

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: «Вычислительные машины, комплексы, системы и сети»

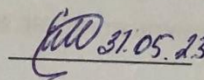
БАКАЛАВРСКАЯ РАБОТА

на тему

СИСТЕМА АВТОМАТИЗИРОВАННОГО
ТЕСТИРОВАНИЯ МОБИЛЬНОГО
ПРИЛОЖЕНИЯ ПЕДАГОГА

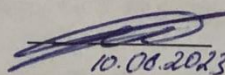
Студент

(подпись, дата)

 31.05.23 Мереняшева Е.А.

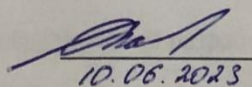
Руководитель

(подпись, дата)

 10.06.2023 Синев М.П.

Нормоконтролёр

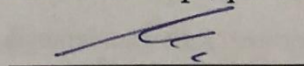
(подпись, дата)

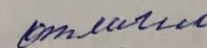
 10.06.2023 Бычков А.С.

Работа допущена к защите (протокол зас. кафедры от 02.06.2023 № 14)

Заведующий кафедрой

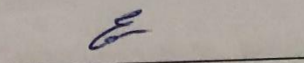
(подпись)

 Митрохин М.А.

Работа защищена с отметкой  (протокол заседания ГЭК от 16.06.23 № 1)

Секретарь ГЭК

(подпись)

 Гурин Е.И.

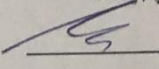
Пенза, 2023

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

Факультет
вычислительной техники

Кафедра
«Вычислительная
техника»

«Утверждаю»
Заведующий кафедрой

 М.А. Митрохин

«26» апреля 2023г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Мереняшева Елизавета Андреевна гр. 19BBB1 факультета вычислительной техники.

2. Тема работы «Система автоматизированного тестирования мобильного приложения педагога».

Тема утверждена приказом ПГУ № 458/о от "26" апреля 2023 г.

3. Руководитель бакалаврской работы к.т.н., доцент Синев М. П.

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

Назначение разработки: система автоматизированного тестирования мобильного приложения на языке программирования «Java».

Разработать систему тестов для функциональных проверок мобильного приложения на операционной системе Android.

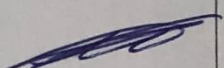
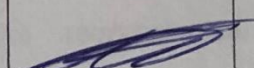
5. Перечень подлежащих разработке вопросов

1. Анализ предметной области автоматизированного функционального тестирования и фреймворков автоматизации тестирования.

2. Разработка системы функционального автоматизированного тестирования на основе проведенного анализа.

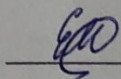
3. Анализ системы функционального автоматизированного тестирования.

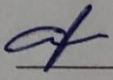
6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области функционального тестирования	13%	26.04.2023-30.04.2023	
2.	Анализ предметной области автоматизированного тестирования	13%	01.04.2023-05.04.2023	
3.	Анализ фреймворков для автоматизации тестирования	20%	06.05.2023-15.05.2023	
4.	Разработка системы функционального автоматизированного тестирования на основе проведенного анализа	29%	16.05.2023-31.05.2023	
5.	Анализ системы функционального автоматизированного тестирования	15%	01.06.2023-05.06.2023	
6.	Оформление выпускной работы	10%	06.06.2023-08.06.2023	

Дата выдачи задания "26" апреля 2023 г.

Руководитель бакалаврской работы  Синева М. П.
(подпись, дата)

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

Работу к защите допустить
Декан факультета  Фионова Л.Р.
(подпись, дата)
14.06.2023

Реферат

Бакалаврская работа содержит 80 листов, 24 рисунка, 2 таблицы, 20 источников, 3 приложения

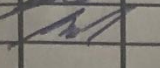
АВТОТЕСТЫ, АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, APPIUM, ТЕСТИРОВАНИЕ СИСТЕМЫ, ANDROID.

Цель работы – изучение систем автоматизированного тестирования и разработка программной системы для реализации функциональных проверок мобильного приложения педагога.

Объектом разработки является система автоматизированного тестирования, осуществляющая функциональную проверку мобильного приложения. Данная система позволяет выполнять виртуально действия пользователя и генерирует отчет о проведенных проверках.

В результате проведенной работы разработана система автоматизированного тестирования на базе фреймворка Appium на языке Java.

Результаты данной работы применены в научно-исследовательской работе ООО «Ай-Теко ВЦ» для организации рабочего процесса.

					ПГУ 1.090401.13.001 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата				
Разраб.		Мереняшева Е.А.			Система автоматизированного тестирования мобильного приложения педагога Пояснительная записка	Лит.	Лист	Листов
Провер.		Синев М.П.					2	77
Н. Контр		Бычков А.С.				ФВТ, каф. ВТ		
Утверд.		Синев М.П.				гг. 19ВВВ1		

Содержание

Перечень сокращений и обозначений.....	5
Введение	6
1 Анализ предметной области.....	6
1.1 Мобильное приложение	6
1.2 Тестирование системы.....	7
1.2.2 Достоинства внедрения автоматизированного тестирования	10
1.2.3 Использование автоматизированного тестирования	12
1.2.4 Этапы тестирования.....	13
1.3 Выбор инструментов и решения	14
1.3.1 Возможные фреймворки.....	15
1.3.2 Преимущество <i>Appium</i>	16
1.3.3 Использование <i>Appium Inspector</i>	17
1.3.4 Использование adb для написания автотестов	18
1.3.5 Тестирование мобильного приложения в <i>AndroidStudio</i>	19
1.3.6 Аналоги эмулятора <i>Android Studio</i>	22
1.3.7 Выбор языка программирования	26
1.4 Проектирование и разработка автоматизированных тестов.....	30
1.4.1 Развертывание системы проектирования	30
1.5 Разработка автоматизированных тестов	40
2 Охрана труда	54
2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ.....	54
2.1.1 Эргономика рабочего места	55
2.2 Оценка уровня шума в рабочем помещении	57
Заключение	60
Список использованных источников	62
Приложение А–UML диаграммы	64
Приложение Б – Листинг программы	69

					ПГУ 1.090401.13.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		4

Перечень сокращений и обозначений

В настоящей работе применяют следующие обозначения и сокращения

ABD (AndroidDebugBridge) - отладочный мост Android.

COM (Component Object Model) - объектная модель компонентов.

GIT (Global Information Tracker) - глобальный информационный трекер.

JDK (Java Development Kit) - комплект для разработки Java.

TestNG (Test Next Generation) – тестового поколения.

UML (Unified Modeling Language) - унифицированный язык моделирования.

XML (eXtensible Markup Language) - расширяемый язык разметки.

БЖД – безопасность жизнедеятельности.

ПДУ – предельно допустимый уровень.

ПК – персональный компьютер.

ПО - программное обеспечение.

ПЭВМ - персональная электронно-вычислительная машина.

СВФ – система выходных функций.

СКУ – система канонических уравнений.

					ПГУ 1.090401.13.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

В наше время, когда качество программного обеспечения доходит до критически высоких значений, важно, чтобы каждый компонент был тщательно проверен и оттестирован перед запуском. Тестирование является одним из ключевых этапов жизненного цикла разработки ПО и определяет готовность продукта к поставке или выпуску в продуктивную среду. Тестирование может быть проведено как вручную, так и автоматически. Автоматизированное тестирование — это процесс, при котором тестирование выполняется с помощью программного обеспечения, которое было разработано для выполнения тестирования. Оно позволяет ускорить процесс тестирования, увеличить его точность, повысить эффективность и снизить затраты на тестирование. В данной работе будет рассмотрено автоматизированное тестирование как один из методов тестирования мобильного приложения.

1 Анализ предметной области

1.1 Мобильное приложение

Мобильное приложение — это программное обеспечение, специально разработанное под конкретную мобильную платформу (*iOS, Android, Windows Phone* и т. д.).

Разработка мобильного приложения проходит в несколько этапов:

- продуктовая аналитика. На данном этапе запрашивается описание и видение проекта от заказчика, конкретизируется запрос и проводится анализ рынка. Этот этап включает в себя анализ целевой аудитории, конкурентов, изучение портрета потребителя и т.д. Этот этап нужен для создания продукта, который не только будет стабильно функционировать, но и станет востребованным для выбранной аудитории;
- спецификация и графический прототип. Данный этап проводится для определения технических требований к разработке, которые смогут после реализовать разработчики. Сюда включены требования к логике системы, работа

функционала и т.д. Это нужно для подбора наиболее выгодной стратегии разработки, которая будет оптимальной;

- оценка и планирование. На данном этапе готовится итоговый план работ, стоимость, а также срок реализации приложения. Также учитываются возможные нюансы и принцип работы мобильного приложения, определяются основные действия. Далее подписывается договор, где оговорены все детали, включая стоимость и сроки разработки;

- дизайн приложения. На этом этапе начинается непосредственная работа над приложением. Разрабатываются дизайн-концепции на выбор. Цель этапа — разработать не только красивый, но и функциональный дизайн;

- программирование. После утверждения дизайна команда приступает к разработке функционала мобильного приложения;

- тестирование. Перед запуском проекта нужно убедиться в отсутствии багов в приложении. Для этого проводится тестирование функций мобильного приложения в несколько этапов для корректной работы без ошибок. Тестирование проводится как вручную, так и с помощью автоматизированных средств;

- запуск приложения. Это итоговый этап реализации проекта. Приложение готовится к запуску и модерируется в магазине приложений. Затем оно публикуется и может быть доступно пользователям;

На этом работа над проектом обычно не заканчивается и может быть продолжена. При желании заказчика добавляются обновления и разрабатываются дополнительные функции, подключаются новые сервисы и т.д. При данном варианте многие этапы разработки повторяются.

1.2 Тестирование системы

Тестирование ПО— проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом. Конечной целью тестирования является предоставление пользователю качественного программного обеспечения. Первое впечатление

очень важно и, если продукт не сможет его произвести, пользователи найдут другой продукт, который будет отвечать всем требованиям[1].

Так же целью тестирования является нахождение ошибок, пробелов или отсутствующих требований по сравнению с фактическими требованиями.

Тестирование важно, так как оно выявляет дефекты до поставки приложения клиенту, что гарантирует качество программного обеспечения. Тестирование делает программное обеспечение более надежным, удобным и простым в использовании. Тщательно протестированное программное обеспечение обеспечивает высокопроизводительную работу системы.

Безопасность, это еще один важный момент, почему следует принимать во внимание тестирование программного обеспечения.

Благодаря тестированию программного обеспечения пользователи могут получать продукты, которые защищены от уязвимостей.

Тестирование приложения перед запуском позволяет проверить его совместимость с широким спектром операционных систем и устройств.

1.2.1 Виды тестирования

Существует большое разнообразие типов тестирования системы. Ниже описаны некоторые из них:

– функциональное тестирование (*Functional testing*). Оценивает программное обеспечение по установленным функциональным требованиям. Оно фокусируется на том, что делает приложение и что должно делать приложение, не вдаваясь в его внутреннюю работу. Это означает, что тестировщик может не иметь знаний о структуре или конструкции тестируемой системы. Функциональное тестирование сосредоточено главным образом на тестировании основных функций системы, ее доступности для пользователей и т.п.;

– модульное тестирование (*Unit testing*). Включает тестирование отдельных компонентов или функций для проверки того, что они работают в соответствии с требованиями. Обычно это делается на этапе разработки приложения и, следовательно, часто является первым типом тестирования,

выполняемого в приложении. Модульное тестирование обычно выполняется разработчиком и всегда выполняется перед интеграционным тестированием. Модульные тесты чрезвычайно полезны, поскольку они помогают выявить ошибки на ранней стадии разработки [2];

– интеграционное тестирование (*Integrationtesting*). Включает тестирование всех различных блоков приложения в единице. Основное внимание уделяется оценке соответствия системы в целом функциональным требованиям, установленным для нее. Интеграционное тестирование работает путем изучения того, как различные модули взаимодействуют друг с другом при объединении. Интеграционное тестирование, как правило, следует за модульным тестированием и помогает обеспечить бесперебойное взаимодействие между различными функциями для обеспечения бесперебойного функционирования программного обеспечения в целом;

– нефункциональное тестирование (*Non-functionaltesting*). Включает в себя тестирование всех различных нефункциональных элементов приложения, таких как производительность, надежность, удобство использования и т.д. Он отличается от функционального тестирования тем, что фокусируется не на том, что делает продукт, а на том, насколько хорошо он это делает. Как правило, нефункциональное тестирование следует за функциональным тестированием. Некоторые из наиболее распространенных типов нефункционального тестирования включают тестирование производительности, тестирование надежности, тестирование безопасности, нагрузочное тестирование, тестирование масштабируемости, тестирование совместимости и т.д.;

– регрессионное тестирование (*Regressiontesting*). Включает в себя повторное выполнение функциональных и нефункциональных тестов, чтобы проверить, выполняет ли программное обеспечение те же функции, что и раньше, после того как был сделан какой-либо вид изменения программы [3]. Основная задача регрессионного тестирования заключается в обеспечении того, чтобы ни одно из внесенных изменений не оказывало негативного влияния на существующие функциональные возможности.

Все представленные типы тестирования могут проводиться как вручную, так и с использованием автоматизированных средств;

Выбор того или иного типа тестирования зависит от нескольких факторов, например, таких как:

- тип компании. Является одним из основных факторов для определения типов тестирования системы. В данном факторе решающей особенностью является размер компании. Поскольку методы тестирования крупными компаниями, отличаются от средних или малых компаний;

- время тестирования. Тестировщик может использовать достаточно большое число типов тестирования. Однако время часто ограничивает использование только наиболее релевантных типов тестирования программного обеспечения;

- ресурсы, доступные для тестирования. Платное высокотехнологичное программное обеспечение может влиять на тип тестирования. Следовательно, ресурсы, выделяемые на тестирование, также влияют на тип тестирования;

- опыт тестировщика. Данный фактор влияет на выбор типа тестирования, так как каждые типы требуют особых навыков и умений;

- стоимость тестирования. Бюджет, который определяется для тестирования, также определяет, какой тип тестирования может быть выбран. Данный фактор связан со стоимостью программного обеспечения, которое будет использоваться для тестирования;

1.2.2 Достоинства внедрения автоматизированного тестирования

Автоматизированное тестирование — это тестирование программного обеспечения и других технических продуктов с помощью специальных программных средств автоматизации тестирования. Различные типы средств тестирования автоматизации, скриптов и фреймворков используются для выполнения тестов путем повторения заранее определенных действий и выполняются без вмешательства тестировщика.

					ПГУ 1.090401.13.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

Автоматизированные тесты используют программные средства для автоматизации ряда тестовых задач, которые ранее выполнялись ручными тестировщиками.

Вместо ручного тестирования всех элементов и функций автотесты автоматизируют процесс, чтобы сэкономить время при сохранении точности, связанной с функциональностью, производительностью и безопасностью[4].

Ниже приведены некоторые преимущества автоматизированных тестов:

- увеличение покрытия теста: одновременно можно протестировать больше сценариев, что приведет к увеличению покрытия теста. Это помогает экономить время и снимает нагрузку с ручных тестировщиков;
- повышенная точность: непрерывное тестирование увеличивает вероятность ошибок при выполнении вручную, но при автоматическом тестировании повторные тесты могут выполняться с одинаковой точностью;
- экономия затрат: при автоматизированном тестировании тесты выполняются с более высокой скоростью, а ошибки выявляются на ранних этапах цикла разработки и устраняются. Несмотря на высокие начальные затраты, после создания автоматизированной структуры происходит общее сокращение расходов;
- сокращение времени регрессионного тестирования: автоматизированная регрессия позволяет освобождать ручных тестировщиков от выполнения монотонных регрессионных тестов. Тестировщики получают время для выполнения других задач;
- помощь ручным тестировщикам: поскольку тесты могут запускаться автоматически при изменении исходного кода и уведомлять тестеров о любой проблеме, это очень полезно как тестировщикам, так и разработчикам;

Но не во всех случаях автоматизированное тестирование превосходит ручное. Существуют определенные ситуации, когда автоматизированное тестирование может оказаться не лучшим вариантом:

– приложение имеет короткий срок службы: автоматизированное тестирование требует значительных затрат времени и ресурсов, поэтому оно может быть экономически неэффективным для приложений с коротким сроком службы;

– приложение имеет низкий уровень сложности: автоматизированное тестирование лучше всего подходит для сложных приложений, которые, как ожидается, будут использоваться в течение длительного периода времени. Если приложение относительно простое, ручное тестирование может быть более эффективным;

– приложение имеет высокую степень непредсказуемости: приложения, которые имеют высокую степень непредсказуемости, может быть трудно покрыть автоматизированными тестами;

– приложение имеет высокую степень человеческого взаимодействия: приложения, которые требуют большого количества человеческого взаимодействия, например, те, которые связаны с распознаванием изображений и видео, может быть трудно покрыть автоматизированными тестами;

– приложение находится на ранних стадиях разработки: автотесты лучше всего подходят для более поздних стадий разработки, когда приложение имеет более стабильную конструкцию;

– приложение имеет низкий тестовый охват: автотесты лучше всего подходят для приложений с высоким тестовым охватом, если приложение имеет низкий тестовый охват, лучше протестировать приложение вручную.

Иногда ручное тестирование может быть более эффективным и экономичным, чем автоматизированное тестирование.

1.2.3 Использование автоматизированного тестирования

Автоматизированные тесты могут использовать разные люди в процессе разработки программного обеспечения:

- разработчики. Автоматизированные тесты позволяют разработчикам быстро идентифицировать и исправлять ошибки в коде, улучшая качество программного продукта и сокращая время, затраченное на отладку;
- тестировщики. Автоматизированные тесты помогают тестировщикам проверять повторяющиеся задачи, что позволяет сократить время тестирования и повысить качество продукта. Автотесты могут использоваться как ручными тестировщиками, так и автотестерами;
- аналитики. Автоматизированные тесты могут помочь аналитикам проводить эффективный анализ не только функциональности, но и производительности приложений, а также взаимодействие между системами;
- управляющие продуктом. Автоматизированные тесты могут помочь управляющим продуктом получать быструю обратную связь в отношении качества приложения и прогнозировать возможные риски.

Кроме того, для использования автоматизированных тестов не требуется специфических навыков, следовательно, они могут быть использованы в разных аспектах разработки продукта, не только в программировании или тестировании.

UML диаграмма прецедентов автоматизированных тестов представлена в приложении А.2.

1.2.4 Этапы тестирования

Общее количество этапов автоматизированного тестирования мобильного приложения может различаться в зависимости от инструментов тестирования, которые используются и от характеристик приложения.

К примеру, могут выделять следующие этапы:

- анализ требований. На этом этапе разработчики тестов должны ознакомиться со всеми требованиями к приложению, включая функциональные и нефункциональные. Это позволит им правильно выбрать наиболее подходящие технологии и инструменты для автоматизации тестирования;
- планирование тестирования. На этом этапе тестировщики должны разработать стратегию тестирования, определить тестовые сценарии и наборы

тестовых данных. При этом необходимо учесть разные сценарии использования приложения (как положительные, так и негативные), чтобы выявить все возможные проблемы;

- настройка среды автоматизации тестирования. На этом этапе разработчики тестов должны настроить среду, чтобы она соответствовала требованиям к тестированию мобильного приложения. Они должны также установить и настроить инструменты автоматизации;

- разработка и выполнение тестов. На этом этапе происходит разработка тест-кейсов. Тест-кейс, это документ, который описывает конкретные шаги, которые необходимо выполнить для тестирования определенной функциональности или фичи в приложении или программном обеспечении. Тест-кейс содержит данные об ожидаемых результатах, которые должны быть получены в результате выполнения определенных шагов теста, а также ожидаемый входной и выходной параметры. Так же этом этапе тестировщики разрабатывают тесты для выполнения автоматизированного тестирования. Это может быть тестирование функций, производительности, стабильности и т.д. Затем тестировщики запускают тесты и собирают результаты в отчеты, которые могут быть использованы командой разработчиков или ручных тестировщиков [5];

- анализ результатов тестирования. На последнем этапе тестировщики должны проанализировать результаты тестирования и отметить все инциденты или ошибки. Так же можно запустить тесты на реальных устройствах, чтобы убедиться, что они работают корректно. Если выявлены ошибки, они должны быть переданы команде разработки для исправления.

Блок схема этапов автоматизированного тестирования представлена в приложении А.3.

1.3 Выбор инструментов и решения

Перед тем, как решать любую задачу, требующую использования информационных технологий, важным шагом является тщательный и

всесторонний анализ. Необходимо выяснить, какие технологии будут наиболее эффективны для решения данной задачи, а также определить все необходимые аспекты, связанные с программой для проведения автоматизированного тестирования. Особое внимание уделяется сравнению и выбору инструмента для автоматизированного тестирования.

1.3.1 Возможные фреймворки

Выбор инструмента зависит от многих факторов, таких как тип тестируемого приложения, цели тестирования, бюджет и уровень навыков команды тестирования.

Ниже приведены некоторые из наиболее распространенных инструментов для автоматизированного тестирования ПО:

– *Selenium*. Это один из самых популярных инструментов для тестирования веб-приложений. Он позволяет записывать и воспроизводить тестовые сценарии и имеет множество функций, например, проверку корректности отображения веб-страниц, поддержку нескольких языков программирования и использование расширений. *Selenium* позволяет автоматизировать различные типы тестирования, такие как функциональное, регрессионное, нагрузочное и тестирование совместимости. Благодаря широкой поддержке и большому сообществу пользователей, *Selenium* стал одним из наиболее популярных инструментов для автоматизированного тестирования веб-приложений;

– *Appium*. Это инструмент для автоматизации тестирования мобильных приложений. Он позволяет записывать и воспроизводить тестовые сценарии для *iOS* и *Android*-приложений и поддерживает несколько языков программирования, например, такие как *Java*, *Python*, *Ruby*, *JavaScript* и другие. Так же он позволяет тестировать приложения на различных платформах, таких как *iOS* и *Android*, используя один и тот же *API*[6];

– *TestComplete*. Это инструмент для автоматизированного тестирования ПО, который позволяет тестировать веб-приложения, настольные приложения, мобильные приложения и другие типы ПО. Он позволяет создавать

функциональные тесты для приложений на различных платформах, включая *Windows*, *Web*, *Android* и *iOS*. Он имеет множество функций, включая поддержку множества языков программирования, таких как *JavaScript*, *Python*, *VBScript*, *C++* и других, позволяет создавать тесты как на уровне интерфейса пользователя, так и на уровне *API*., имеет возможность записи и воспроизведения тестовых сценариев и многое другое;

- *JMeter*. Это инструмент для тестирования производительности веб-приложений. Он позволяет создавать и проводить тесты нагрузки на сервере, проверять функциональность веб-приложения и другие задачи, связанные с тестированием производительности;

- *Cucumber*. Это инструмент для функционального тестирования ПО, который позволяет писать и запускать тесты на основе определенных учетных записей. Он поддерживает множество языков программирования и форматов данных, и позволяет создавать свои собственные шаблоны тестирования.

1.3.2 Преимущество *Appium*

Appium является отличным инструментом автоматизации тестирования мобильных приложений. Ниже приведены несколько причин, почему *Appium* подходит лучше всего:

- *Appium* поддерживает множество операционных систем, включая *iOS*, *Android*, *Windows* и другие, а также различные языки программирования, такие как *Java*, *Python*, *Ruby* и другие. Это позволяет использовать инструмент в зависимости от потребностей [7];

- *Appium* поддерживает ряд распространенных функций автоматизации тестирования мобильных приложений, таких как мультиязычность, работа с геопозицией и биометрической авторизации;

- *Appium* использует собственный протокол *WebDriver* для взаимодействия с мобильным приложением, что обеспечивает стабильную работу в широком диапазоне условий тестирования.

Appium является одним из наиболее популярных и функциональных инструментов автоматизации тестирования мобильных приложений, что является отличным выбором для проекта.

Appium можно установить в двух вариациях: установить с использованием командной строки, используя *Node.js* или установить десктопную версию с официального сайта. Интерфейс десктопной версии *Appium* представлен на рисунке 1.

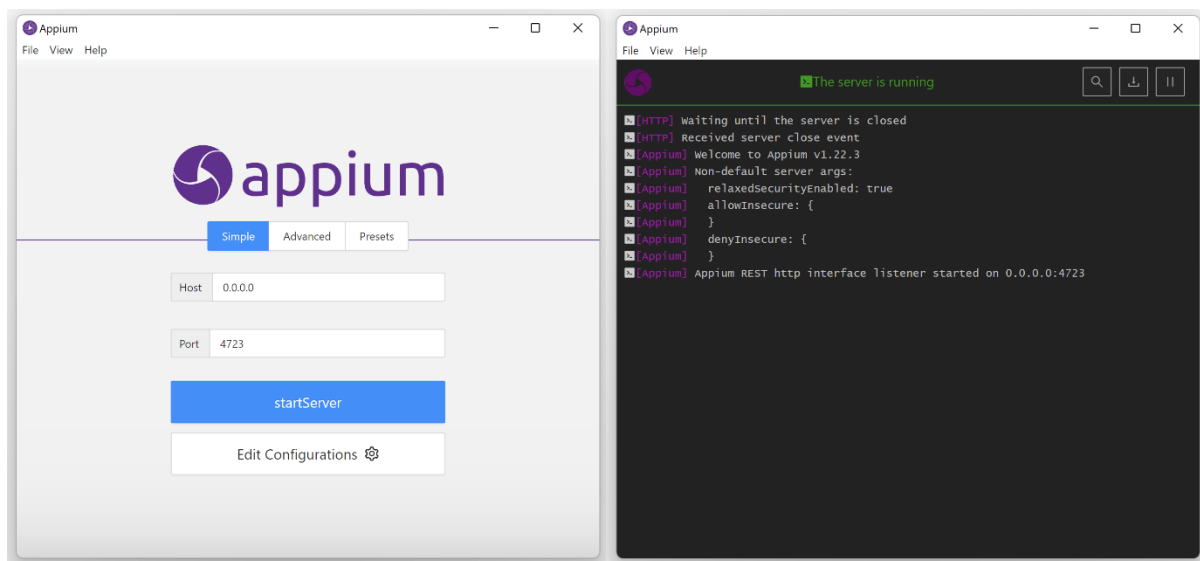


Рисунок 1 - Интерфейс *Appium*

1.3.3 Использование *AppiumInspector*

AppiumInspector— это инструмент визуального поиска элементов пользовательского интерфейса (UI) в мобильных приложениях, который предоставляется в составе фреймворка *Appium*. Интерфейс *AppiumInspector* представлен на рисунке 2:

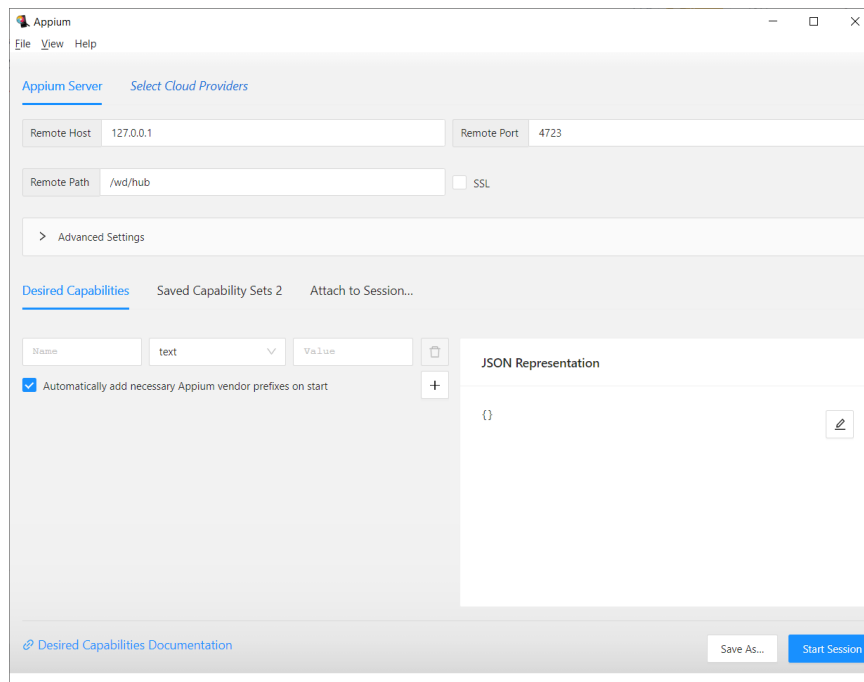


Рисунок 2 - Интерфейс *AppiumInspector*

В *AppiumInspector* можно просмотреть иерархию элементов на экране, их свойства, значения атрибутов и другие детали. Это делает процесс создания тестов более быстрым и эффективным, позволяя избежать ошибок при создании кода для тестирования составных элементов пользовательского интерфейса.

AppiumInspector может использоваться в сочетании с различными языками программирования и средами разработки, что позволяет разработчикам и тестировщикам выбирать наиболее удобный для них способ создания и отладки автоматизированных тестов в *Appium*.

1.3.4 Использование adb для написания автотестов

adb — это инструмент командной строки, который используется для управления устройствами, работающими на операционной системе *Android*. Он может использоваться для тестирования мобильных приложений, в том числе для выполнения следующих операций:

- установка и удаление приложений на устройстве;
- получение логов устройства;
- получение информации о устройстве (разрешение экрана, версии операционной системы и т.д.);

- отправка намерений и событий;
- эмуляция нажатий и жестов пальцев на экране;
- изменение настроек устройства (включение/выключение *Wi-Fi*, *Bluetooth* и т.д.) [8].

Для использования *adb* необходимо установить драйверы для соответствующего устройства и выполнить команду *adbdevices* для проверки его наличия и готовности к использованию. После этого можно выполнять необходимые команды *adb* для тестирования приложения на устройстве.

1.3.5 Тестирование мобильного приложения в *AndroidStudio*

Android Studio — это интегрированная среда разработки *Android*-приложений, которая предоставляет широкий спектр инструментов для создания, тестирования и отладки приложений на платформе *Android*[9]. Интерфейс начальной страницы *AndroidStudio* представлен на рисунке 3.

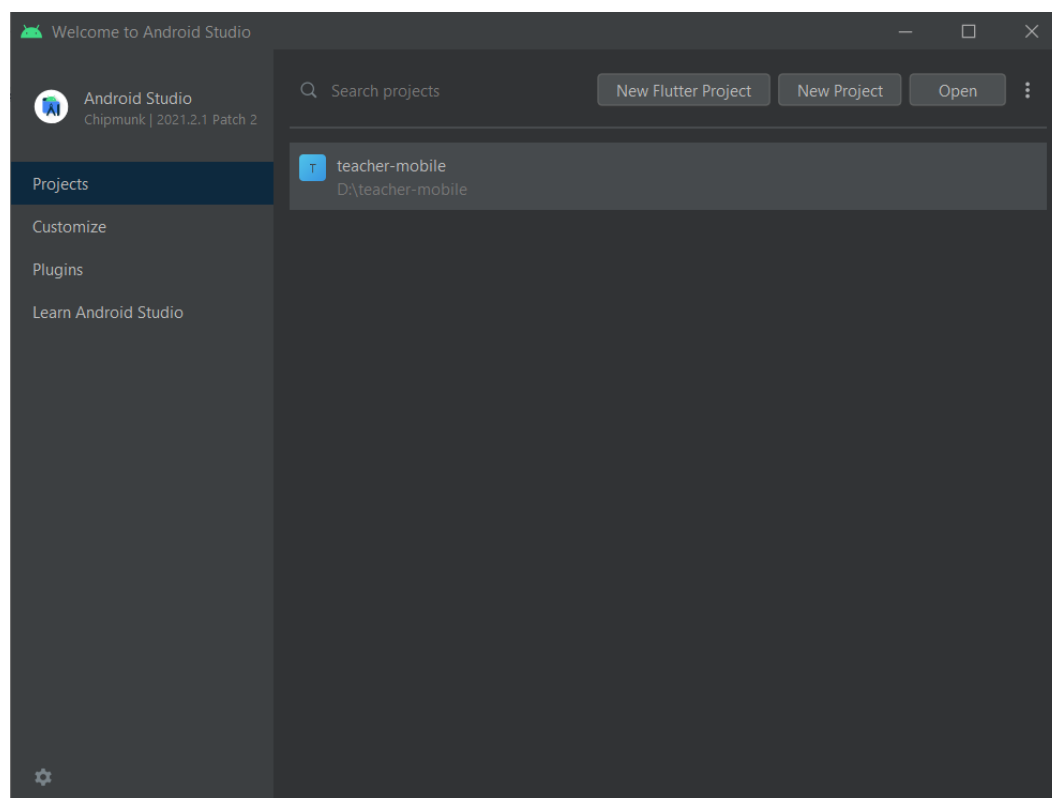


Рисунок 3 - Начальная страница *AndroidStudio*

Она создана на базе *IntelliJ IDEA* и предлагает множество инструментов для сборки, отладки и тестирования приложений для устройств на *Android*.

AndroidStudio также предоставляет возможности для быстрой разработки и тестирования приложений на реальных устройствах или в эмуляторах. С использованием данной среды можно создавать и настраивать эмуляторы, изменять разрешения экрана и установленные компоненты и т.д.

Эмулятор *AndroidStudio* является частью этой среды разработки и позволяет тестировать приложения без необходимости установки и использования физического устройства. Он имитирует работу устройства на платформе *Android*, так что тестировщик может тестировать приложения на разных версиях операционной системы и разных размерах экранов.

Создать эмулятор можно через менеджера устройств (рисунок 4).

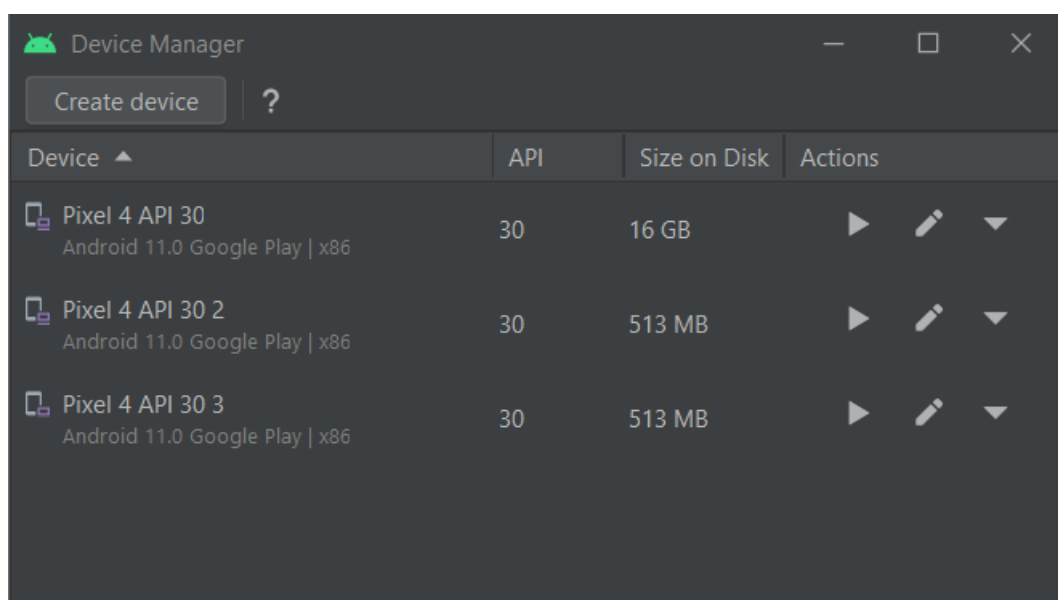


Рисунок 4 - Менеджер девайсов

Эмулятор *Android Studio* обладает возможностями настройки различных параметров для эмулируемого устройства, таких как количество памяти, доступное разрешение экрана, версии *Android* и другие параметры для тестирования приложения (рисунок 5).

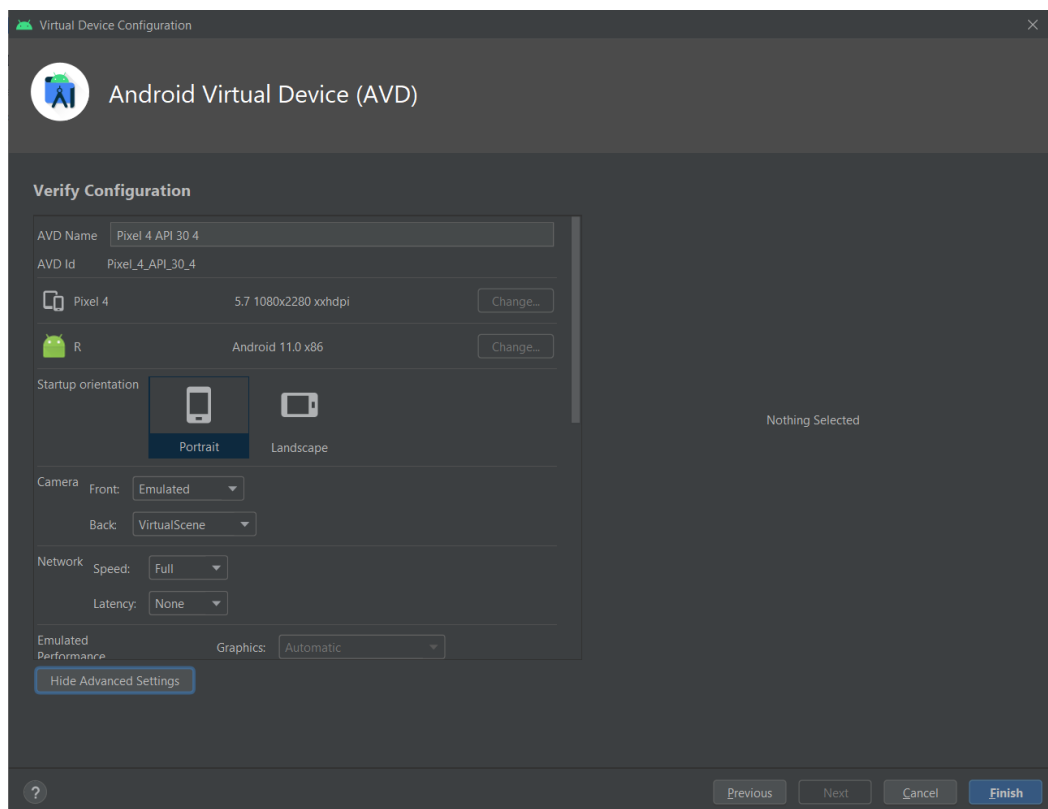


Рисунок 5 - Настройка эмулятора

Одним из недостатков работы с эмулятором является то, что скорость работы иногда может быть медленнее, чем на физическом устройстве, особенно при выполнении ресурсоемких задач. Кроме того, эмулятор не может полностью эмулировать аппаратные особенности устройства, такие как акселерометр или наличие сенсорного экрана.

Тем не менее, эмулятор *Android Studio* является удобным инструментом для разработки и тестирования приложений, благодаря которому можно быстро проверять работу приложения на различных платформах *Android* и ускорить процесс дебага и тестирования устройства.

Так же *AndroidStudio* поддерживает подключение физического устройства.

Для подключения физического устройства к *Android Studio* нужно убедиться, что на физическом устройстве включен режим разработчика. Подключить физическое устройство к среде разработки можно через *USB* кабель.

Окно выбора физических устройств представлено на рисунке 6.

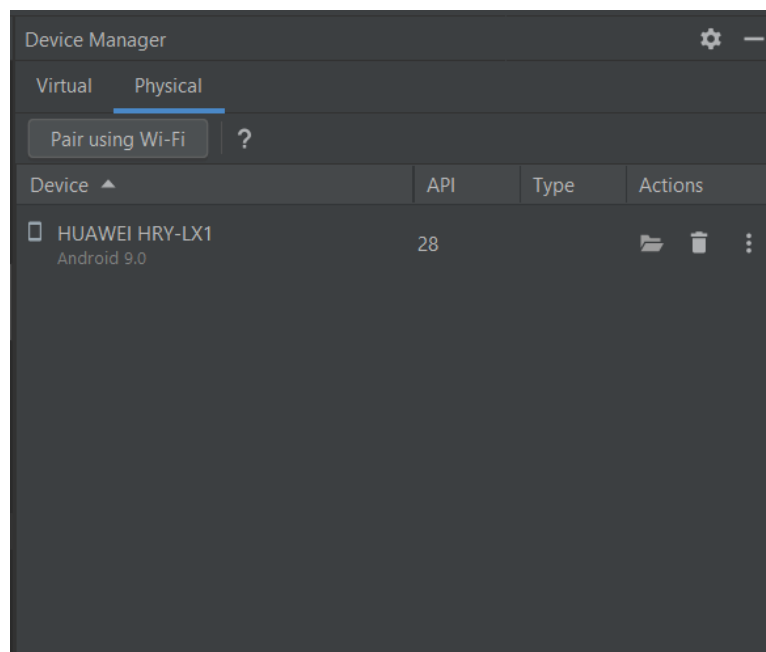


Рисунок 6 - Выбор физического устройства

1.3.6 Аналоги эмулятора *Android Studio*

Существует несколько альтернатив эмулятору *Android Studio*.

Genymotion. Это платформа виртуализации устройств *Android*, которая имеет более быстрый и стабильный эмулятор, поддерживающий широкий спектр устройств и образов *Android* [10].

Программа поддерживает множество различных устройств, начиная от устройств с низким разрешением экрана до планшетов. Это позволяет тестировать приложения на целом ряде различных устройств, чтобы гарантировать их работоспособность.

Один из основных преимуществ *Genymotion*— это быстродействие и стабильность. Он использует технологию виртуализации для создания устройств, что позволяет эмулятору работать много быстрее, чем нативный эмулятор *Android* в *Android Studio*.

Кроме того, *Genymotion* имеет множество дополнительных функций, таких как возможность тестирования *GPS*, камеры и других датчиков виртуального устройства. Также можно запускать несколько экземпляров виртуальных устройств одновременно, что ускорит процесс тестирования продукта.

Genymotion доступен в как бесплатной, так и платной версии, и может быть интегрирован со многими инструментами разработки, в том числе с *Android Studio*.

Интерфейс *Genymotion* представлен на рисунке 7.

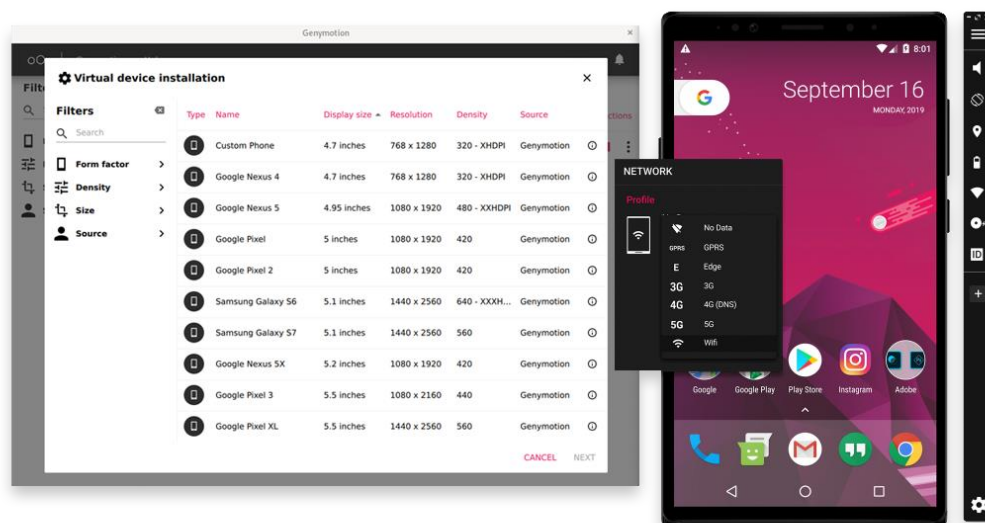


Рисунок 7 - Интерфейс *Genymotion*

BlueStacks. Это эмулятор *Android*, который преимущественно используется для запуска мобильных игр на ПК, но также может использоваться для запуска любых приложений. Данный эмулятор может быть использован для тестирования мобильных приложений на ПК [11].

Эмулятор позволяет имитировать работу *Android* устройства на ПК и обеспечивает доступ к *Google Play*.

BlueStacks имеет целый ряд функций, которые делают его отличным выбором для тестирования мобильных приложений. Он поддерживает множество различных устройств и версий *Android*, что позволяет тестировать приложения на более широком диапазоне конфигураций устройств.

BlueStacks также интегрирован с *Android Studio*, а это значит, что его легко использовать во время разработки приложений. Это может быть особенно полезно, когда необходимо быстро проверить, как выглядит или работает новая версия приложения на разных устройствах.

Интерфейс *BlueStacks* представлен на рисунке 8.

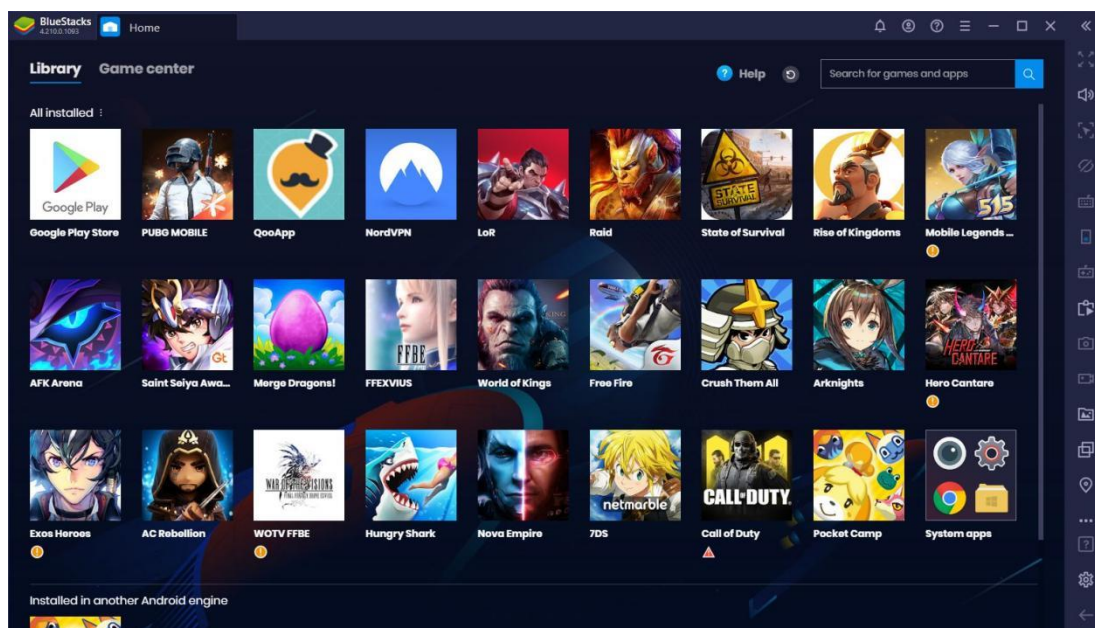


Рисунок8 - Интерфейс *BlueStacks*

Xamarin. Это платформа для разработки кроссплатформенных приложений для *Android*, *iOS* и *Windows*, которая включает в себя эмуляторы для тестирования приложений.

Xamarin имеет как достоинства, так и недостатки.

Достоинства:

- кроссплатформенность: *Xamarin* позволяет разработчикам создавать мобильные приложения для *iOS*, *Android* и *Windows Phone*;
- интеграция с *Visual Studio*: *Xamarin* интегрирован с *Visual Studio*, что позволяет разработчикам полностью использовать возможности этой *IDE*.

Недостатки:

- обучение: использование *Xamarin* требует знания языка программирования *C#* и *.NET* фреймворка, что может быть проблемой для тех, кто не знаком с этими технологиями.

Интерфейс *Xamarin* представлен на рисунке 9.

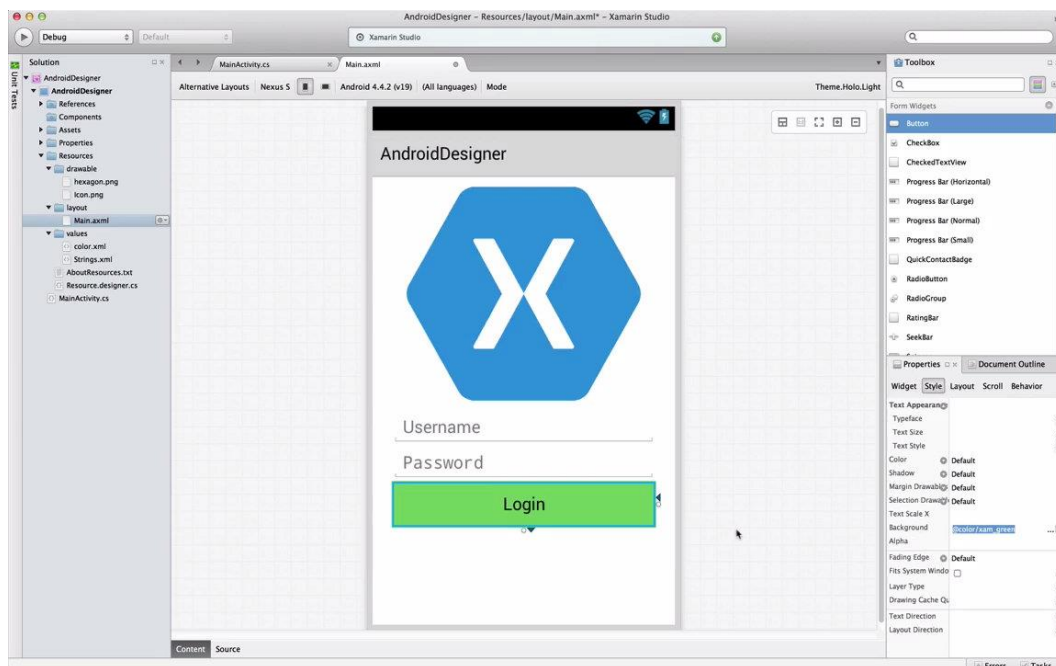


Рисунок9 - Интерфейс *Xamarin*

NoxPlayer. Это эмулятор *Android*, который может использоваться для тестирования мобильных приложений на разных устройствах и различных версиях *Android*.

NoxPlayer поддерживает разные версии *Android*, начиная от 4.4 и до последней версии, что дает возможность тестировать приложения на разных ОС.

NoxPlayer позволяет открыть несколько приложений на одном экране и работать с ними одновременно, что упрощает тестирование интерактивных функций, таких как обмен сообщениями или отправка файлов.

NoxPlayer имеет встроенные инструменты для тестирования приложений, такие как запись и воспроизведение действий, сравнение скриншотов, многократное вход в приложение и т.д.

NoxPlayer так же имеет несколько недостатков, к примеру *NoxPlayer* может имитировать работу портативных устройств, но не полностью эмулирует их поведение и производительность.

Интерфейс *NoxPlayer* представлен на рисунке 10.



Рисунок 10 - Интерфейс *NoxPlayer*

1.3.7 Выбор языка программирования

Одним из ключевых аспектов успешного автоматизированного тестирования является выбор подходящего языка программирования для написания скриптов тестирования. Оптимальный выбор языка может повысить эффективность тестирования, минимизировать затраты на разработку и поддержку, ускорить процесс автоматизации и сделать тестирование более гибким и масштабируемым. Однако, существует множество различных языков программирования для автоматизированного тестирования, каждый из которых имеет свои преимущества и недостатки, а также возможности и ограничения, которые нужно учитывать при выборе. В данной главе рассматриваются существующие языки программирования.

Три лучших языка программирования для написания автотестов - *Python*, *Java* и *C#*. Эти языки популярны среди разработчиков и компаний, занимающихся автоматизированным тестированием, благодаря простоте их использования и широкому спектру доступных для тестирования фреймворков и библиотек.

1.3.8 Язык программирования *Python*

Python - один из распространенных языков программирования для автоматизированного тестирования. Он имеет множество функций, которые делают его подходящим для автоматизированного тестирования.

Python хорошо известен тем, что имеет множество библиотек практически для всего, поэтому не нужно тратить время на написание кода с нуля.

Так же стандартная библиотека *Python* уже включает в себя множество инструментов для модульного тестирования.

Ниже приведены фреймворки, которые можно подключить и использовать в *Python* для автоматизированного тестирования:

- *PyUnit*;
- *Nose2*;
- *PyTest*;
- *Behave*;
- *Lettuce*;
- *Robot Framework*;
- *Selenium Framework* [12].

1.3.9 Язык программирования C#

C# — это подходящий язык программирования для написания автоматизированных тестов, особенно для тестирования веб-приложений, так как он обладает богатым инструментарием для работы с веб-технологиями и базами данных [13]. Наиболее популярные фреймворки для написания автоматизированных тестов на C# представлены ниже:

- *NUnit*;
- *MSTest*;
- *xUnit*.

C# обладает мощной интегрированной средой разработки (*IDE*) - *Microsoft Visual Studio*, которая поддерживает такие функциональности, как *IntelliSense*, автоматическую генерацию кода и отладку, что делает процесс написания автотестов более быстрым и эффективным. C# также имеет возможность обращаться к объектам *COM* и использовать код, разработанный на языке C++, что расширяет возможности тестирования приложений.

C# поддерживает платформу *.NET*, которая обеспечивает возможность кроссплатформенной разработки и простую интеграцию с другими технологиями *Microsoft*. Кроме того, C# достаточно прост в использовании и усвоении, особенно для разработчиков, имеющих опыт с языком программирования *Java*.

В целом, C# является очень хорошим выбором для написания автоматизированных тестов, особенно для тестирования веб-приложений. Однако, как и в случае с любым языком программирования, выбор C# для написания автоматизированных тестов должен зависеть от конкретных требований проекта и доступного инструментария.

1.3.10 Язык программирования *Java*

Java — это очень популярный и широко используемый язык программирования для написания автоматизированных тестов на *Android* и других платформах. Преимущества языка *Java* для написания автоматизированных тестов следующие:

Кроссплатформенность: тесты, написанные на языке *Java*, могут работать на различных операционных системах и мобильных устройствах.

Надежность: *Java* имеет строгую типизацию, что позволяет рассчитывать на более надежную систему автоматизированного тестирования.

Легко читаемый код: *Java* код относительно прост в написании, читаемый и поддерживаемый, что делает его более доступным для новичков.

Большой выбор инструментов: *Java* обладает огромным сообществом разработчиков по всему миру, что означает широкий выбор инструментов и библиотек для автоматизированного тестирования, к примеру [14]:

- *JBehave*;
- *Junit*;
- *Serenity*;
- *TestNG*;
- *Selenide*;
- *Gauge*;

- *Geb*;
- *Spock*.

Легкость интеграции: *Java* легко интегрируется с другими языками и инструментами, что упрощает процесс автоматизации тестирования.

В моем случае *Java* является наиболее подходящим языком программирования для написания автотестов. Во-первых, язык крайне распространен, популярен и имеет широкое сообщество разработчиков и тестировщиков. Кроме того, *Java* объектно-ориентированный язык программирования, что делает этот язык более легким в использовании и понимании. Он является также кроссплатформенным, что позволяет запускать автотесты на любой из множества платформ. Кроме того, *Java* обладает производительным инструментарием для тестирования, включая такие инструменты, как *JUnit*, *TestNG* и *SeleniumWebDriver*, которые обеспечивают отличную функциональность автоматизированного тестирования. Наконец, *Java* постоянно развивается и обновляется, что делает его подходящим языком программирования для автоматизированного тестирования в современном мире информационных технологий. Так же, выбор данного языка программирования для автоматизированного тестирования зависит от конкретных потребностей проекта, навыков команды и доступного инструментария[15].

1.4 Проектирование и разработка автоматизированных тестов

Разработка автоматизированных тестов — это процесс создания тестовых сценариев, которые выполняются программными средствами без участия человека. Это позволяет разработчикам и тестировщикам автоматизировать процесс проверки работы приложения на соответствие его требованиям [16].

1.4.1 Развертывание системы проектирования

Развертывание системы проектирования — это процесс установки и подготовки системы для использования в работе над проектами. Этот процесс включает в себя установку необходимых программных компонентов, конфигурацию среды разработки и настройку системы на конкретные требования.

Первый шаг в развертывании системы проектирования - установка среды разработки.

Затем необходимо настроить среду разработки на определенные требования, связанные с проектами, над которыми будет работать тестировщик. Может понадобиться интеграция с другим программным обеспечением или системами, которые используются в рамках проекта.

Следующим этапом развертывания является установка плагинов и расширений для среды разработки. Эти инструменты позволяют тестировщикам увеличить свою производительность и расширить возможности среды разработки.

В целом, развертывание системы проектирования — это неотъемлемый шаг в процессе проектирования систем. Необходимость корректной настройки и наличия подходящих инструментов для увеличения производительности и улучшения качества проектов не может быть переоценена.

Ниже представлено подробное описание развертывания системы.

1.4.2 Установка *JavaJDK*

Java является одним из самых распространённых языков программирования в мире и используется в большинстве систем.

Java JDK— это пакет разработки *Java*, который включает в себя необходимые инструменты и библиотеки для разработки и запуска *Java*-приложений.

Установка *Java JDK* на компьютер является первоначальной задачей для тех, кто работает с *Java*-приложениями. Ниже представлено, как установить *JavaJDK* на компьютер под управлением *Windows*.

Шаг 1.Перейти на официальный сайт *jdk.java* и скачать нужную версию *JDK* для *Windows* (рисунок 11).

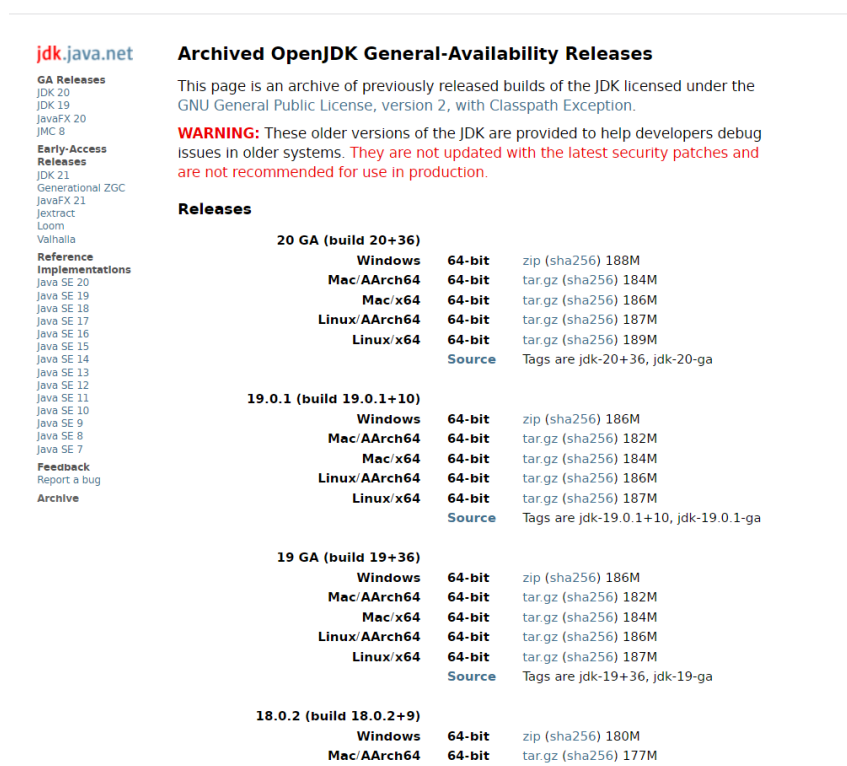


Рисунок 11 - Официальный сайт *JavaJDK*

Шаг 2.Разархивовать архив и перенести содержимое архива в желаемую папку.

Шаг 3.Добавить в переменные среды *JAVA_HOME*. Переменные среды в ПК — это настройки, которые определяют поведение программ и операционной системы, а также позволяют им общаться между собой. Например, переменная *PATH* указывает на путь к исполняемым файлам, которые можно запустить из любой директории в командной строке.

Для добавления перейти в настройки ПК и нажать на «Дополнительные параметры системы». Откроется окошко «Свойства системы». В окошке нажать на кнопку «Переменные среды» (рисунок 12).

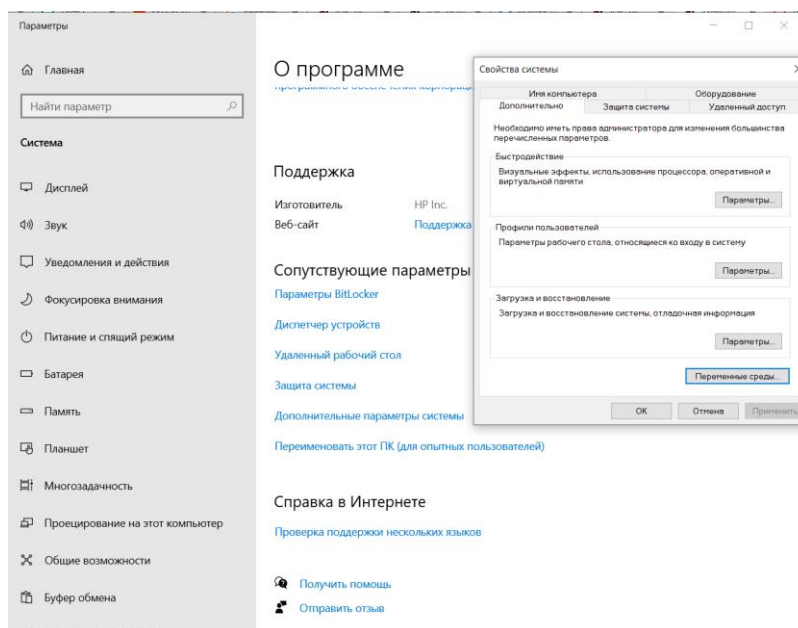


Рисунок 12 - Настройки компьютера

Нажать на кнопку «Создать» (рисунок 13).

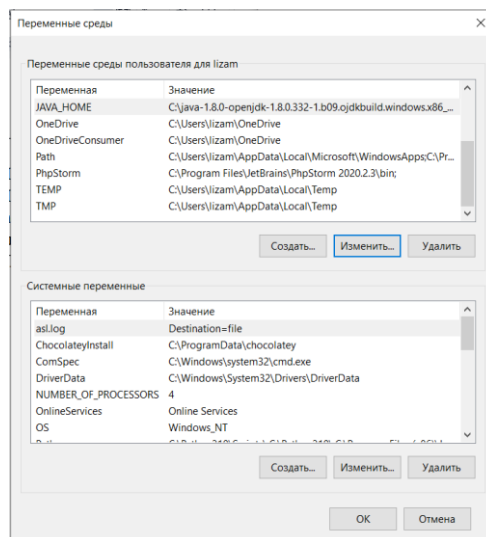


Рисунок 13 - Переменные среды

В появившемся окне «Новая пользовательская переменная» ввести имя *JAVA_HOME* и в поле значение путь до папки с установленным *JDK*. В моем

случае это `C:\java-1.8.0-openjdk-1.8.0.332-1.b09.ojdkbuild.windows.x86_64\java-1.8.0-openjdk-1.8.0.332-1.b09.ojdkbuild.windows.x86_64` (рисунок 14).

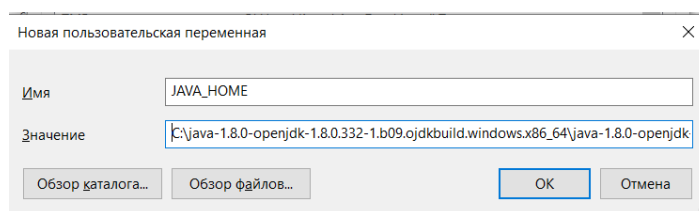


Рисунок 14 - Переменные среды

Шаг 4.Перезапустить ПК.

Шаг 5.Проверить успешность установки, открыв командную строку и введя команду `"java -version"`. Если *Java* была успешно установлена, появится сообщение с информацией о версии *Java*, установленной на компьютере (рисунок 15).

```
C:\Users\lizam>java -version
java version "1.8.0_291"
Java(TM) SE Runtime Environment (build 1.8.0_291-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.291-b10, mixed mode)
```

Рисунок 15 - Проверка установки *Java*

1.4.3 Скачивание *Android Studio* и создание эмулятора

Для того, чтобы скачать и установить *Android Studio*, нужно выполнить действия, описанные ниже.

Первым шагом нужно перейти на официальный сайт *Android Studio* и скачать установочный файл (рисунок 16).

<div> <div>developers</div> <div>Platform</div> <div>Design & Plan</div> <div>Multidevice</div> <div>Google Play</div> <div>Does</div> <div>Ещё</div> <div>Search</div> <div>Language</div> <div>Android Studio</div> <div>11</div> </div> <div> <div>Download</div> <div>Android Studio editor</div> <div>Android Gradle Plugin</div> <div>SDK tools</div> <div>Preview</div> </div>				
<h2>Android Studio downloads</h2> <p>Download the latest version of Android Studio. For more information, see the Android Studio release notes.</p>				
Platform	Android Studio package	Size	SHA-256 checksum	
Windows (64-bit)	android-studio-2022.2.119-windows.exe Recommended	1.0 GB	6b31be219dbcca315866ce94bbc7b5c244d617ba1be97d66b1b3c49c02506d56	
Windows (64-bit)	android-studio-2022.2.119-windows.zip No .exe installer	1.0 GB	8ec5b71a0f9df07542348a8021c39ea19e4bdfad56b104ea688a0d53bb275d7	
Mac (64-bit)	android-studio-2022.2.119-mac.dmg	1.2 GB	c726466e9db294580488f8b986cf86a35f03c3a0db15999d59e13fc1c2c34	

Рисунок 16 - Официальный сайт *AndroidStudio*

Далее нужно запустить скачанный установочный файл *Android Studio* и следовать инструкциям по установке.

Далее нужно запустить *Android Studio* (рисунок 17).

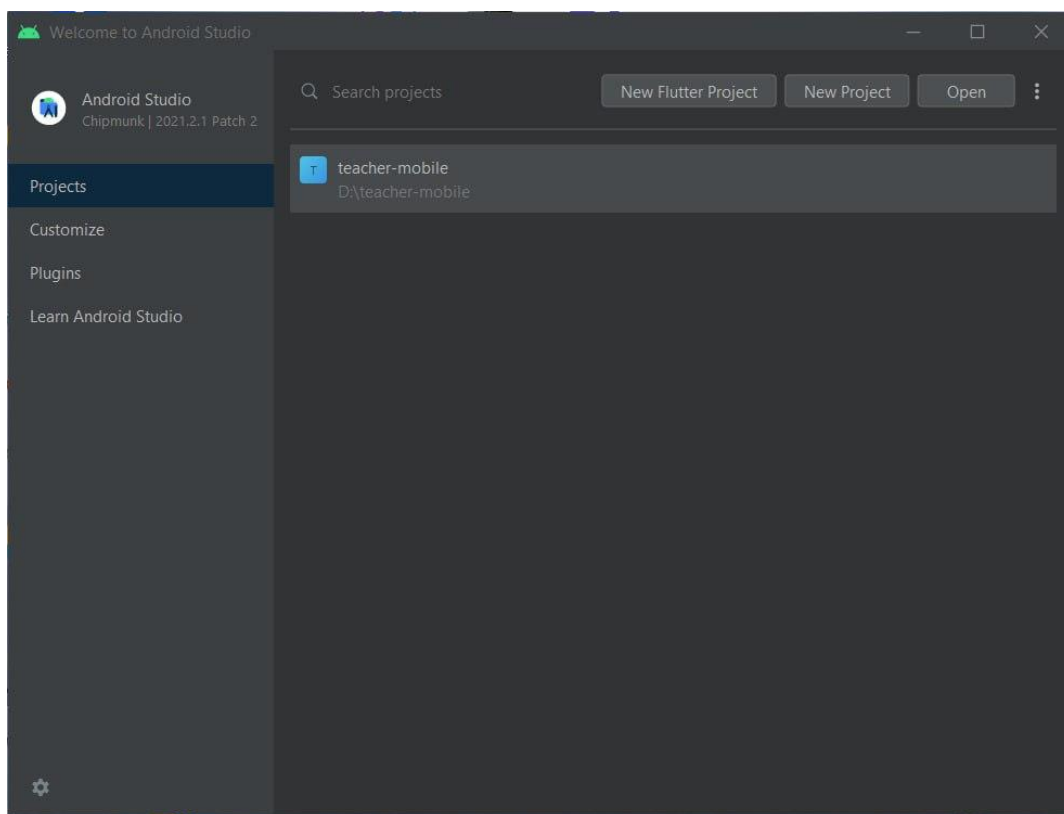


Рисунок 17 - Начальный экран *AndroidStudio*

После нужно создать эмулятор. Для создания эмулятора в разделе *DeviceManager* нужно нажать на кнопку «*CreateDevice*» (рисунок 18).

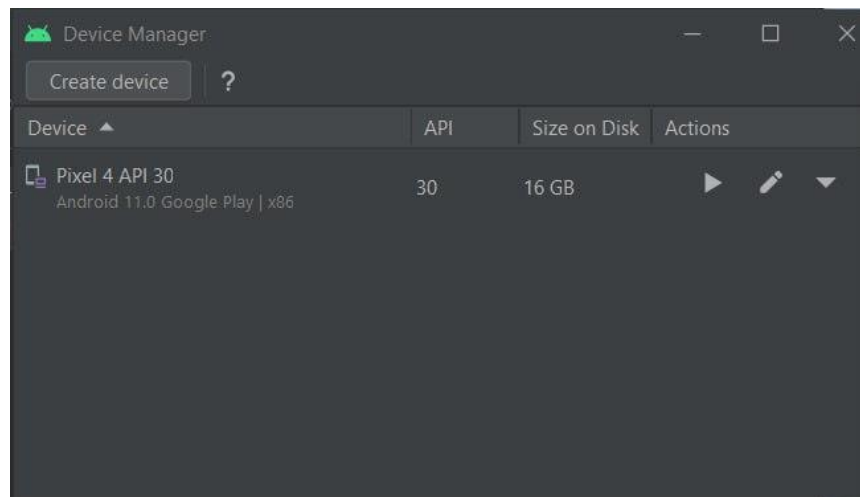


Рисунок 18 - Окно "DeviceManager"

Откроется окно с настройками нового эмулятора. Нужно выбрать подходящие характеристики на всех страницах и нажать на кнопку «Finish» (рисунок 19).

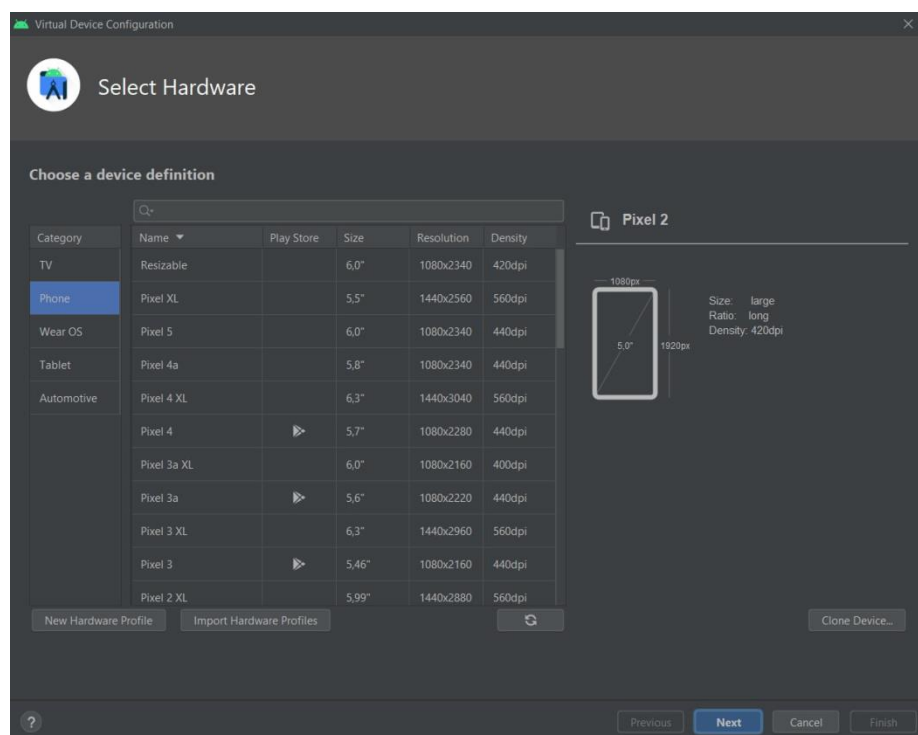


Рисунок 19 - Окно настройки эмулятора

– Нужно добавить в переменные среды значения (рисунок 9):

PATH:

C:\Users\[ИМЯПОЛЬЗОВАТЕЛЯ]\AppData\Local\Android\Sdk\platform-tools

ANDROID_HOME:

C:\Users\[ИМЯПОЛЬЗОВАТЕЛЯ]\AppData\Local\Android\Sdk

ANDROID_SDK_ROOT:

C:\Users\[ИМЯПОЛЬЗОВАТЕЛЯ]\AppData\Local\Android\Sdk

1.4.4 Установка *Node.js* и *Appium*

Для установки *Node.js* нужно скачать с официального сайта установщик (рисунок 20), запустить его и следовать инструкциям по установке.

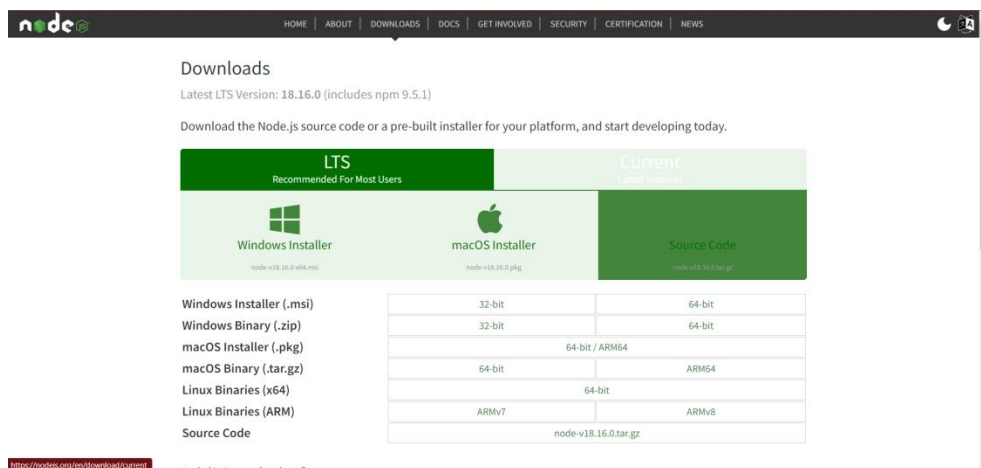


Рисунок 20 - Скачивание *Node.js*

После установки *Node.js* можно приступить к установке *Appium* с плагинами.

Для установки *Appium* нужно открыть командную строку и ввести команды, приведенные ниже.

```
npm uninstall -g appium
```

```
npm install -g appium@next
```

```
npm install -g webdriverio
```

```
appium driver list
```

```
appium driver install uiautomator2
```

```
appium driver install flutter
```

```
appium driver install xcuitest
```

```
appium plugin install execute-driver
```

Запуск Appium происходит с помощью команды `appium server --base-path /wd/hub/ --use-drivers=xcuitest,uiautomator2,flutter --use-plugins=execute-driver`

1.4.5 Скачивание и установка *IntelliJ IDEA*

Среда разработки (*IDE*) является важным инструментом для программистов. В настоящее время на рынке существует множество *IDE*, что может вызывать затруднения при выборе наиболее подходящей [17].

Критерии для выбора подходящей среды разработки:

- язык программирования. Первым шагом при выборе *IDE* является определение языка программирования, на котором будет выполняться работа. Каждая *IDE* предназначена для определенного языка программирования и может быть несовместима с другими языками;

- функциональность. Следующим шагом является определение необходимой для функциональности. Некоторые *IDE* предоставляют широкий спектр инструментов, таких как отладка, автодополнение кода, система контроля версий и т.д., в то время как другие *IDE* могут предоставлять более ограниченный набор функций;

- совместимость с операционной системой. Выбор *IDE* также может зависеть от операционной системы, которая используется. Некоторые *IDE* могут быть частично или полностью не совместимы с определенной операционной системой;

- удобство использования. Наконец, главный критерий при выборе *IDE* — это удобство использования. Выбранный *IDE* должен быть легким в использовании и иметь интуитивно понятный интерфейс пользователя, который позволит быстро и эффективно создавать, и отлаживать программы.

Несколько самых распространенных *IDE*:

- *Eclipse*. Кроссплатформенная *IDE*, разработанная для работы с языками *Java*, *C* и *C++*. Она оснащена широким набором утилит и инструментов, позволяющих программистам создавать, тестировать и отлаживать программы;

– *Visual Studio*. Кнтегрированная среда разработки для работы с языками *C#, C++, .NET* и другими языками. Она предоставляет программистам множество инструментов и возможностей для создания, тестирования и отладки кода, в том числе функции отката изменений и инструменты для разработки приложений для мобильных устройств;

– *IntelliJ IDEA*. Кроссплатформенная *IDE* для работы с языками *Java, Kotlin, Scala, Groovy* и другими языками. Она предоставляет широкий спектр инструментов для создания, тестирования и отладки программного кода, включая автодополнение, интеграцию с системами контроля версий и инструменты для работы с базами данных.

В завершение, выбор *IDE* зависит от многих факторов, таких как язык программирования, функциональность, совместимость с операционной системой и удобство использования. Важно осознавать потребности и требования к *IDE*, чтобы выбрать наиболее подходящий и эффективный инструмент для разработки программного обеспечения.

Для разработки автоматизированных тестов я использую *IntelliJ IDEA*.

Для скачивания *IntelliJ IDEA* нужно перейти на официальный сайт и скачать установщик (рисунок 21).

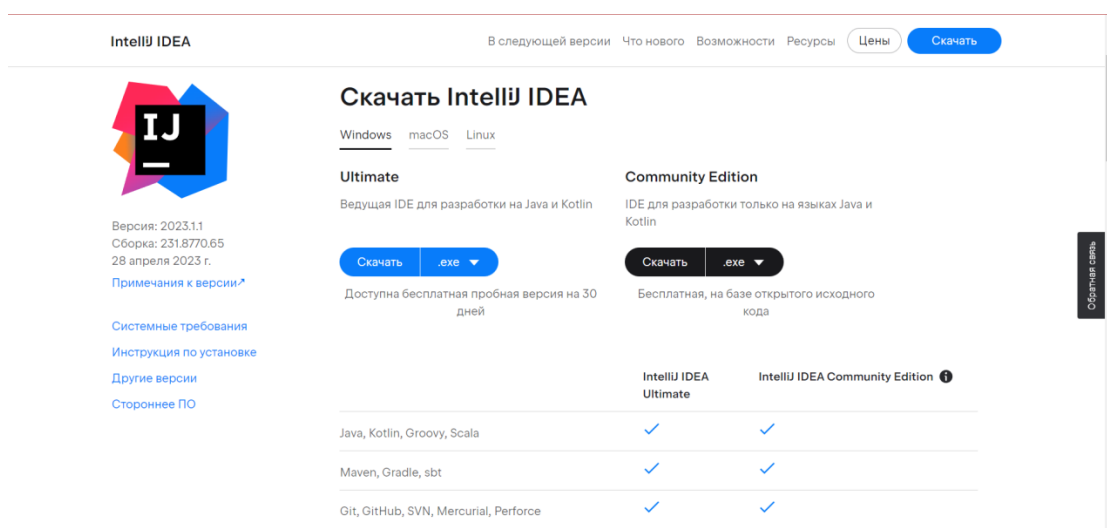


Рисунок 21 - Официальный сайт *IntelliJIDEA*

Для установки *IntelliJ IDEA* нужно запустить скачанный файл с расширением *.exe* и следовать всем инструкциям по установке.

1.4.6 Работа с системой контроля версий.

Система контроля версий — это инструмент, используемый в разработке программного обеспечения и других проектах, который позволяет отслеживать изменения в файловой системе или в других исходных кодах и сохранять информацию об изменениях в виде версий. Система контроля версий позволяет разработчикам работать вместе над проектом, фиксировать необходимые изменения и возвращаться к предыдущим версиям, если это необходимо[18].

Некоторые из популярных систем контроля версий это *Git*, *SVN* (*Subversion*), *Mercurial* и *CVS*.

Для работы над автоматизированными тестами в проекте используется *Git*.

Git — это распределенная система контроля версий, которая используется для управления изменениями в исходном коде программного обеспечения, а также других проектах и документации. Она была разработана Линусом Торвальдсом в 2005 году и с тех пор стала очень популярной в сообществе разработчиков.

Git позволяет разработчикам работать с локальным репозиторием кода, вносить изменения, создавать ветки, объединять изменения и делать другие операции, необходимые при разработке программного обеспечения.

Одной из преимуществ *Git* является распределенность, что означает, что каждый разработчик имеет полную копию репозитория. Это упрощает совместную работу, так как изменения могут быть записаны в локальной версии, а затем, после проверки, объединены с другими изменениями в удаленном репозитории.

Git имеет также широкий набор инструментов для работы с репозиториями, включая командную строку и графические клиенты для *Windows*, *Mac* и *Linux*.

1.5 Разработка автоматизированных тестов

Автоматизированный тест можно представить в виде абстрактного автомата.

Абстрактный автомат – это математический инструмент, описывающий систему в виде состояний и переходов между ними при воздействии на нее некоторым набором событий [19].

Представление автоматизированного теста в виде абстрактного автомата позволяет разработчикам лучше понимать как тест работает, какие данные он вводит и как он должен реагировать на некоторые события. Каждый тест может быть разбит на отдельные состояния, описывающие конкретные этапы проверки программного продукта.

Например, тест на проверку авторизации пользователя может быть разбит на следующие состояния: "Ввод данных пользователем", "Проверка правильности введенных данных", "Отправка данных на сервер", "Проверка ответа сервера". Каждое состояние описывает определенную часть теста и позволяет разработчику понимать, что происходит на каждом этапе проверки.

Переходы между состояниями описывают, как тест должен реагировать на определенные входные данные или действия системы. Например, если данные пользователя введены некорректно, то тест должен переместиться в состояние "Ввод данных пользователем" и запуститься заново. Если же данные введены правильно, то тест перейдет в следующее состояние и проверит следующий этап авторизации.

Для примера описания абстрактного автомата взят автоматизированный тест проверки сортировки на странице мобильного приложения:

```
@Test(priority = 3)
@Owner(Authors.MERENYASHEVA_EA)
public void testHomeworkResultSort() {
    setTestName("Проверка сортировки списка результатов выполнения ДЗ ");
    new BaseActions(driver);
```

```

switchContext("FLUTTER");
Step.checkLoginState(driver);
SchedulePageschedulePage = new SchedulePage(driver);
FirstPageDeterminantfirstPageDeterminant = new
FirstPageDeterminant(driver);

firstPageDeterminant.waitForExpectPage(FirstPageDeterminant.isMyScheduleAppPa
ge, Constant.TimeOut.MINUTE);

ResultHomeworkPageresultHomeworkPage = schedulePage
.checkElements()
.clickHomeworkPageButton()
.clickHomeworkRegister()
.checkElements()
.checkTaskActions()
.clickView()
.clickResult()
.checkListSortingResult(3)
.checkListSortingResult(2)
.checkListSortingResult(1)
.checkListSortingResult(0);
schedulePage.clickSchedulePageButton().checkElements();
}

```

Для данного автоматизированного теста абстрактный автомат представлен на рисунке 22.

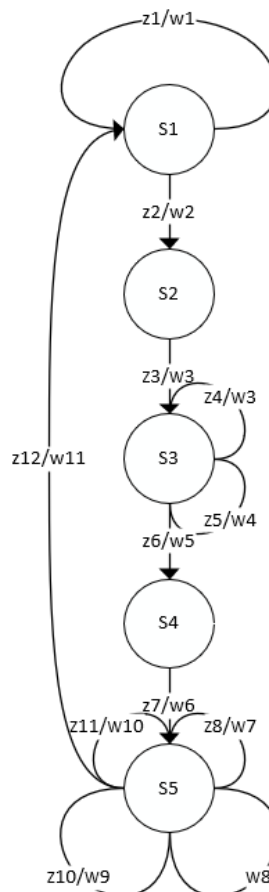


Рисунок 22 - Пример абстрактного автомата

Состояние «S1» представляет начальную вкладку при открытии приложения, «S2» – начальная страница на вкладке домашнего задания, «S3» – страница проверки домашнего задания на вкладке домашнего задания, «S4» – страница просмотра домашнего задания/Общие сведения, «S5» – страница просмотра домашнего задания/Результаты выполнения.

Каждое состояние автомата имеет переходы и петли: переходы описывают переход с одной страницы на другую, а петли, тестовые действия на странице.

Например, переход из состояния «S2» с сигналом «z3» ведёт в состояние «S3», что соответствует переходу к странице проверки домашнего задания на вкладке домашнего задания, петля с входным сигналом «z4» соответствует проверки элементов на этой странице, петля с входным сигналом «z5» соответствует проверки модального окна действий с домашним заданием.

В соответствии графом была построена таблица переходов автомата (таблица 1).

Таблица1 - Таблицапереходов

SM\zF	z1	z2	z3	z4	z5	z6	z7	Z8	Z9	Z10	Z11	Z12
S1	w1	w2										
S2			w3									
S3				w3	w4	w5						
S4							w6					
S5								w7	w8	w9	w10	w11

СКУ:

$$S_1(t+1) = z_1 S_1 \vee z_{12} S_5;$$

$$S_2(t+1) = z_2 S_1;$$

$$S_3(t+1) = z_3 S_2 \vee z_4 S_3 \vee z_5 S_3;$$

$$S_4(t+1) = z_6 S_3;$$

$$S_5(t+1) = z_7 S_4 \vee z_8 S_5 \vee z_9 S_5 \vee z_{10} S_5 \vee z_{11} S_5.$$

СВФ:

$$w_1(t) = z_1 S_1;$$

$$w_2(t) = z_2 S_1;$$

$$w_3(t) = z_2 S_2 \vee z_4 S_3;$$

$$w_4(t) = z_5 S_3;$$

$$w_5(t) = z_6 S_3;$$

$$w_6(t) = z_7 S_4 ;$$

$$w_7(t) = z_8 S_5 ;$$

$$w_8(t) = z_9 S_5 ;$$

$$w_9(t) = z_{10} S_5;$$

$$w_{10}(t) = z_{11} S_5;$$

$$w_{11}(t) = z_{12} S_5.$$

При написании автоматизированных тестов использовался фреймворк *testNG*.

TestNG — это фреймворк для тестирования приложений на языке *Java*. Он предоставляет расширенный функционал по сравнению с стандартными средствами тестирования такими, как *JUnit*. *TestNG* предназначен для упрощения и автоматизации процесса тестирования и предоставляет возможности для создания более сложных тестовых структур с использованием аннотаций и конфигурационного файла *XML* [20].

TestNG предоставляет следующие преимущества:

- поддержка многопоточного выполнения тестов;
- автоматическое определение зависимостей между тестами;
- более гибкое описание конфигурации тестов;
- поддержка параметризации тестов;
- интеграция с различными средствами контроля версий.

TestNG считается надежным и эффективным инструментом для тестирования приложений, который используют многие разработчики и тестировщики по всему миру.

С использованием *testNG* были описаны приоритет тестов и авторство для отчета *Allure*.

TestNG позволяет задавать приоритет тестов с помощью атрибута *priority* у аннотации *Test*. Приоритет может быть любым целым числом, при этом тесты будут запускаться в порядке увеличения приоритета. Синтаксис использования приоритета выглядит следующим образом:

`@Test(priority = 3)`

Для указания авторства тестов используется аннотация `@Owner`.

Такая аннотация может быть использована для:

- указания, какой автор написал или создал тестовый метод;

– помощи в поиске и отладке проблем, когда нужно установить, кто имеет больше опыта или знаний об этом тестовом методе, если возникнут ошибки при его выполнении;

– упрощения процесса обновления тестовых методов и их переноса между авторами, если это необходимо в процессе поддержки или разработки.

Можно использовать аннотацию `@Owner` как для методов, так и для классов тестов. В этом случае она будет применяться ко всем методам теста. Если аннотация указана как для класса, так и для конкретного метода теста, то значение параметра из аннотации метода будет иметь больший приоритет и перекроет значение параметра из аннотации класса.

Синтаксис использования указания авторства выглядит следующим образом:

```
@Owner(Authors.MERENYASHEVA_EA)
```

Данный тест наследуется от класса `BaseTest`:

```
@Epic("Тесты на проверку раздела «Задания»")
```

```
public class TestHomework extends BaseTest {
```

```
...
```

```
}
```

В классе `BaseTest` описываются методы для работы тестов, к примеру метод для автоматического запуска и остановки `AppiumServer` (приложение Б). Диаграмма классов представлена в приложении А.1.

В данном автоматизированном тесте все методы разбиты по классам. Каждый класс описывает определенную страницу.

Метод `checkElements()` относится к классу `SchedulePage` (класс, описывающий страницу расписания),

Метод `clickHomeworkPageButton()` относится к классу `NavBar` (класс, описывающий навигационную панель приложения),

Метод `clickHomeworkRegister()` относится к классу `HomeworkGivePage` (класс, описывающий страницу «Выдать ДЗ»),

Метод *checkElements()* относится к классу *HomeworkCheckPage* (класс, описывающий страницу «Проверить ДЗ»),

Метод *checkTaskActions()* относится к классу *HomeworkCheckPage* (класс, описывающий страницу «Проверить ДЗ»),

Метод *clickView()* относится к классу *HomeworkCheckPage* (класс, описывающий страницу «Проверить ДЗ»),

Метод *clickResult()* относится к классу *HomeworkGeneralInfoPage* (класс, описывающий страницу ДЗ\Общие сведения),

Метод *checkListSortingResult(3)* относится к классу *HomeworkResultPage* (класс, описывающий страницу ДЗ\Результаты),

Метод *checkListSortingResult(2)* относится к классу *HomeworkResultPage* (класс, описывающий страницу ДЗ\Результаты),

Метод *checkListSortingResult(1)* относится к классу *HomeworkResultPage* (класс, описывающий страницу ДЗ\Результаты),

Метод *checkListSortingResult(0)*; относится к классу *HomeworkResultPage* (класс, описывающий страницу ДЗ\Результаты),

Ниже приведен метод *checkTaskActions()*:

```
public HomeworkCheckPage checkTaskActions() {
```

```
    waitMobileElement(moreItemKey);
```

```
    clickMobileElement(moreItemKey);
```

```
    waitMobileElement(moreItemModalKey, Constant.TimeOut.SHORT);
```

```
    waitMobileElementText("Действия заданием");
```

```
    waitMobileElementText("Просмотреть");
```

```
    waitMobileElementText("Редактировать");
```

```
        //waitMobileElementText("Удалить"); TODO: сделать проверку в зависимости от вложенных заданий
```

```
    return this;
```

```
}
```

В данном методе используется комментарий *TODO* в *Java* для пометки фрагментов кода, которые нужно доработать или улучшить в будущем. Он

начинается со слова *TODO*, за которым следует описание того, что нужно сделать. Такие комментарии могут помочь разработчикам быстро найти места в коде, требующие внимания, и не забыть об их исправлении.

Методы *waitMobileElement*,

waitMobileElementText и *clickMobileElement* описаны в классе *BaseTest*. К примеру:

```
@Step(value = "Нажата на Flutter элемент {0}")
protected void clickMobileElement(String valueKey) {
    MobileElement element = (MobileElement)
driver.executeScript("flutter:waitForTappable", find.byValueKey(valueKey),
Constant.TimeOut.MEDIUM * 1000);
    element.click();
}

@Step(value = "Ожидание Flutter элемента {0}")
protected MobileElement waitMobileElementText(String text) {
    try {
        MobileElement temp = (MobileElement) driver
        .executeScript("flutter:waitFor", find.text(text), Constant.TimeOut.MEDIUM *
1000);
        } catch (WebDriverException webDriverEx) {
            if (checkCatErrorPage()) {
                String errorMessage = "";
                if (boolWaitMobileElement("keyErrorMessage",
Constant.TimeOut.SECOND)) {
                    errorMessage = waitMobileElement("keyErrorMessage").getText();
                }
                if (errorMessage.equals("")) {
                    Assert.fail("Ошибка,");
                } else {
                    Assert.fail("Ошибка " + errorMessage);
                }
            }
        }
    }
```

```

        } else if (waitSystemAlert()) ;
        else {
            Assert.fail("Не найден текст \"" + text + "\" за " + Constant.TimeOut.MEDIUM
+ " сек., ошибка:\n" + webDriverEx.getMessage()
        );
        }
    }
}

```

Полный код тестов и методов представлен в приложении В.

Для второго примера используется автоматизированный тест проверки прикрепленных материалов на странице «Выдать ДЗ»:

```

@Test(priority = 4)
@Owner(Authors.MERENYASHEVA_EA)
public void checkCheckingAttachedMaterials() {
    setTestName("Проверка прикрепленных материалов на странице \"Выдать
ДЗ\");
    Step.checkLoginState(driver);
    new FirstPageDeterminant(driver)
        .waitFirstExpectPage(FirstPageDeterminant.isMyScheduleAppPage,
Constant.TimeOut.MINUTE);
    TimeDevice timeDevices = new TimeDevices(driver);
    int count = 0;
    String dataTargetLesson;
    int randomNum = ThreadLocalRandom.current().nextInt(0, 1000);
    String descriptionHomework = "Добавление дз, со страницы выдать дз" +
randomNum;
    HomeworkGivePage homeworkGivePage = new SchedulePage(driver)
        .clickHomeworkPageButton()
        .clickSetHomework();
    count = homeworkGivePage.scrollToFieldCreateHomework();
    dataTargetLesson = homeworkGivePage.getDataTargetHomework(count);
}

```

```

homeworkGivePage.enterDescriptionHomework(descriptionHomework, count)
.clickBtnAddFile(count)
.clickBtnMakePhoto()
.createPhoto()
.clickBtnSetHomework(count)
.checkModalPageSetEveryone()
.clickInModalPageSetEveryone();

String      timeCreatedHomework      =      "Задано:      "      +
timeDevices.getTimeWhenCreateInHomework();

homeworkGivePage
.checkElementsSetHomework(count, timeCreatedHomework, dataTargetLesson,
"Всемуклассу", descriptionHomework, true)
.clickSchedulePageButton()
.checkElements();
}

```

Для данного автоматизированного теста абстрактный автомат представлен в приложении А.4.

В соответствии графом была построена таблица переходов автомата (таблица 2).

Таблица 2 - Таблица переходов

SM\zF	z1	z2	z3	z4	z5	z6	z7	Z8	Z9	Z10
S1	w1									w7
S2		w2								
S3			w2	w3	w4	w5			w7	
S4							w6	w2		

СКУ:

$$S_1(t+1) = z_{10} S_{1V} z_9 S_3;$$

$$S_2(t+1) = z_1 S_1;$$

$$S_3(t+1) = z_2S_2 \vee z_3S_3 \vee z_4S_3 \vee z_8S_4;$$

$$S_4(t+1) = z_6S_3 \vee z_7S_4.$$

СВФ:

$$w_1(t) = z_1S_1;$$

$$w_2(t) = z_2S_1 \vee z_3S_3 \vee z_8S_4;$$

$$w_3(t) = z_4S_3;$$

$$w_4(t) = z_5S_3;$$

$$w_5(t) = z_6S_3;$$

$$w_6(t) = z_7S_4;$$

$$w_7(t) = z_9S_3 \vee z_1S_1.$$

В данном автоматизированном тесте все методы разбиты по классам. Каждый класс описывает определенную страницу.

Метод *clickHomeworkPageButton()* относится к классу *NavBar*(класс, описывающий навигационную панель приложения),

Метод *clickSetHomework()* относится к классу *HomeworkGivePage*(класс, описывающий страницу «Выдать ДЗ»),

Метод *scrollToFieldCreateHomework()* относится к классу *HomeworkGivePage*(класс, описывающий страницу «Выдать ДЗ»),

Метод *enterDescriptionHomework(descriptionHomework, count)* относится к классу *HomeworkGivePage*(класс, описывающий страницу «Выдать ДЗ»),

Метод *clickBtnAddFile(count)* относится к классу *HomeworkGivePage*(класс, описывающий страницу «Выдать ДЗ»),

Метод *clickBtnMakePhoto()* относится к классу *HomeworkGivePage*(класс, описывающий страницу «Выдать ДЗ»),

Метод *createPhoto()* относится к классу *HomeworkGivePage*(класс, описывающий страницу «Выдать ДЗ»),

Метод *clickBtnSetHomework(count)* относится к классу *HomeworkGivePage*(класс, описывающий страницу «Выдать ДЗ»),

Метод *checkModalPageSetEveryone()* относится к классу *HomeworkGivePage*(класс, описывающий страницу «Выдать ДЗ»),

Метод `clickInModalPageSetEveryone()` относится к классу `HomeworkGivePage` (класс, описывающий страницу «Выдать ДЗ»).

Метод `checkElementsSetHomework(count, timeCreatedHomework, dataTargetLesson, "Всему классу", descriptionHomework, true)` относится к классу `HomeworkGivePage` (класс, описывающий страницу «Выдать ДЗ»).

Метод `clickSchedulePageButton()` относится к классу `NavBar` (класс, описывающий навигационную панель приложения).

Метод `checkElements()` относится к классу `SchedulePage` (класс, описывающий страницу расписания),

Ниже приведен метод `HomeworkGivePage.createPhoto()`:

```
@Step("Нажатие на кнопку 'Сфотографировать'")
public HomeworkGivePage.createPhoto() {
    driver.context("NATIVE_APP");
    clickAllow();

    MobileElement element2 = (MobileElement)
driver.findElementById("com.android.camera2:id/shutter_button");
    element2.click();

    waitFor(2000);

    element2 = (MobileElement)
driver.findElementById("com.android.camera2:id/done_button");
    element2.click();

    driver.context("FLUTTER");
    return this;
}
```

В данном методе используется `driver.context("NATIVE_APP")`, эта строка для переключения контекста приложения на нативный режим. При работе с мобильными приложениями с использованием *Appium*, можно использовать несколько типов контекста, к примеру: `NATIVE_APP` для работы с нативными элементами интерфейса пользователя и `WEBVIEW_<название приложения>` для работы с веб-контентом, который был встроен в приложение.

Строчка `driver.context("NATIVE_APP")` переключает контекст на нативный режим, чтобы можно было выполнять действия с элементами интерфейса эмулятора, такими как кнопки, поля ввода и т.д.

Так же используется `driver.context("FLUTTER")`, эта строчка позволяет скриптам на Java взаимодействовать с элементами *Flutter* и выполнять действия в приложении.

Для просмотра отчета о выполнении автоматизированных тестов используется фреймворк *Allure*.

Allure — это открытый фреймворк для создания, хранения и отображения результатов тестирования в удобном и красивом формате. Он позволяет создавать детальные отчеты о тестировании и предоставляет интерактивный пользовательский интерфейс для их просмотра.

Allure поддерживает различные языки программирования, такие как *Java*, *Python*, *Ruby*, *C#*, *PHP* и другие, и может быть использован с любым фреймворком для тестирования, таким как *JUnit*, *TestNG*, *PyTest*, *NUnit* и другими.

С помощью *Allure* можно получить детальную информацию о каждом тесте, включая входные данные, ожидаемый результат, фактический результат и скриншоты ошибок. Он также обеспечивает возможность фильтрации и поиска результатов тестирования, что упрощает поиск и устранение ошибок.

Allure предоставляет готовые отчеты, которые можно легко подключить к системам непрерывной интеграции и развернуть на сервере для общего доступа. Он также интегрируется с различными инструментами для автоматического тестирования, включая *Jenkins*, *TeamCity*, *Bamboo* и другие.

Для запуска отчета *Allure* нужно в IDE открыть вкладку *Maven*. *Maven* — это управляющий инструмент для *Java*-проектов, который упрощает управление зависимостями, сборку проектов и тестирование и поочередно выполнить `clean`, `validate`, `compile`, `test`. Для запуска страницы с отчетом следует выполнить `allure:serve` (рисунки 23).

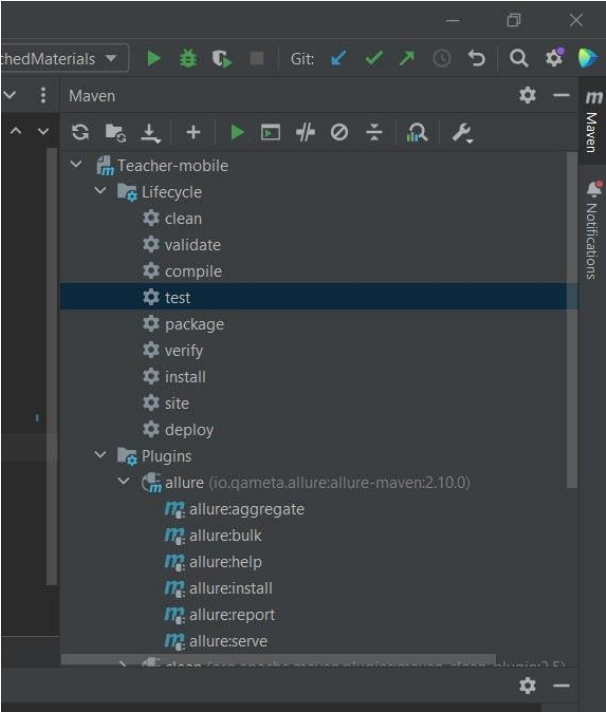


Рисунок 23 - Запуск отчета Allure

Интерфейс отчета Allure представлен на рисунке 24.

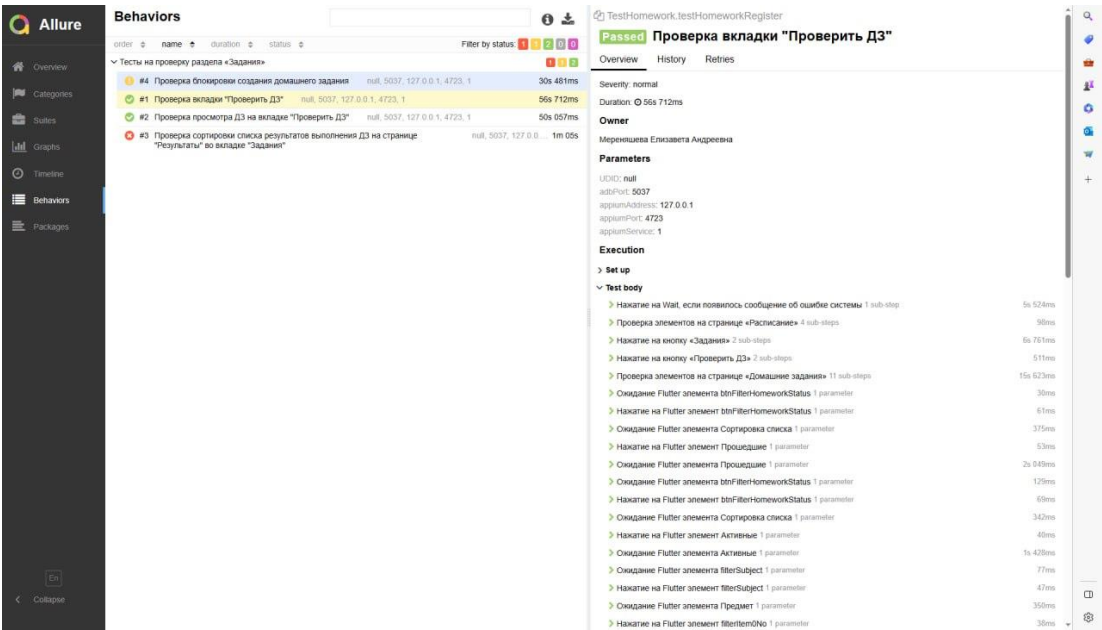


Рисунок 24 - Отчет Allure

2 Охрана труда

В этой части рассматриваются требования для обеспечения безопасности здоровья и охраны труда пользователей ПК, работающих в сфере разработки и использования автоматизированного тестирования, а также методы снижения негативного влияния на здоровье.

2.1 Характеристика потенциальных опасностей и вредностей, которые могут возникнуть при работе с ЭВМ

Потенциальные опасные и вредные последствия, которые могут возникнуть при работе с ПЭВМ, могут быть разнообразными и довольно серьезными. Например, длительное использование ПК может привести к утомлению глаз, болезням позвоночника, проблемам с сердцем и кровообращением. Отсутствие правильной эргономики в организации рабочего места может привести к повреждению зрения, головных болей и болезням мышц и костей.

Работа с ПК также может создавать угрозу для информационной безопасности. Неправильное использование компьютера может привести к утечке конфиденциальной информации и взлому систем безопасности. Кроме того, использование онлайн-сервисов может стать источником киберпреступлений и мошенничества.

Наконец, работа на ЭВМ может создавать опасности для физического здоровья человека. Нехватка физической активности и сидячий образ жизни могут привести к ожирению, диабету, болезням сердца и др. Внимательное отношение ко всем этим факторам и соблюдение необходимых мер предосторожности помогут снизить риски, связанные с работой на ПК.

Нормы помещения для работы с ПК в соответствии с техникой безопасности являются соответствующими требованиями к оборудованию рабочих мест, способствующим профилактике заболеваний, а также повышению производительности труда и качества работы.

					ПГУ 1.090401.13.001 ПЗ	Лист
						54
Изм.	Лист	№ докум.	Подпись	Дата		

2.1.1 Эргономика рабочего места

Эргономика, это наука, которая изучает связь между человеком и его рабочей средой, с целью создания максимально комфортных и безопасных условий для работы и предотвращения риска возникновения заболеваний опорно-двигательного аппарата, утомления и стресса. Эргономическое рабочее место включает в себя всю необходимую мебель и оборудование, которые обеспечивают правильную посадку и расположение тела человека в процессе работы. Кроме того, эргономическое рабочее место должно соответствовать задачам и особенностям работы, что позволяет повысить эффективность и продуктивность труда.

Современная жизнь заставляет нас проводить много времени за компьютером, что может приводить к различным заболеваниям. Чтобы снизить риск заболеваний и повысить производительность труда, необходимо заботиться об эргономике рабочего места.

Первым шагом к правильной организации рабочего места является выбор качественного и удобного стула с возможностью регулировки по высоте. Стул должен быть удобным, иметь спинку и поддержку для поясницы.

Для выбора эргономичного стула следует принять во внимание такие факторы, как:

- регулировка высоты. Стул должен быть легко регулируемым, чтобы можно было настроить высоту сиденья под свой рост, а также под высоту стола;
- поддержка поясницы. Стул должен иметь поддержку для поясницы, чтобы помочь поддерживать правильную осанку и уменьшить давление на нижнюю часть спины;
- регулировка наклона спинки. Стул должен иметь возможность регулировки наклона спинки, чтобы можно было изменять позу и уменьшать напряжение на спине;
- регулируемые подлокотники. Регулируемые подлокотники помогают поддерживать руки и снижать напряжение в плечах и шее;

– дышащие материалы. Стул должен быть изготовлен из материалов, которые позволяют телу "дышать", таким образом уменьшая образование пота и дискомфорта.

Вторым шагом к правильной организации рабочего места является выбор клавиатуры и мыши. Их следует располагать на расстоянии не менее 10 см от края стола, чтобы руки смогли свободно передвигаться. Клавиатуру желательно выбирать с небольшой высотой и наклоном.

Третьим шагом к правильной организации рабочего места является выбор монитора. Монитор должен находиться на уровне глаз, так, чтобы ни шея, ни глаза не уставали. Желательно, чтобы удаление между глазами и экраном было от 50 до 70 сантиметров. Также рекомендуется, чтобы верхняя граница монитора была не выше уровня глаз, а нижняя граница – не ниже уровня середины грудной клетки. Оптимальный угол наклона монитора 10-20 градусов вверх от горизонтальной поверхности. При работе за компьютером рекомендуется делать перерывы и проводить гимнастику глаз и шеи, чтобы избежать усталости и развития заболеваний опорно-двигательной системы

По нормам БЖД освещение должно быть равномерным и неярким. Желательно выбрать гибкий источник света, который можно направить на рабочую поверхность. Дневной свет, который попадает в комнату, следует разбить с помощью жалюзи или штор, чтобы избежать бликов на экране монитора.

Освещение при работе с ПК должно соответствовать следующим требованиям:

– яркость освещения должна быть не менее 300 лк на поверхности рабочего стола в случае работы с документами и не менее 500 лк при работе с монитором;

– светильники должны быть расположены таким образом, чтобы рабочая зона была равномерно освещена, без ярких бликов на поверхности монитора;

- не допускается использование флуоресцентных ламп с частотой пульсации света менее 100 Гц и люминесцентных ламп в низковольтных зажигах;
- цветовая температура света должна быть не менее 4000К и не более 6500К;
- необходимо обеспечить защиту от слепящего действия светильников на монитор и отражения света от стен и других поверхностей;
- рабочее место должно быть дополнительно оснащено настольной лампой для индивидуального освещения, которая должна иметь глушение нижней части лампы, чтобы не создавать бликов на поверхности монитора.

Важно помнить, что правильное освещение при работе с ПК является необходимым условием сохранения здоровья глаз и предотвращения утомления и слабости при длительной работе за компьютером.

Важно следить за тем, чтобы помещение, где располагается компьютер, было достаточно вентилируемым и прохладным. Общепринятой нормой БЖД для зданий, включая офисы, является поддержание температуры в диапазоне от 20 до 24 градусов Цельсия, при относительной влажности воздуха от 30 до 60%. Также для комфортной и безопасной для здоровья работы необходимо проводить перерывы в работе и выполнять разминку. Для профилактики заболеваний связанных с работой на ПК необходимо установить перерывы в работе каждые 45-60 минут. В эти перерывы можно включать специальные упражнения, направленные на разминку мышц и снятие усталости.

Таким образом, соблюдение правил эргономики рабочего места является одним из главных мероприятий для предотвращения заболеваний, связанных с работой на компьютере, и повышения производительности труда.

2.2 Оценка уровня шума в рабочем помещении

С точки зрения физиологии, шум представляет собой беспокойную смесь звуков, которые могут повлиять на нормальную жизнедеятельность и отдых человека. Звуковые волны возникают из-за различных колебательных процессов в акустической среде. Человек может слышать звуки в диапазоне частот от 16 до

20000 Гц. Колебания с частотой ниже 16 Гц называются инфразвуком, а ультразвуком - с частотой выше 20 кГц, и в определенной мере они могут оказывать воздействие на здоровье человека.

Основной характеристикой шума, необходимой для гигиенической оценки и проведения акустических расчетов, является уровень звукового давления L_p , который измеряется по определенной зависимости.

$$L_p = 20 \lg \frac{P}{P_0}, \text{ дБ}$$

где: P – звуковое давление в точке пространства, Па;

P_0 – пороговое звуковое давление, равное $2 \cdot 10^{-5}$ Па. Оно является нижним порогом слышимости человека на частоте 1000 Гц.

Шум, воздействуя на центральную нервную систему, неблагоприятно влияет на функционирование всего организма человека. Это может проявляться в виде головной боли, головокружения и снижения концентрации внимания, что может быть опасным и привести к несчастному случаю.

Область восприятия человеком уровня звукового давления находится в интервале от 0 до 130 дБ, где 0 дБ – порог слышимости; 130 дБ – болевой порог.

В зависимости от временных характеристик, шум можно разделить на несколько категорий:

- постоянный шум, уровень которого не изменяется более чем на 5 дБА в течение 8-часового рабочего дня;

- непостоянный шум, уровень которого изменяется более чем на 5 дБА в течение 8-часового рабочего дня. Этот тип шума можно подразделить на три вида:

- 1) колеблющийся во времени шум, уровень которого непрерывно меняется;

- 2) прерывистый шум, уровень которого меняется ступенчато (на 5 дБА и более) и длительность интервалов, в течение которых уровень шума остается постоянным, составляет 1 секунду и более;

3) импульсный шум, который состоит из одного или нескольких звуковых сигналов, каждый из которых длительностью менее 1 секунды и превышает фоновый уровень шума более чем на 7 дБА.

Для измерения уровня звука в данной точке помещения по уровням звука от различных источников шума $L_{общ}$ производится по формуле:

$$L_{общ} = 10 \lg(10^{0,1L_1} + 10^{0,1L_2} + \dots + 10^{0,1L_n}),$$

где: $L_1, L_2, \dots, L_n$ – уровни звука в расчётной точке от различных источников определяются по формуле:

$$L_i = L_{ai} + 10 \lg\left(\frac{1}{S_i} + \frac{4}{B}\right), \quad (1)$$

Величина $\Delta = 10^{0,1L_i}$ является табличным значением.

Используя формулу (1) и измеренные данные, вычисляем уровень шума в помещении:

$$L_1 = 55 \text{ дБА} + 10 \lg\left(\frac{1}{50,24 \text{ м}^2} + \frac{4}{30 \text{ м}^2}\right) = 44,85 \text{ дБА},$$

$$L_2 = 48 \text{ дБА} + 10 \lg\left(\frac{1}{3,14 \text{ м}^2} + \frac{4}{30 \text{ м}^2}\right) = 44,54 \text{ дБА},$$

$$L_{общ.эксп.} = 10 \lg(10^{0,1 \cdot L_1} + 10^{0,1 \cdot L_2}),$$

$$L_{общ.эксп.} = 10 \lg(10^{0,1 \cdot 44,85 \text{ дБА}} + 10^{0,1 \cdot 44,54 \text{ дБА}}) = 47,7 \text{ дБА}.$$

ПДУ = 60 дБА у рабочего места. Вычисленное значение не превышает предельно допустимый уровень, поэтому в моем случае не нужно применять меры для понижения шума.

Заключение

В рамках данной работы была разработана система функционального автоматизированного тестирования мобильного приложения для педагога. Данная система автоматизированного тестирования показала свою эффективность и практическую значимость.

Разработанный программный продукт позволяет значительно сократить время тестирования, а также повысить его точность и надежность. Благодаря использованию средств автоматизации тестирования, удастся сократить ручную работу персонала и снизить ошибки в процессе проверки.

Итогом проведенной работы являются следующие результаты: исследован материал, касающийся следующих предметных областей:

- виды и направления тестирования;
- функциональное тестирование, направления функционального тестирования;
- автоматизированное тестирование, направления автоматизированного тестирования;
- фреймворки и другие программные средства для реализации автоматизированного тестирования.

В данной работе были использованы современные технологии, предусмотрены возможности расширения и развития системы в будущем.

В ходе проектирования разработаны диаграмма классов, прецедентов и абстрактный автомат алгоритма.

Результаты исследования имеют как практическую, так и научную ценность. Они были использованы в научно-исследовательской работе ООО "Ай-Теко ВЦ" для улучшения рабочего процесса. Кроме того, эти результаты могут быть применены для тестирования мобильных приложений, аналогичных по бизнес-логике.

Таким образом, можно сделать вывод, что разработанная система автоматизированного тестирования мобильного приложения является

полноценным и эффективным инструментом для проверки работоспособности мобильного приложения.

					ПГУ 1.090401.13.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		61

Список использованных источников

1. Куликов С. С. Тестирование программного обеспечения. – 2017.
2. Матвеева Н. А., Герасимов В. В., Игнатьева Д. О. Исследование технологий модульного тестирования на платформе Java //Системні технології. – 2016. – №. 1. – С. 49-55.
3. Садов С. А. ЭФФЕКТИВНОСТЬ РЕГРЕССИОННОГО ТЕСТИРОВАНИЯ В МОБИЛЬНЫХ ПРИЛОЖЕНИЯХ //Прикладная математика и информатика: современные исследования в области естественных и технических наук. – 2021. – С. 55-59.
4. Лопатина К. И., Темникова Е. А. Ручное и автоматизированное тестирование программного обеспечения отличия и взаимозаменяемость //Вестник современных исследований. – 2017. – №. 10-1. – С. 75-77.
5. Рыбалко М. А., Иванова Е. А. Тестирование программного обеспечения, методы тестирования //Информационное общество: современное состояние и перспективы развития. – 2017. – С. 320-322.
6. Воробьев Н. А., Бурмин Л. Н., Степанов Ю. А. Сравнительный анализ средств тестирования мобильных приложений //Евразийский союз ученых. – 2020. – №. 6-1 (75). – С. 36-38.
7. Singh S., Gadgil R., Chudgor A. Automated testing of mobile applications using scripting technique: A study on appium //International Journal of Current Engineering and Technology (IJCET). – 2014. – Т. 4. – №. 5. – С. 3627-3630.
8. Козлов С. В., Машенков А. С. ТЕСТИРОВАНИЕ ПРОГРАММНЫХ ПРИЛОЖЕНИЙ НА УСТРОЙСТВАХ ANDROID С ИСПОЛЬЗОВАНИЕМ ADB. – 2022.
9. Hagos T., Hagos T. Android studio //Learn Android Studio 3: Efficient Android App Development. – 2018. – С. 5-17.
10. Кукушкин Ю. М., Ильина В. Ю. Сравнительный анализ и проблема выбора эмуляторов Android //Конвергентные технологии XXI: вариативность, комбинаторика, коммуникация. – 2020. – С. 262-267.

11. Ачилов Р. Эмуляторы Android. Общий обзор. BlueStacks //Системный администратор. – 2013. – №. 1-2. – С. 18-22.
12. Okken B. Python Testing with pytest. – Pragmatic Bookshelf, 2022.
13. Hordiienko Y. Migration of legacy unit tests in C# programs :дис. – InstytutInformatyki, 2022.
14. Загидуллин Р. А. ЯЗЫК ПРОГРАММИРОВАНИЯ JAVA //Редакционный совет. – 2017. – Т. 43. – №. 3. – С. 889.
15. Нимейер П., Леук Д. Программирование на Java. – Litres, 2022.
16. Пенкина Д. П. РАЗРАБОТКА ПРОДУКЦИОННОЙ МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ В ПРЕДМЕТНОЙ ОБЛАСТИ «АВТОМАТИЗИРОВАННОЕ ТЕСТИРОВАНИЕ» //Актуальные вопросы науки и практики и перспективы их решений. – 2023. – С. 62-66.
17. Чуйкова А. Г., Игрунова С. В. Интеллектуальная система выбора среды разработки (IDE) //Молодежный научный форум: технические и математические науки. – 2017. – №. 6. – С. 183-190.
18. Шурыгин В. Н., Сиваченко Д. А. Системы контроля версий //Вестник Московского государственного университета печати. – 2015. – №. 5. – С. 103-104.
19. Сергина И. Г. Способы представления конечного автомата //Информационные технологии в науке и образовании. Проблемы и перспективы. – 2020. – С. 97-100.
20. George S., Nagaraja G. S. Automated Framework for API Testing. – 2020.

Приложение А

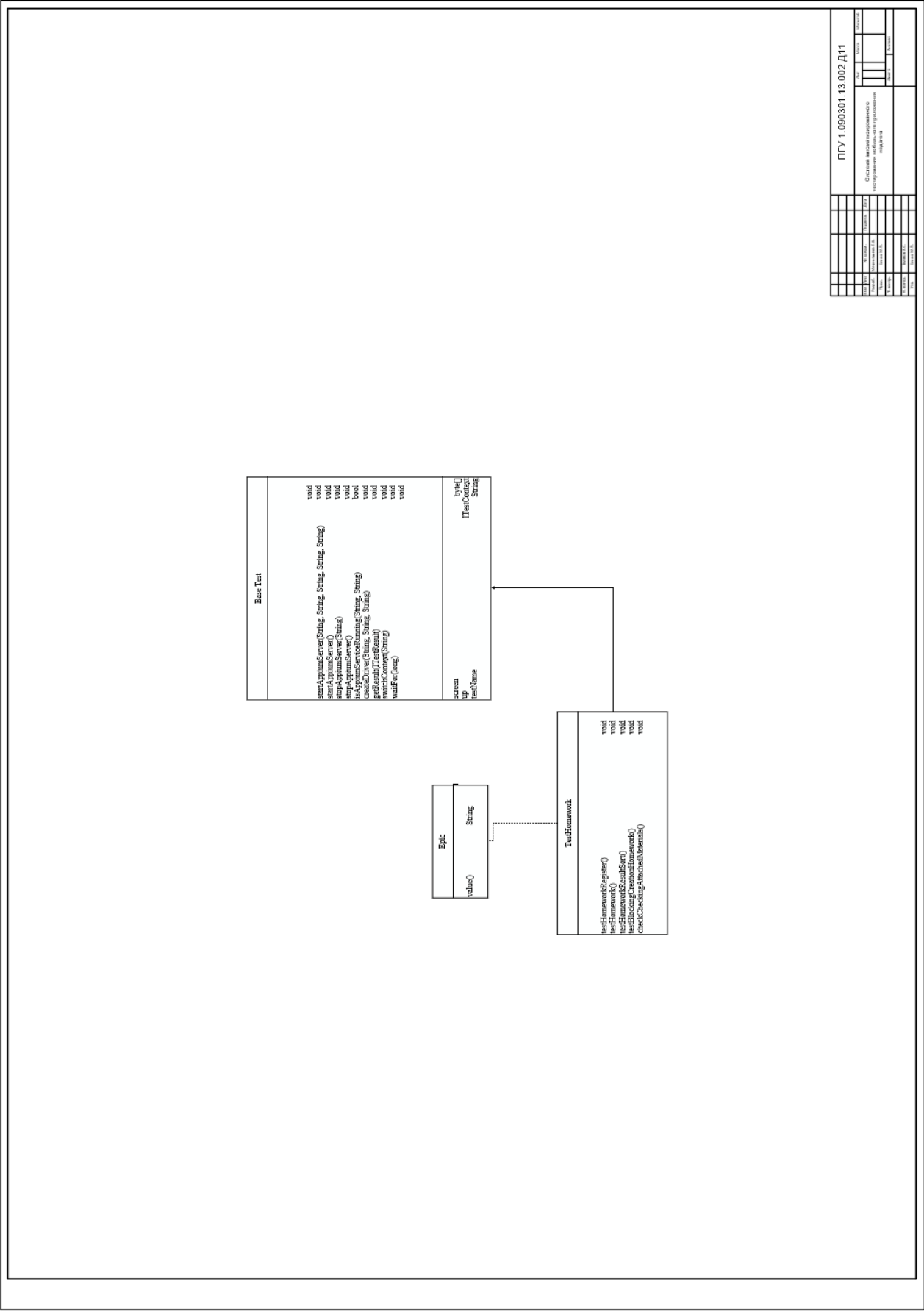
(обязательное)

Система автоматизированного тестирования мобильного приложения педагога

UML-диаграммы

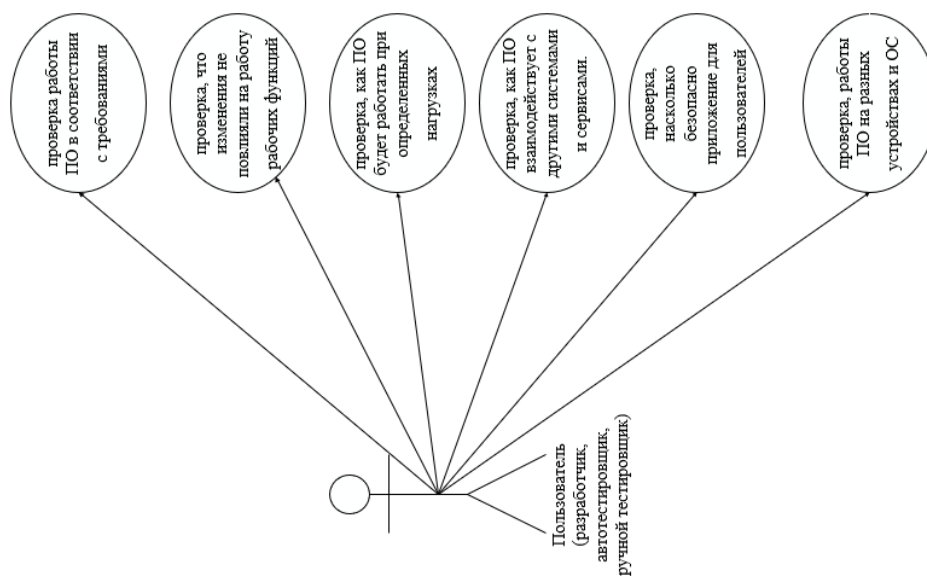
					ПГУ 1.090401.13.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

Приложение А.1 - Диаграмма классов



ПГУ 1.090301.13.002 Д11									
Система автоматизированного тестирования приложений									
История изменений									
№	№ докум.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.	ИЗМ.
1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9
10	10	10	10	10	10	10	10	10	10

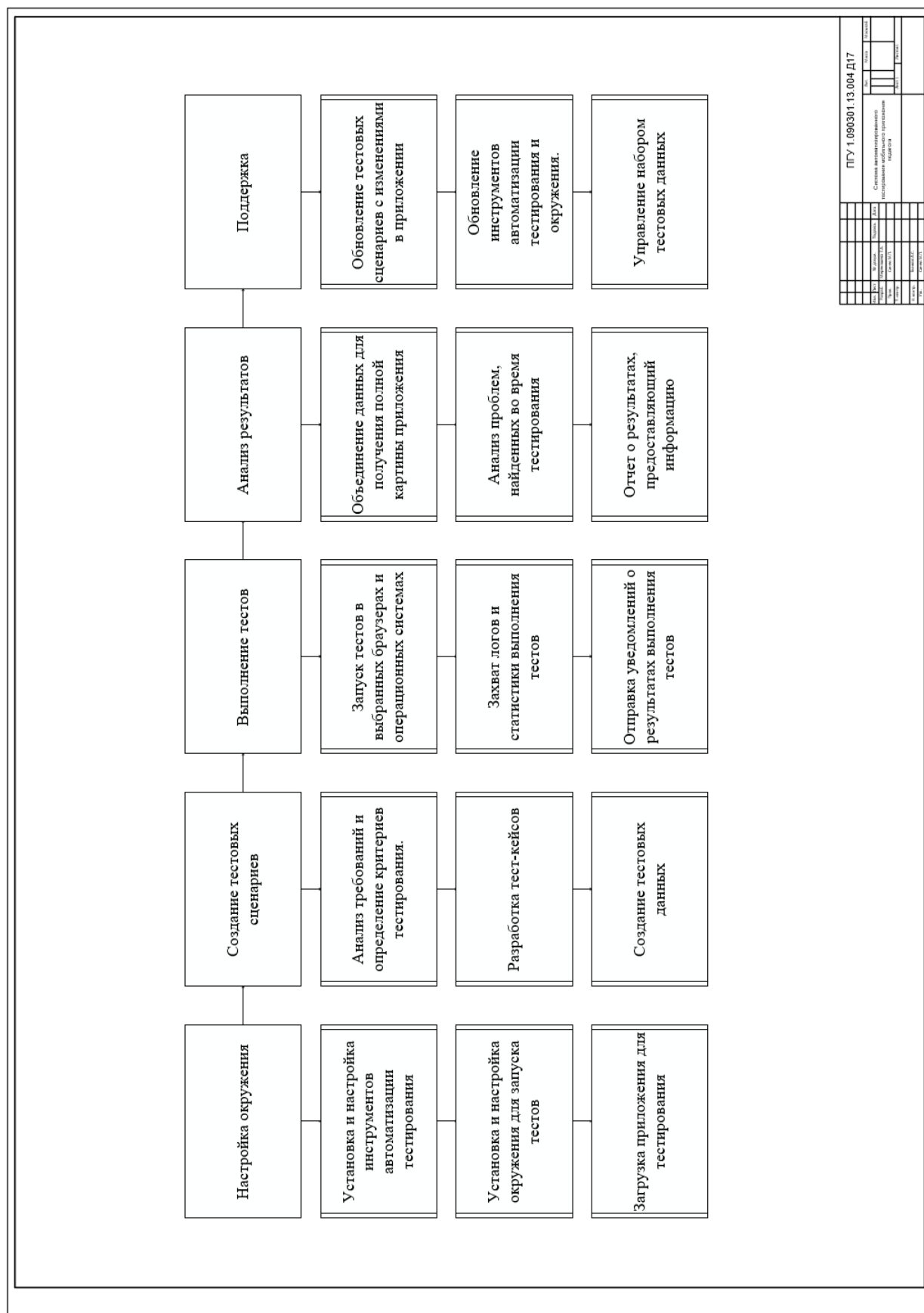
Приложение А.2 - Диаграмма прецедентов

[illegible]

Изм.	Лист	№ докум.	Подпись	Дата

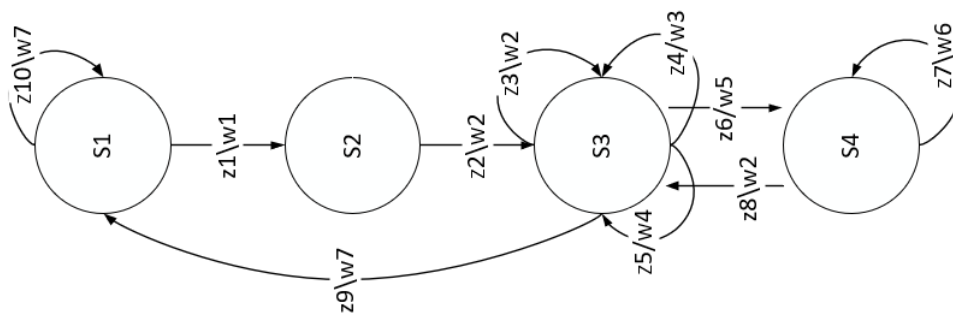
ПГУ 1.090401.13.001 ПЗ

Приложение А.3 - Блок схема этапов автоматизированного тестирования



ПГУ 1.090301.13.004 Д17									
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя
№	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя

Приложение А.4—Диаграмма состояний



ПГУ 1.090301.13.005 Д16									
Система автоматизированного проектирования объектов и процессов									
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя
Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя	Имя

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090401.13.001 ПЗ

Приложение Б

(обязательное)

Система автоматизированного тестирования мобильного приложения педагога

Листинг программы

					ПГУ 1.090401.13.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		69

Файл testHomework.java

```
import base.BaseTest;
import constant.Authors;
import constant.Constant;
import io.qameta.allure.Epic;
import io.qameta.allure.Owner;
import org.testng.annotations.Test;
import pages.TimeDevices;
import pages.base.BaseActions;
import pages.homework.HomeworkCheckPage;
import pages.homework.HomeworkEditPage;
import pages.homework.HomeworkGivePage;
import pages.homework.infoTaskPage.HomeworkResultPage;
import pages.register.ClassRegisterPage;
import pages.schedule.SchedulePage;
import steps.FirstPageDeterminant;
import steps.PreparationTestAddHomework;
import steps.Step;

import java.io.IOException;
import java.util.concurrent.ThreadLocalRandom;

@Epic("Тесты на проверку раздела «Задания»")
public class TestHomework extends BaseTest {

    @Test(priority = 1)
    @Owner(Authors.MERENYASHEVA_EA)
    public void testHomeworkRegister() {

        setTestName("Проверка вкладки \"Проверить ДЗ\");
        Step.checkLoginState(driver);

        new FirstPageDeterminant(driver)
            .waitFirstExpectPage(FirstPageDeterminant.isMyScheduleAppPage,
                Constant.TimeOut.MINUTE);

        new SchedulePage(driver)
            .checkElements()
            .clickHomeworkPageButton()
            .clickHomeworkRegister()
            .checkElements()
            .checkFilterHomeworkStatus("Прошедшие")
            .checkFilterHomeworkStatus("Активные")
            .checkFilterSubject()
            .checkFilterParallel()
            // .checkFilterCheckDate() // Фильтр временно убран
            .checkListSorting("По дате проверки (по убыванию)")
            .checkListSorting("По дате проверки (по возрастанию)")
            .checkListSorting("По дате задания (по убыванию)");
    }
}
```

					ПГУ 1.090401.13.001 ПЗ	Лист
						70
Изм.	Лист	№ докум.	Подпись	Дата		

```

.checkListSorting("По дате задания (по возрастанию)")
.clickSchedulePageButton()
.checkElements();
    }

    @Test(priority = 2)
    @Owner (Authors.MERENYASHEVA_EA)
    public void testHomework() {

        setTestName("Проверка просмотра ДЗ на вкладке \"Проверить ДЗ\");
        Step.checkLoginState(driver);

        new FirstPageDeterminant(driver)
        .waitFirstExpectPage(FirstPageDeterminant.isMyScheduleAppPage,
        Constant.TimeOut.MINUTE);

        new SchedulePage(driver)
        .checkElements()
        .clickHomeworkPageButton()
        .clickHomeworkRegister()
        .checkElements()
        .checkTaskActions()
        .clickView()
        .checkViewHomework()
        .clickSchedulePageButton()
        .checkElements();
    }

    @Test(priority = 3)
    @Owner (Authors.MERENYASHEVA_EA)
    public void testHomeworkResultSort() {

        setTestName("Проверка сортировки списка результатов выполнения ДЗ на
        странице \"Результаты\" во вкладке \"Задания\");
        Step.checkLoginState(driver);

        SchedulePageschedulePage = new SchedulePage(driver);
        FirstPageDeterminantfirstPageDeterminant = new
        FirstPageDeterminant(driver);

        firstPageDeterminant.waitFirstExpectPage(FirstPageDeterminant.isMySc
        heduleAppPage, Constant.TimeOut.MINUTE);

        HomeworkResultPagehomeworkResultPage = schedulePage
        .checkElements()
        .clickHomeworkPageButton()
        .clickHomeworkRegister()
        .checkElements()
        .checkTaskActions()

```

```

.clickView()
.clickResult()
.checkListSortingResult(3)
.checkListSortingResult(2)
.checkListSortingResult(1)
.checkListSortingResult(0);
schedulePage.clickSchedulePageButton().checkElements();
    }

    @Test(priority = 4)
    @Owner(Authors.MERENYASHEVA_EA)
    public void testBlockingCreationHomework() {

setTestName("Проверкаблокировкисозданиядомашнегозадания");
Step.checkLoginState(driver);

FirstPageDeterminantfirstPageDeterminant = new
FirstPageDeterminant(driver);
SchedulePageschedulePage = new SchedulePage(driver);

firstPageDeterminant.waitForExpectPage(FirstPageDeterminant.isMySc
heduleAppPage, Constant.TimeOut.MINUTE);
ClassRegisterPageclassRegisterPage = schedulePage
.checkElements()
.clickRegisterPageButton()
.checkElements()
.clickClassRegisterBtnNoLesson()
.checkElements()
.clickHomework()
.scrollToBtnAddHomeworkOnList()
.clickCreateHomeworkNoLesson()
.checkIssuingAtTaskPage()
.clickUnderstandIssuingAtTask();
schedulePage.clickSchedulePageButton().checkElements();
    }

    @Test(priority = 5)
    @Owner(Authors.MERENYASHEVA_EA)
    public void checkCheckingAttachedMaterials() {
setTestName("Проверкаприкрепленныхматериаловнастранице
\"ВыдатьДЗ\");

Step.checkLoginState(driver);

        new FirstPageDeterminant(driver)
        .waitForExpectPage(FirstPageDeterminant.isMyScheduleAppPage,
Constant.TimeOut.MINUTE);

TimeDevicestimeDevices = new TimeDevices(driver);

```

					ПГУ 1.090401.13.001 ПЗ	Лист
						72
Изм.	Лист	№ докум.	Подпись	Дата		

```

        int count = 0;
        String dataTargetLesson;
        int randomNum = ThreadLocalRandom.current().nextInt(0,
1000);
        String descriptionHomework = "Добавлениедз,
со страницывыдатьдз" + randomNum;
        HomeworkGivePagehomeworkGivePage= newSchedulePage(driver)
        .clickHomeworkPageButton()
        .clickSetHomework();
        count = homeworkGivePage.scrollToFieldCreateHomework();

        dataTargetLesson = homeworkGivePage.getDataTargetHomework(count);
        homeworkGivePage.enterDescriptionHomework(descriptionHomework,
count)
        .clickBtnAddFile(count)
        .clickBtnMakePhoto()
        .createPhoto()
        .clickBtnSetHomework(count)
        .checkModalPageSetEveryone()
        .clickInModalPageSetEveryone();

        String timeCreatedHomework = "Задано: " +
timeDevices.getTimeWhenCreateInHomework();
        homeworkGivePage
        .checkElementsSetHomework(count, timeCreatedHomework,
dataTargetLesson, "Всеукрацы", descriptionHomework, true)
        .clickSchedulePageButton()
        .checkElements();
    }
}

```

Файл baseTest.java

```
package base;
import common.Config;
import common.Driver;
import constant.Constant;
import emulator.AndroidEmulator;
import io.appium.java_client.MobileElement;
import io.appium.java_client.android.AndroidDriver;
import io.appium.java_client.service.local.AppiumDriverLocalService;
import io.appium.java_client.service.local.AppiumServiceBuilder;
import io.appium.java_client.service.local.flags.GeneralServerFlag;
import io.qameta.allure.Allure;
import io.qameta.allure.AllureLifecycle;
import io.qameta.allure.Attachment;
import org.openqa.selenium.OutputType;
import org.testng.ITestContext;
import org.testng.ITestNGMethod;
import org.testng.ITestResult;
import org.testng.annotations.*;

import java.io.File;
import java.io.IOException;
import java.net.InetAddress;
import java.net.MalformedURLException;
import java.net.ServerSocket;
import java.util.Set;
import java.util.concurrent.TimeUnit;

import static common.Config.*;

public class BaseTest {

    public static AndroidDriver<MobileElement> driver;
    public static AndroidDriver<MobileElement> driverNative;
    private static AppiumDriverLocalService serviceAppium = null;
    private static boolean bAppiumService = false;

    @BeforeSuite
    public void setUp(ITestContext context) {
        if (Config.RETRY_TESTS) {
            for (ITestNGMethod method : context.getAllTestMethods()) {
                method.setRetryAnalyzerClass(TestNGRetry.class);
            }
        }
    }

    @BeforeSuite
    @Parameters({"appiumAddress", "appiumPort", "adbPort", "UDID",
        "appiumService"})
```

					ПГУ 1.090401.13.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

```

        public void startAppiumServer(
            @Optional("127.0.0.1") String appiumAddress,
            @Optional("4723") String appiumPort,
            @Optional("5037") String adbPort,
            @Optional("") String udid,
            @Optional("1") String enableAppiumService
        ) throws MalformedURLException {
System.out.println("udid:" + udid + "");
System.out.println("ADB port:" + adbPort);
System.out.println("Appium ip:" + appiumAddress);
System.out.println("Appium port:" + appiumPort);
System.out.println("Appium service enable:" + enableAppiumService);

        if (enableAppiumService.equals("1") && RUN_APPIUM_SERVER) {
bAppiumService = true;
            if (serviceAppium != null && serviceAppium.isRunning()) {
serviceAppium.stop();
            }

AppiumServiceBuilder builder;
            builder = new AppiumServiceBuilder()
                .withArgument(() -> "--base-path", "/wd/hub")
                .withArgument(() -> "--use-drivers", "uiautomator2,flutter")
                .withArgument(() -> "--allow-insecure", "chromedriver_autodownload")
                .withLogFile(new File("target/appium_server.log"))
                .withIPAddress(appiumAddress)
                .usingPort(Integer.parseInt(appiumPort))
                .withArgument(GeneralServerFlag.SESSION_OVERRIDE)
                .withArgument(GeneralServerFlag.LOG_TIMESTAMP)
                .withArgument(GeneralServerFlag.DEBUG_LOG_SPACING)
                .withArgument(GeneralServerFlag.LOG_LEVEL,
                    (DEBUG_MODE && APPIUM_SERVE_LOG_CONSOLE)
? "debug:debug" : "error:debug")
                .withStartUpTimeout(120, TimeUnit.SECONDS);

System.out.println("Info: Appium service start");
            if (isAppiumServiceRunning(appiumAddress, appiumPort)) {
System.out.println("Info: Appium already launch");
stopAppiumServer(appiumPort);
System.out.println("Info: Appium try stop");
                if (isAppiumServiceRunning(appiumAddress,
appiumPort)) {
System.out.println("Info: Appium yet launch");
                } else {
System.out.println("Info: Appium stopped");
                }
            }

        }

        serviceAppium = AppiumDriverLocalService.buildService(builder);
    }

```

```

serviceAppium.start();
System.out.println("Info: Appium service launch");
    }
}

    public void stopAppiumServer(String port) {
String[] commands;
    String os = System.getProperty("os.name").toLowerCase();
    if (os.contains("win")) {
        commands = new String[]{"cmd", "/c", "netstat -ano |
findstr :\" + port + \" | findstr LISTENING | FOR /F \"tokens=5\" %a
in ('more') do taskkill /PID %a /F"};
    } else {
        commands = new String[]{"bash", "-c", "lsof \\-i :\" +
port + \" -t | xargs kill"};
    }
    try {
Runtime.getRuntime().exec(commands).waitFor();
    } catch (IOException | InterruptedException e) {
e.printStackTrace();
    }
}

    @AfterSuite
    public void stopAppiumServer() {
        if (bAppiumService&&serviceAppium.isRunning()) {
serviceAppium.stop();
serviceAppium = null;
        }
    }

    private boolean isAppiumServiceRunning(String ip, String port) {
boolean isServerRunning = false;
ServerSocket serverSocket;
    try {
serverSocket = new ServerSocket(Integer.parseInt(port), 50,
InetAddress.getByAddress(ip));
serverSocket.close();
    } catch (IOException e) {
isServerRunning = true;
    } finally {
serverSocket = null;
    }
    return isServerRunning;
}

    @BeforeMethod
    @Parameters({"address", "adbPort", "UDID"})

```

```

        public void createDriver(@Optional("127.0.0.1") String
addressAppium,
                                @Optional("5037") String adbPort,
                                @Optional("") String udid)
                                throws MalformedURLException {
        if (driver == null) {

                if (!AndroidEmulator.isEmulatorRunning()) {
System.out.println("Error: Emulator not found or not running");
                return;
                }

driverNative = Driver.createDriverNative(addressAppium, adbPort,
udid);

                if (driverNative != null) {
//                        new AndroidEmulator(driverNative)
//                                .unlockDevice()
//                                .closeChrome();
driverNative.closeApp();
driverNative.quit();
driverNative = null;
                }

                driver = Driver.createDriver(addressAppium, adbPort,
udid);

                if (bAppiumService&& driver == null) {
serviceAppium.stop();
serviceAppium.start();
System.out.println("Appium server restart");
                }
        }

switchContext(Constant.Context.FLUTTER);
}

@AfterMethod
public void getResult(ITestResult result) {

        if (!AndroidEmulator.isEmulatorRunning()) return;
        if (driver == null) return;

        String contextName = driver.getContext();
switchContext(Constant.Context.NATIVE);
        if (result.getStatus() == ITestResult.FAILURE
                || result.getStatus() == ITestResult.SKIP
        ) {
getScreen();
        }
}

```

```

        if (contextName.equals(Constant.Context.CHROME)) {
driver.terminateApp("com.android.chrome"); // Закретьбраузер
        }

driver.closeApp();
driver.quit();
        driver = null;
    }

    @Attachment(value = "Screenshot", type = "image/png")
    public byte[] getScreen() {
        return driver.getScreenshotAs(OutputType.BYTES);
    }

    public static void switchContext(String context) {
        int repeatNumber = 3;
        for (int i = 0; i<repeatNumber; i++) {
            Set<String> con = driver.getContextHandles();
            for (String c : con) {
                if (c.contains(context)) {
driver.context(c);
return;
                }
            }
            // Контекста еще нет, возможно стоит его подождать
repeatNumber раз
waitFor(5000);
        }
        throw new RuntimeException("Context not found");
    }

    protected void setTestName(String nameTest) {
AllureLifecycle lifecycle = Allure.getLifecycle();
lifecycle.updateTestCase(testResult ->testResult.setName(nameTest));
System.out.println(nameTest);
    }

    protected static void waitFor(long timeOut) {
        try {
Thread.sleep(timeOut);
        } catch (InterruptedException e) {
e.printStackTrace();
        }
    }
}
}

```


Система автоматизированного тестирования мобильного приложения педагога

Автор работы:

студентка гр. 19ВВВ1 Мереняшева Елизавета Андреевна

научный руководитель:

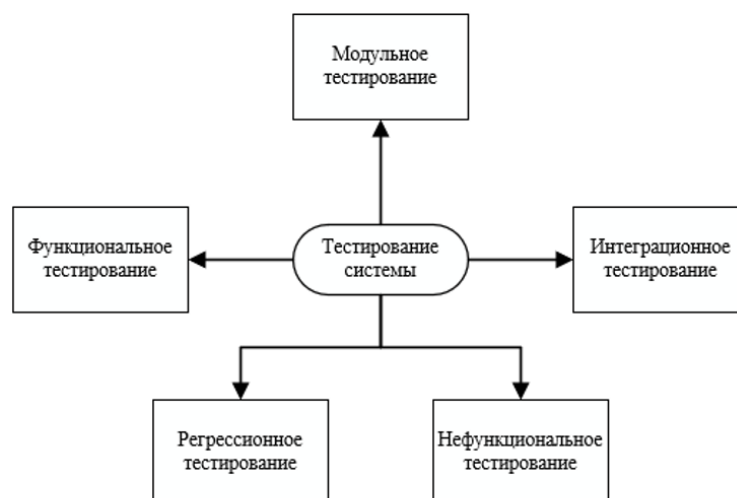
к.т.н., доцент Синев Михаил Петрович

Пенза, 2023

Цель и объект работы

- Цель работы: разработать программную систему для реализации функционального тестирования мобильного приложения педагога.
- Объект работы: система автоматизированного тестирования, обеспечивающая следующие возможности:
 - возможность реализовать функциональное тестирование мобильного приложения педагога;
 - возможность проведения приемочного тестирования мобильного приложения с помощью автоматизированных тестов;
 - возможность генерации отчета о прохождении тестов.

Виды тестирования



Автоматизированное тестирование

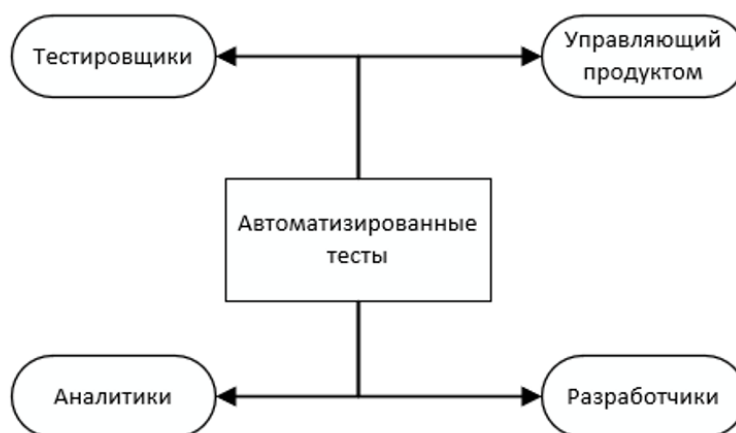
Преимущества:

- увеличение покрытия приложения тестами;
- повышенная точность тестирования;
- экономия затрат;
- помощь ручным тестировщикам.

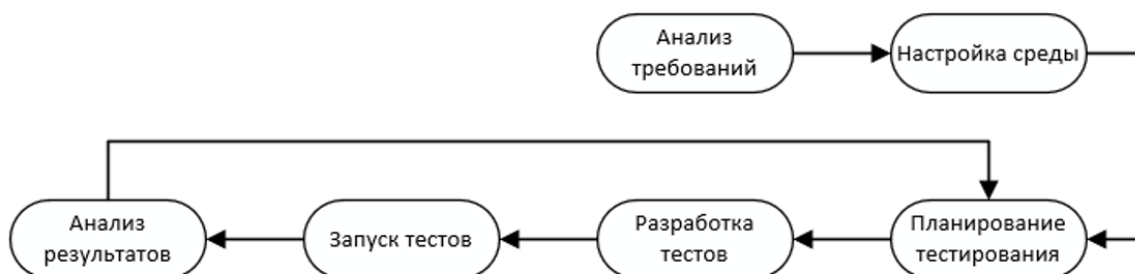
Когда автоматизированное тестирование не подходит:

- приложение имеет короткий срок службы;
- приложение имеет низкий уровень сложности;
- приложение имеет высокую степень непредсказуемости;
- приложение имеет высокую степень человеческого взаимодействия;
- приложение находится на ранних стадиях разработки.

Использование автоматизированных тестов



Этапы тестирования



Стек разработки

TestNG

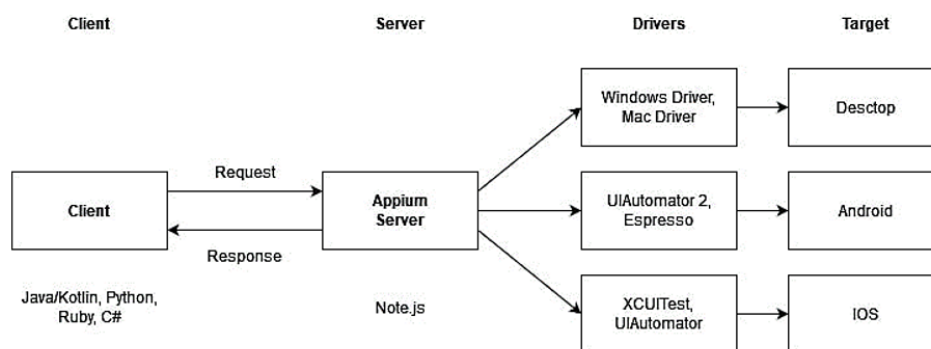
appium™



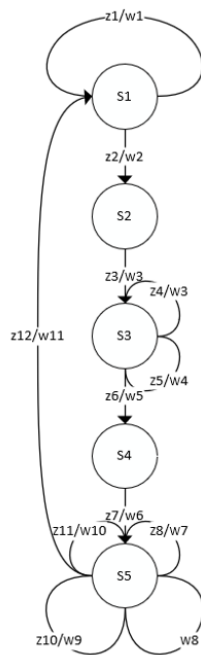
Allure

Appium

Кроссплатформенный фреймворк с открытым кодом для автоматизации тест-кейсов мобильных приложений — нативных, гибридных и веб-приложений.



Абстрактный автомат сценария проверки вариантов сортировки



$S1$ – начальная вкладка при открытии приложения;
 $S2$ – начальная страница на вкладке домашнего задания;
 $S3$ – страница проверки домашнего задания на вкладке домашнего задания;
 $S4$ – страница просмотра домашнего задания/Общие сведения;
 $S5$ – страница просмотра домашнего задания/Результаты выполнения.

$z1...z12$ – входные сигналы
 $w1...w11$ – реакции системы

СКУ и СВФ

$$\begin{aligned} S_1(t+1) &= z_1 S_1 \vee z_{12} S_5; \\ S_2(t+1) &= z_2 S_1; \\ S_3(t+1) &= z_3 S_2 \vee z_4 S_3 \vee z_5 S_3; \\ S_4(t+1) &= z_6 S_3; \\ S_5(t+1) &= z_7 S_4 \vee z_8 S_5 \vee z_9 S_5 \vee z_{10} S_5 \vee z_{11} S_5. \end{aligned}$$

$$\begin{aligned} w_1(t) &= z_1 S_1; \\ w_2(t) &= z_2 S_1; \\ w_3(t) &= z_3 S_2 \vee z_4 S_3; \\ w_4(t) &= z_5 S_3; \\ w_5(t) &= z_6 S_3; \\ w_6(t) &= z_7 S_4; \\ w_7(t) &= z_8 S_5; \\ w_8(t) &= z_9 S_5; \\ w_9(t) &= z_{10} S_5; \\ w_{10}(t) &= z_{11} S_5; \\ w_{11}(t) &= z_{12} S_5. \end{aligned}$$

Результаты тестирования

Allure

Overview

Categories

Suites

Graphs

Timeline

Behaviors

Package

Collapsible

Behaviors

Filter by status

Тесты на проверку раздела «Задания»

#4 Проверка блокировки создания домашнего задания null, 5037, 127.0.0.1, 4723, 1 30s 481ms

#1 Проверка вкладки «Проверить ДЗ» null, 5037, 127.0.0.1, 4723, 1 56s 712ms

#2 Проверка просмотра ДЗ на вкладке «Проверить ДЗ» null, 5037, 127.0.0.1, 4723, 1 50s 057ms

#3 Проверка сортировки списка результатов выполнения ДЗ на странице «Результаты» во вкладке «Задания» null, 5037, 127.0.0.1, 4723, 1 1m 05s

TestHomeWork.testBlockingCreationHomework

Проверка блокировки создания домашнего задания

Overview History Retries

An unknown server-side error occurred while processing the command. Original error: Cannot execute command waitForAppable, server response { "isError": true, "response": "Timeout while executing waitForAppable: TimeoutException after 0:00:15.000000: Future not completed", "type": "extensionType", "method": "ext.flutter.driver" }

Build info: version: '3.141.59', revision: 'af2be7d358', time: '2018-11-14T08:17:03'

System info: host: 'LPTSP-R050517', ip: '10.43.6.99', os-name: 'Windows 10', os-arch: 'amd64', os-version: '10.0', java.version: '10.0.2'

Driver info: io.appium.java.client.android.AndroidDriver

Capabilities {fullReset: false, adbPort: 5037, address: 127.0.0.1, allowCors: false, allowInsecure: [chromedriver_autodownload], appActivity: ru.itco.teacher.MainActivity, appPackage: ru.itco.teacher, automationName: Flutter, basePath: null, callbackPort: 4723, debugOptions: false, deviceName: [], deviceName: (899864), extras: [], fasttest: false, fullReset: false, javascriptEnabled: true, keepAliveTimeout: 0, localInsecure: false, logFile: C:\Users\iteco\Documents\test..., logLevelColors: false, logLevel: debug, longStackTrace: false, newCommandTimeout: 240, noReset: false, noReset: true, platform: ANDROID, platformName: Android, platformVersion: 11, port: 4723, relaxSecurityEnabled: false, reportFormat: html, reportFormat: false, skipInstall: true, skipInstall: true, skipLock: true, strictCaps: false, subcommand: server, testName: Flutter Automation, type: C:\Users\iteco\AppData\Local\... , useDrivers: [uiautomator2, flutter], usePlugins: []}

Session ID: c50830aa-falc-47a8-9618-389132a3cc7

Categories: Test defects

Severity: normal

Duration: 0 30s 481ms

Owner: Меренцева Елизавета Андреевна

Parameters: UICID: null, adbPort: 5037, appiumAddress: 127.0.0.1, appiumPort: 4723, appiumService: 1

Execution: > Set up, > Test body

> Нажатие на УИД, если появилось сообщение об ошибке системы 1 sub-step 5s 474ms

> Проверка элементов на странице «Расписание» 4 sub-steps 73ms

> Нажатие на кнопку «Журналы» 1 sub-step 212ms

Спасибо за внимание!