данным, Правила настроек брандмауэра политики безопасности проекта суперкомпьютерной сети управления процессами социально-экономического развития. Удалось решить комплекс задач, обеспечивающих реализацию цели данной работы. Создание суперкомпьютерной сети управления процессами социально-экономического развития позволит руководству эффективно управлять, защищать и масштабировать ресурсы.

СПИСОК ЛИТЕРАТУРЫ

1.Бондарчук, В. В. Мультисервисная суперкомпьютерная сеть управления процессами социально-экономического развития [Текст] / Отв. ред. к.э.н. Герман Юрьевич Гуляев // EUROPEAN RESEARCH: сборник статей XXVI Международной научно-практической конференции – Пенза: МЦНС «Наука и Просвещение», 2020. – 218 с. – С. 39–43

2. Бондарчук, В. В. Профилактические мероприятия для обеспечения сохранности данных в суперкомпьютерной сети [Текст] / Отв. ред. к.э.н. Герман Юрьевич Гуляев // EUROPEAN SCIENTIFIC CONFERENCE. Сборник статей XX Международной научно-практической конференции – Пенза: МЦНС «Наука и Просвещение», 2020. – 386 с. – С. 82–87

СРАВНИТЕЛЬНЫЙ АНАЛИЗ СРЕДСТВ ТЕСТИРОВАНИЯ МОБИЛЬНЫХ ПРИЛОЖЕНИЙ

Воробьев Н. А.,
студент КемГУ, каф. ЮНЕСКО по ИВТ
Бурмин Л.Н.,
канд.техн.наук, доцент каф. ЮНЕСКО по ИВТ, КемГУ
Степанов Ю.А.,
д-р.техн.наук, профессор каф. ЮНЕСКО по ИВТ, КемГУ

Аннотация. Рассмотрены популярные фреймворки и библиотеки для тестирования мобильных приложений (React-Native, Android). Выявлены позитивные и негативные стороны инструментов. Проведен сравнительный анализ рассмотренных фреймворков по выбранным характеристикам.

Ключевые слова: Тестирование, фреймворки, мобильные приложения, Android, кроссплатформенная разработка

Введение

В связи с большой популяризацией мобильных приложений мы все чаще и чаще начинаем пользоваться мобильными устройствами. Такси, погода, новости, заказ еды и прочее, практически для всего уже существуют свои мобильные сервисы. А раз количество приложений растет, то растет и потребность в качестве выпускаемых приложений. Все больше компаний разработчиков более осознано подходят к тестированию своих мобильных приложений. Разработчики осознают что, если приложение имеет большую аудиторию, то цена дефекта растет, а значит необходимо использовать современные технологии для эффективного нахождения дефектов. Тесты - это тоже код, который требует подержки. Более того, код тестов должен быть прост для понимания, чтобы его можно было верифицировать визуально.В связи с этим, целесообразно инвестирование в упрощение кода тестов, избавление от дублирования и повышение читабельности. Рассмотрим наиболее популярные библиотеки, которые используются в современных IT-компаниях.

Проблема повышения качества продукта на данный момент имеет острый характер, так как ручное тестирование занимает много времени, а использование различных инструментов поможет упростить данную проблему. Рассмотрим перечень широко используемых фреймворков (программных каркасов). Для автоматизации тестирования существует две основные группы: нативные (которые были разработаны для использования на определённой платформе) и кроссплатформенные. Для первого случая мы возьмем платформу Android, а для второго рассмотрим React-Native.

1.Тестирование приложений под Android

1.1.**Unit-тестирование.** Существует два популярных фреймворка для модульного тестирования.

1.1.1.JUnit. Сам фреймворк состоит из нескольких проектов: Jupiter, Vintage, Platform. В данное статье будет рассмотрен Junit Jupiter, который является основным проектом JUnit. Он позволяет создавать тесты и свои расширения. В проекте есть свой TestEngine, который запускает тесты на JUnit платформе. Инструмент позволяет запускать тесты автоматически при каждой сборке проекта. Тем самым можно повысить качество продукта и выпускаемого функционала. Порог вхождения минимален, никаких сторонних зависимостей ставить не требуется, JUnit встраивается непосредственно в проект. В официальной документации можно найти примеры использования с лучшими практиками. Довольно просто интегрируется с другими библиотеками и фреймворками для тестирования, такими как Mockito или Espresso. У данного инструмента есть некоторые недостатки, например, нельзя тестировать зависимости и он не подходит для тестирования больших наборов тестов. К недостаткам можно отнести и малое число аннотаций, однако для небольшого проекта их хватает.

1.1.2.Robolectric. Фреймворк разработан компанией PivotalLabs в 2010 году. Он занимает промежуточное положение между чистым Junit-тестами и инструментированными тестами, запускаемыми на устройстве, симулируя реальное Android окружение. Порог вхождения не такой низкий как у Junit и требует более тщательной работой с документацией. Стоит отметить, что инструмент имеет поддержку AndroidX. Это очень актуально, так как новые проекты автоматически создаются с поддержкой данного инструмента. В среднем частота обновления мажорной версии составляет один раз в месяц. Инструмент легко настраивается и может работать с другими библиотеками. К недостаткам можно отнести тот факт, что он не охватывает все функции с датчиков реального устройства. Для запуска тестов нужен эмулятор или реальное устройство.

1.1.3.Mockito. Фреймворк для работы с моками или “заглушками”, которые используются вместо реальных объектов для использования их в тестах. Mockito позволяет создавать жесткие (неизменяемые) моки, чтобы их подведение было предсказуемым, насколько это возможно. Можно обойтись и без использования фреймворка. В этому случае нужно будет создавать альтернативные тестовые реализации интерфейсов, наследовать нужные классы с переопределением функционала и так далее, что потребует дополнительного времени на сложную аналитическую работу для описания всех возможных вариантов. Более удобное в этом случае решение - специализированные фреймворки для создания заглушек. Одним из таковых самых распространенных в среде Java является Mockito. Имеется официальный сайт, на котором в подробно расписано, как встроить каркас в программный код. Заглушки можно описывать буквально одной аннотацией, что очень сильно экономит время на разработку тестов..

1.2.UI - тестирование.

1.2.1.Espresso. Данная библиотека была разработана Google в 2013 году. В 2014 это проект получил статус Open Source. Espresso не может самостоятельно работать с другими приложениями и системой Android через API, однако использует Recorder, с помощью которого можно записывать простые сценарии и использовать их на начальном этапе автоматизации. Если нужно тестировать только приложение, без учёта его взаимодействия с системой или другими программами, целесообразнее лучше использовать Espresso. К тому же в нём реализованы удобные функции вроде автоматической синхронизации тестов с UI приложения, позволяющие не писать различные команды вызова ожиданий по таймауту. Если же вам нужно протестировать приложение в связке с другим программным обеспечением или с функциями операционной системы, а доступ есть только к apk-файул, то лучше использова UIAutomator. Документации достаточно, чтобы создавать UI тесты на базовом уровне.

1.2.2.Ui automator. Многообразный фреймворк для тестирования UI, поставляемая с Android SDK. Включает в себя две утилиты:

UIAutomatorviewer — графический инструмент для распознавания компонентов пользовательского интерфейса в Android приложении. Делает снимок экрана устройства, который подключен к компьютеру, а также предоставляет графический интерфейс для отображения иерархии слоев и просмотра свойств каждого компонента интерфейса в отдельности. Наличие этой информации значительно упрощает процесс создания UIAutomator-скрипта.

UIAutomator использует библиотеки Java API, содержащие методы для создания тестов пользовательского интерфейса. Включает API, который позволяет конкретный элемент. Для этого мы создаем идентификатор, по которому нам нужно найти элемент и в последующем работаем уже с сами идентификатором.

1.2.3.Appium. Кроссплатформенный инструмент, который позволяет писать тесты для мобильных платформ (iOS, Android, Windows), используя API. Это один из широко используемых инструментов для системного тестирования приложений на смартфонах и планшетах. На этот раз вызовы WebDriver API преобразовываются в вызовы методов фреймворка от Google — UI Automator. Приложение можно тестировать и на симуляторе, и на реальном устройстве. Есть некоторые отличия между работой с операционными системами. К примеру, Android, в отличие от iOS, не ограничивает разработчика тестов одним приложением в рамках одной сессии, есть возможность входить и выходить из любых приложений неограниченное количество раз за сессию.

2.Тестирование React-Native приложений

Если говорить про тестирование приложений под React-Native, мы можем использовать инструменты для тестирования web-приложений. К одному из таких инструментов можно отнести **Jest.** Он был создан компанией Facebook и представляет собой фреймворк для тестирования кода JavaScript и React-приложений. Тесты простых компонентов, которые занимаются только отображением данных, можно проверять с помощью снимков (snapshots), после запуска теста появляется снимок, по которому можно проверить, соответствует ли фактический результат – ожидаемому. У Jest есть обширная система имитаций событий: click, focus, submit и т.д. Так же инструмент может имитировать различные функции и файлы. Если нужно сымитировать файлы по-своему, то просто кладем его в каталог «mocks». Инструмент обладает подробной документацией, однако порог вхождения не такой низкий, как у нативных фреймворков. Каркас требуется конфигурирования для конкретного проекта, например, нужно указать путь до тестов и местоположения результатов

3.Сравнительный анализ

Для проведения сравнительного анализа описаных выше популярных фреймворков для тестирования, выделим несколько критериев:

1. Время развертывания - так как некоторые инструменты используют другие зависимости, данное время может расти, что может увеличить время разработки.

2. Время внедрения - это очень важный критерий, т.к. с момента чтения документации, и готовым тестом проходит немало времени.

3. Порог вхождения - сколько нужно изучить документации, чтобы написать тест, который сможет решить задачу.

4. Документация - критерий, насколько полная документация по данному инструменту.

Результаты проведенного анализа представлены в таблице 1.

Таблица 1

Сравнения программных каркасов для тестирования.

	JUnit	Mockito	Espresso	Ui automator	Jest	Appium
Время развертывания, мин	<5	7-15	44109	<5	10-20	15-30
Время внедрения, мин	5-10	30-40	10-20	20-40	>30	20-40
Порог вхождения,от 1 до 5	2	2	1	2	2	2
Документация, от 1 до 5	4	4	3	5	4	3

Заключение.

Современные фреймворки для тестирования довольно хорошо справляются с задачами, которые определены надежностью функционирования приложений. Разработчики данных инструментов делают все возможное, чтобы разработчикам было легче покрывать свой код тестами. А многие фреймворки еще и могут использоваться одновременно, взаимно дополняя друг друга, например: Jest + Mockito. Применение этих инструментов позволило протестировать неочевидные проблемы, снизить дублирование кода и в целом повысить читабельность тестов.

Список литературы.

1. Jest: сайт. – URL: https://jestjs.io/ (дата обращения: 22.10.2019). – Текст: электронный.

2. JUnit: сайт. – URL: https://github.com/junit-team/junit5 (дата обращения: 15.10.2019). – Текст: электронный.

3. Mockito: сайт. – URL: https://github.com/mockito/mockito (дата обращения: 17.10.2019). – Текст: электронный.

4. Автоматизация тестирования Android приложений с помощью UIAutomator: сайт. – URL: https://habr.com/ru/company/intel/blog/205864/ (дата обращения: 20.10.2019). – Текст: электронный.

5. Модульное тестирование React-приложения с помощью Jest и Enzyme: сайт. – URL: https://medium.com/devschacht/berry-de-witte-unit-testing-your-react-application-with-jest-and-enzyme-6ef3658fdc93 (дата обращения: 22.10.2019). – Текст: электронный.

РАЗРАБОТКА ТЕХНОЛОГИИ ПОЛУЧЕНИЯ ПЛАТИНОВЫХ МЕТАЛЛОВ ИЗ ТЕХНОГЕННЫХ ОТХОДОВ

Вохидов Б. Р.
доцент кафедры «Металлургия»
Навоийского государственного горного института.
г. Навои, Узбекистан.
DOI: 10.31618/ESU.2413-9335.2020.1.75.822

АННОТАЦИЯ

В данной статье рассматривается возможность селективного выделения платины и палладия из техногенного отхода медного электролитного шлама после извлечения золота и серебра. В работе определено эффективность методов селективного осаждения платины и палладия, а также уделено внимание способам растворения, восстановления платиновых металлов и методы их очистки из различных примесив. На основе изучения данной тематики и анализа результатов проведенных исследований авторы пришли выводу, что в качестве оптимального реагента для осаждения палладия и других МПГ из отработанных растворов целесообразно выбрать тиомачевинный раствор. Также было выявлено, что в процессе восстановления палладия наиболее эффективным восстановителем является раствор гидразина. Установлено, что растворимости металлического палладия и платины царско-водочного растворение, как