act_type == 'drag':
            btn = params.get('button', 'left')
            lines.append(f"drag {params['start_x']}
{params['start_y']} {params['end_x']} {params['end_y']} {btn}")
        elif act_type == 'text':
            lines.append(f"text {shlex.quote(params['char'])}")
        elif act_type == 'key':

```

```

        lines.append(f"key {params['key']}")
    elif act_type == 'screenshot':
        coords = []
        for k in ('x', 'y', 'width', 'height'):
            if k in params:
                coords.append(str(params[k]))
        if coords:
            lines.append(f"screenshot {' '.join(coords)}")
        else:
            lines.append("screenshot")
    elif act_type == 'wait':
        lines.append(f"wait {params['seconds']}")
    elif act_type == 'cli':
        lines.append(f"cli {params['command']}")
    else:
        # резервный вариант - выгрузить как JSON строку
        lines.append(f"# {json.dumps(action,
ensure_ascii=False)}")
    return '\n'.join(lines)

```

Приложение В

(Обязательное)

Приложение записи и воспроизведения сценариев тестирования графических
пользовательских интерфейсов

Презентация

Приложение записи и воспроизведения сценариев тестирования графических пользовательских интерфейсов

Автор работы: студент группы 22ВВП2 Бормотов А.А.

Научный руководитель: доцент кафедры ВТ Кузнецов А.В.

Пенза, 2026

Актуальность проекта

В современной разработке графический интерфейс является ключевым звеном взаимодействия пользователя с продуктом. Ошибки в интерфейсе составляют до 40 % всех дефектов, обнаруживаемых после выпуска ПО, а стоимость их исправления на этапе эксплуатации в 10–15 раз выше, чем при разработке.

Традиционное ручное тестирование трудозатратно, подвержено человеческому фактору и плохо масштабируется. Автоматизация же часто требует навыков программирования или лишена встроенных средств визуальной верификации.

Цели работы

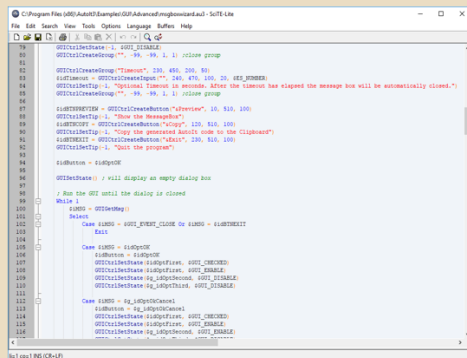
Цель работы - разработать приложение для записи и воспроизведения тестовых сценариев графических пользовательских интерфейсов с возможностью визуальной верификации.

В приложении должно быть реализовано:

- интерактивная запись сценариев с выполнением CLI команд;
- воспроизведение сценариев;
- автоматическая визуальная верификация по скриншотам;
- хранение сценариев в унифицированном формате JSON;
- генерация детализированного отчёта в формате PDF;

Обзор аналогов

Autolt



Достоинства:

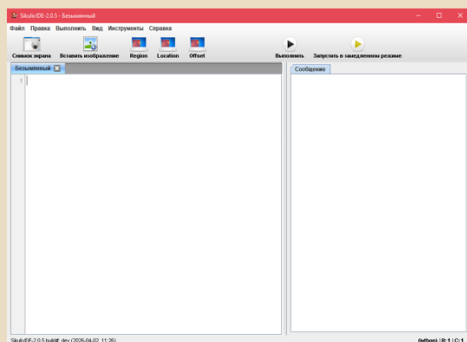
- Высокая скорость и лёгкость
- Надёжность
- Простота старта

Недостатки:

- Привязка к ОС Windows
- Сценарии пишутся вручную
- Проблемы с антивирусами

Обзор аналогов

SikuliX



Достоинства:

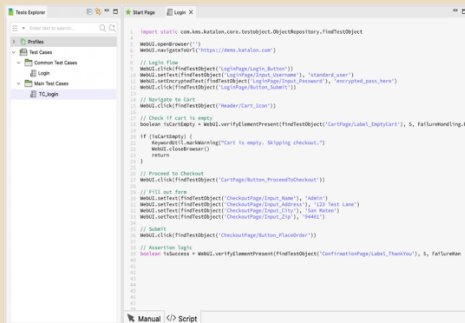
- Кроссплатформенность
- Интуитивно понятный подход «Что вижу – то кликаю»

Недостатки:

- Низкая надёжность
- Низкая производительность
- Сложность поддержки и отладки

Обзор аналогов

Katalon Studio



Достоинства:

- Record & Playback
- Мультиплатформенность
- Интеграция с CI/CD и отчёты

Недостатки:

- Закрытая экосистема
- Высокий порог для работы
- Ограниченная визуальная верификация

Стек технологий

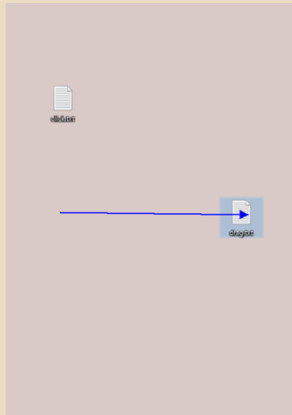


Библиотеки:

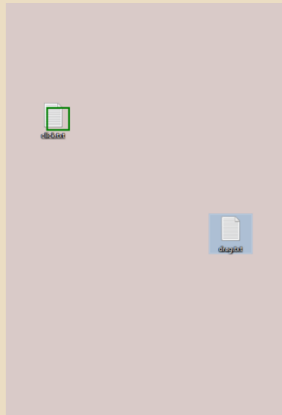
- PyQt5
- pynput и pyautogui
- OpenCV и Pillow
- ReportLab
- jsonschema

[illegible]

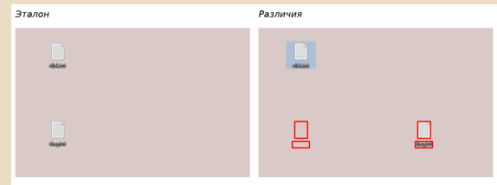
Генерация PDF-отчётов



Маркер перетаскивания

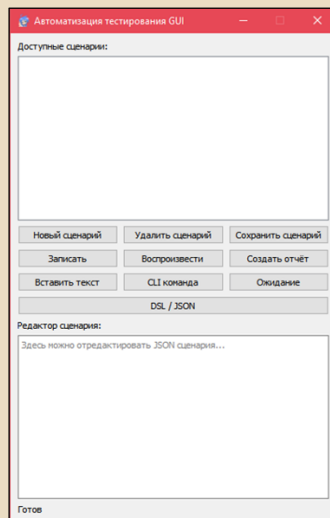


Маркер клика/двойного клика



Выделение различий

Интерфейс приложения



Тестирование

В ходе работы было проведено комплексное тестирование приложения в среде Windows 10. Тесты подтвердили корректность записи всех типов действий, надёжность алгоритмов сравнения изображений и устойчивость работы при некорректном вводе.

Использование инструмента позволяет сократить время на регрессионное тестирование GUI в 3–5 раз.

Заключение

Результатом выполнения данной выпускной квалификационной работы можно считать успешно разработанное приложение, полностью соответствующее заданию.

Разработанное приложение является готовым самостоятельным инструментом, который может применяться как профессиональными тестировщиками, так и разработчиками для отладки интерфейсов.

Спасибо за внимание!