МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

__________ Л.Б. Соколинский

«___»___________ 2024 г.

Разработка системы автоматизированного тестирования многопользовательского продукта СИМИ с использованием JUnit

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-342.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
__________ Т.Ю. Маковецкая

Автор работы,
студент группы КЭ-403
__________ И.М. Колодкин

Ученый секретарь
(нормоконтролер)
_____________ И.Д. Володченко
«___»___________ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
__________ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-403
Колодкину Ивану Михайловичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)
Разработка системы автоматизированного тестирования многопользовательского продукта СИМИ с использованием JUnit.
2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**
3.1. Медицинский портал ЕМИАС.ИНФО. [Электронный ресурс] URL: https://emias.info/ (дата обращения: 01.02.2024 г.).
3.2. JUnit 5. [Электронный ресурс] URL: https://junit.org/junit5/ (дата обращения: 02.02.2024 г.).
3.3. Java Software. [Электронный ресурс] URL: https://www.java.com/ru/ (дата обращения: 05.02.2024 г.).
3.4. Functional testing. [Электронный ресурс] URL: https://www.professionalqa.com/functional-testing (дата обращения: 06.02.2024 г.).
3.5. Guide to Functional testing. [Электронный ресурс] URL: https://www.softwaretestinghelp.com/guide-to-functional-testing/ (дата обращения: 07.02.2024 г.).
4. **Перечень подлежащих разработке вопросов**
4.1. Выполнение анализа предметной области.
4.2. Проектирование системы и наборов тестов.

4.3. Реализация системы и наборов тестов.
4.4. Тестирование системы.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. Т.Ю. Маковецкая

Задание принял к исполнению И.М. Колодкин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ 5
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 7
    1.1. Система ЕМИАС 7
    1.2. Подсистема СИМИ 8
2. ПРОЕКТИРОВАНИЕ 12
    2.1. Проектирование наборов тестов 12
    2.2. Определение требований к системе 14
    2.3. Проектирование системы 16
3. РЕАЛИЗАЦИЯ 20
    3.1. Выбор инструментов реализации 20
    3.2. Реализация системы 21
    3.3. Реализация тестов 28
4. ТЕСТИРОВАНИЕ 33
ЗАКЛЮЧЕНИЕ 34
ЛИТЕРАТУРА 35

ВВЕДЕНИЕ

Актуальность

Развитие медицины играет ключевую роль в улучшении качества жизни и увеличении продолжительности жизни населения. Современные медицинские достижения позволяют эффективно бороться с ранее неизлечимыми заболеваниями, проводить сложные хирургические операции и обеспечивать высококачественную реабилитацию пациентов. В условиях стремительного научно-технического прогресса, медицинские учреждения сталкиваются с необходимостью интеграции новых технологий для повышения эффективности своей работы и улучшения качества предоставляемых услуг.

Одним из важнейших аспектов модернизации здравоохранения является внедрение информационных систем в медицинских учреждениях. Эти системы позволяют автоматизировать процессы управления, ускорять обмен информацией, улучшать координацию между различными отделениями и обеспечивать доступ к актуальным данным о пациентах. Внедрение такой системы способствует снижению вероятности ошибок, повышению оперативности медицинской помощи и улучшению общей организационной структуры учреждения.

Важным компонентом любой медицинской информационной системы является подсистема для работы с документами. Данная подсистема обеспечивает хранение, управление и доступ к медицинской документации, включая истории болезней, результаты анализов и диагностические заключения. Эффективное управление документами является критически важным для обеспечения непрерывного и качественного медицинского обслуживания. Оно позволяет медицинскому персоналу быстро получать необходимую информацию, что значительно улучшает принятие решений и качество лечения.

Поддержка таких масштабных продуктов обязательно требует выпуск новых обновлений, так как в процессе использования таких систем обязательно находятся новые ошибки и недочеты, добавляются новые функции. Ввиду такого частого обновления системы было бы удобно использовать систему автоматизированного тестирования, нежели тестировать каждое обновление вручную. Автоматизированное тестирование позволяет значительно ускорить процесс проверки, повысить его точность и обеспечить непрерывность тестирования при каждом обновлении системы.

Постановка задачи

Целью выпускной квалификационной работы является разработка системы автоматизированного тестирования многопользовательского продукта СИМИ с использованием JUnit.

Для достижения поставленной цели необходимо решить следующие задачи:

1) выполнить анализ предметной области;
2) спроектировать систему и наборы тестов;
3) разработать систему и наборы тестов;
4) протестировать систему.

Структура и содержание работы

Работа состоит из введения, четырех глав, заключения и списка литературы. Объем работы составляет 36 страниц, объем списка литературы – 15 источников.

В первой главе проводится анализ предметной области.

Вторая глава посвящена проектированию системы и наборов тестов на основе функциональных и нефункциональных требований.

В третьей главе описываются программные средства реализации системы, а также приводятся детали реализации основных компонентов системы.

Четвертая глава посвящена функциональному тестированию реализованной системы.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В Москве и Московской области основной медицинской системой является ЕМИАС [2] – единая медицинская информационно-аналитическая система. Основным компонентом, который реализует работу с документами, является СИМИ – система интегрированной медицинской информации [3]. Далее приведено более подробное описание этих систем.

1.1. Система ЕМИАС

Единая медицинская информационно-аналитическая система (ЕМИАС) города Москвы разработана для повышения качества и доступности медицинской помощи в государственных учреждениях здравоохранения.

Проект разработан и реализуется Департаментом информационных технологий города Москвы совместно с Департаментом здравоохранения города Москвы в рамках программы «Информационный город» [4]. К 2015 году к ЕМИАС подключены 660 государственных учреждений здравоохранения: городские поликлиники (детские и взрослые), женские консультации и стоматологии. В системе работает более 23 тысяч медицинских работников. Кроме того, ведутся работы по организации информационного взаимодействия между амбулаторно-поликлиническим звеном и стационарами с использованием ЕМИАС. ЕМИАС является региональным сегментом Единой государственной информационной системы в здравоохранении.

Внедрение ЕМИАС реализуется в несколько этапов, первым из которых является переход на «электронную регистратуру», позволяющую удаленно записываться на прием к врачу, перенести прием и запись без предварительной отмены, найти ближайшую поликлинику по месту проживания и многое другое. Вторым этапом внедрения является разработка сервисов для сбора и систематизации истории болезней граждан, а также для выписывания электронных рецептов.

Система позволяет:

1) управлять потоками пациентов (СУПП);

2) выписывать электронные рецепты;

3) вести консолидированный управленческий учет, а также персонифицированный учет медицинской помощи.

При этом система содержит интегрированную амбулаторную медицинскую карту.

Помимо этого, система содержит информацию о загруженности медицинских учреждений и востребованности врачей и позволяет управлять медицинскими регистрами, решая медико-организационные задачи применительно к различным категориям граждан, имеющих определенные заболевания.

1.2. Подсистема СИМИ

Согласно технической документации, система интегрированной медицинской информации Единой медицинской информационно-аналитической системы города Москвы (СИМИ ЕМИАС) представляет собой один из общегородских информационных сервисов в составе ЕМИАС, предназначенный для обеспечения автоматизации следующих процессов.

1. Сбора, хранения (в том числе архивного), обработки и консолидации медицинской информации о пациенте в составе интегрированной электронной медицинской карты (иЭМК) пациента – в едином хранилище на основе стандартизированных электронных медицинских документов (СЭМД) по всем случаям обращения гражданина в медицинские учреждения города Москвы.

2. Оперативного авторизованного поиска и извлечения медицинской информации о пациенте из любого государственного медицинского учреждения города Москвы, а также из других регионов РФ через федеральные сервисы (после ввода в эксплуатацию федеральных сервисов и реализации интеграции системы с ними).

3. Накопления в структурированном виде фактографической информации о состоянии здоровья и оказании медицинской помощи.

4. Создания, редактирования, публикации протоколов.

Архитектура системы позволяет постепенное расширение перечня обрабатываемых протоколов без необходимости внесения изменений в основные программные модули системы. В процессе развития системы предполагается возможность увеличения доли медицинской информации, которая хранится в системе в структурированном виде.

Документ СИМИ

Документ СИМИ – стандартизированный электронный медицинский документ в формате XML, с помощью которого вносится информация в иЭМК пациента.

Документ СИМИ состоит из метаданных, контента документа, и, в некоторых случаях, подписи. Контент документа, также именуемый как тело документа, содержит сведения, которые необходимо внести в иЭМК пациента. Контент хранится в формате TDD (структурированный), XML или PDF. Документ за время своего существования несколько раз меняет свой статус, в системе предусмотрено 5 статусов – созданный, черновик, подписанный, архивированный или аннулированный. Документы подписываются с помощью электронной подписи работником медицинской организации, если это необходимо.

API СИМИ

Сервис СИМИ предоставляет 11 API методов для работы с документами. Каждый из этих методов имеет свой функционал и выполняет определенную задачу. Например, метод `createDocument` создает документ, `getDocument` возвращает документ. Список всех API методов СИМИ сервиса приведен ниже:

1) `createDocument` – создает документ;
2) `getDocument` – выполняет операцию получения документа;
3) `archiveDocument` – архивирует документ;

4) deprecateDocument – аннулирует документ;

5) getAuditRecords – возвращает данные аудита действий пользователей;

6) getCareEventDocuments – возвращает документы, относящиеся к указанному клиническому событию;

7) getCareEventDocumentsForSign – возвращает документы для подписания, которые относятся к указанному клиническому событию;

8) getDocumentsByPatientForSign – возвращает документы для подписания по указанному пациенту, относящиеся к данному клиническому событию;

9) getDocumentsForSign – возвращает документы для подписания, автором которых является данный пользователь;

10) saveDocument – сохраняет документ в хранилище;

11) searchDocuments – возвращает метаданные документов, удовлетворяющих условиям поиска.

Для достижения определенной цели, такой как, например, сохранение подписанного документа (рисунок 1), необходимо выполнить последовательно несколько методов:

1) createDocument для создания документа;

2) saveDocument для сохранения документа в статусе черновика;

3) getDocument для получения содержимого документа;

4) saveDocument с измененным запросом для сохранения и подписания документа.

В данном случае на рисунке 1 приведен один из самых распространенных сценариев использования СИМИ сервиса. Такие сценарии могут состоять из одного действия или 10 действий, приводя к различным результатам.

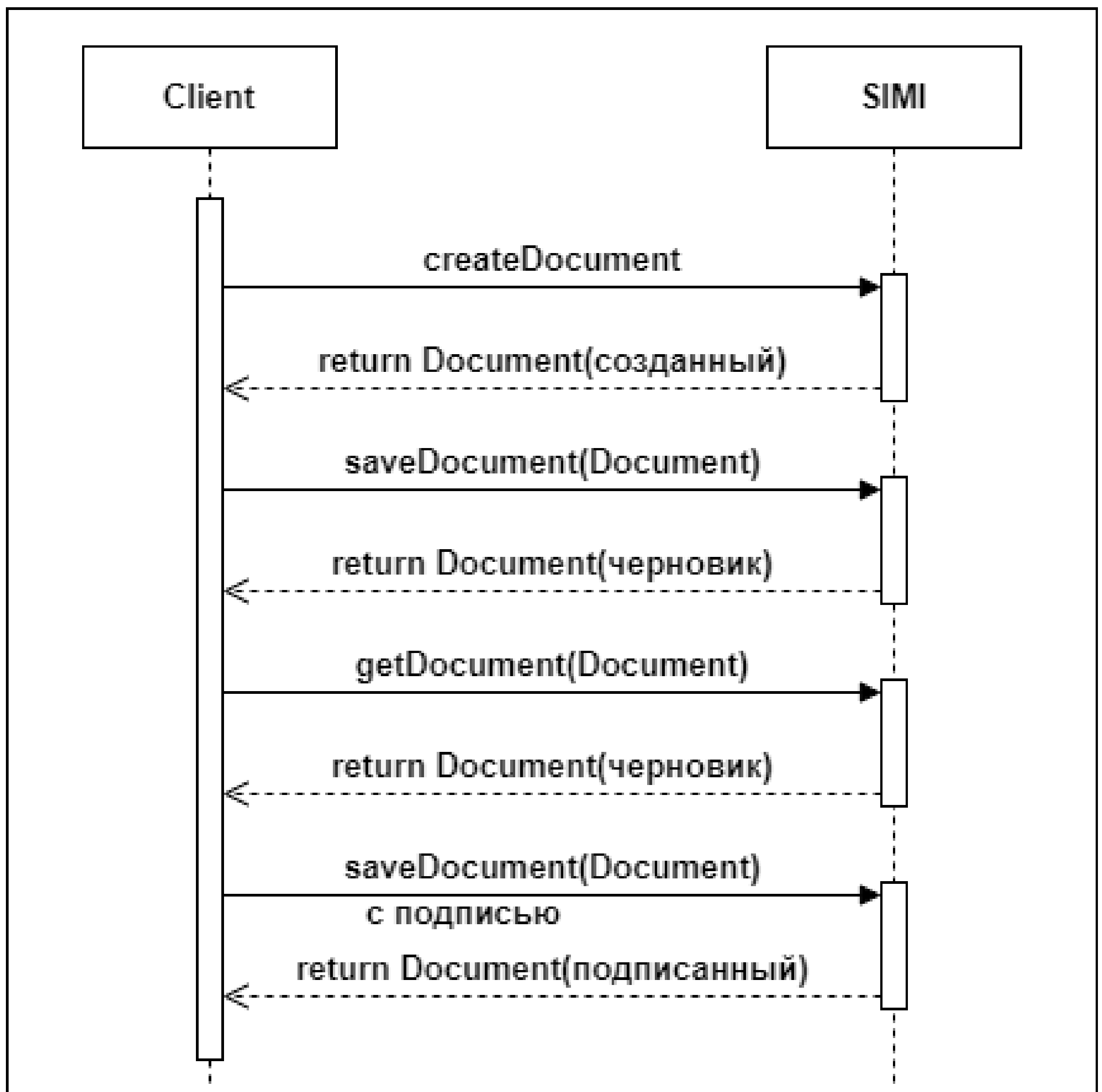

Рисунок 1 – Процесс сохранения подписанного документа

Вывод по первой главе

СИМИ обладает широким спектром возможностей для взаимодействия с документами, поэтому при проектировании тестов необходимо сделать акцент на тестировании разных групп функциональности. Также необходимо учесть, что сервис постоянно обновляется и в будущем возможно добавление нового функционала, поэтому систему необходимо спроектировать так, чтобы потом в нее можно было беспрепятственно добавлять новые наборы тестов.

2. ПРОЕКТИРОВАНИЕ

2.1. Проектирование наборов тестов

При проектировании тестов важно учитывать, что каждый набор тестов должен отвечать за тестирование определенной группы функциональности. Исходя из этого, а также учитывая данные из главы 1, были определены следующие наборы тестов:

1) BaseTests;
2) OptionsTests;
3) DifferentTests.

Набор BaseTests

В первом наборе тестов будет сделан акцент на тестировании базовых сценариев использования сервиса. Будут использованы следующие операции с документами: создание, сохранение (с подписанием и без), архивирование и аннулирование. С помощью этих операций документ сможет получить следующие статусы: CREATED, DRAFT, SIGNED, ARCHIVED, DEPRECATED. Также в этом наборе тестов будет использоваться два типа контента документа: XML и TDD. Сформированный итоговый набор тестов выглядит следующим образом:

1) удаление черновика xml;
2) удаление черновика tdd;
3) удаление подписанного xml;
4) удаление подписанного tdd;
5) архивирование черновика xml;
6) архивирование черновика tdd;
7) архивирование подписанного xml;
8) архивирование подписанного tdd.

Набор OptionsTests

Задачей второго набора тестов (OptionsTests) является тестирование опций. У каждого метода СИМИ сервиса может быть свой доступный набор

опций. Опция добавляется к запросу и указывает детали выполнения метода СИМИ. Во втором наборе тестов будут задействованы следующие опции:

1) addTag;
2) removeTag;
3) addAssociation;
4) removeAssociation;
5) partiallyFilledDocument;
6) suppressContent;
7) suppressVisualization;
8) includeAllAssocitaions.

Для достижения определенного результата в одном тесте может использоваться несколько опций или одна опция. Итоговый сформированный набор выглядит следующим образом:

1) сохранение документа с ассоциацией;
2) сохранение документа с ассоциацией без типа;
3) сохранение документа с ассоциацией без роли;
4) сохранение документа с ассоциацией без другой роли;
5) сохранение документа с удаленной ассоциацией;
6) добавление и удаление тегов из документа;
7) общий тест на ассоциации;
8) тестирование опции «suppressContent»;
9) тестирование опции «partiallyFilledDocument».

Набор DifferentTests

В третьем наборе будут размещены тесты, которые нельзя отнести к одной группе функциональности. В такой набор будут включены:

1) тест таблиц баз данных;
2) тест на удаление композиции;
3) тест на сохранение документа с превышенным допустимым размером контента документа;
4) тестирование таймаута после сохранения документа.

2.2. Определение требований к системе

Исходя из данных, полученных в первой главе, а также учитывая спроектированные в прошлом пункте наборы тестов, можно сформулировать функциональные требования к системе автоматизированного тестирования. Система должна предоставлять пользователю следующие возможности:

1) запуск одного теста;
2) запуск нескольких тестов из одного набора;
3) запуск одного набора тестов;
4) запуск нескольких наборов тестов;
5) запуск всех наборов тестов.

Также, с учетом некоторых особенностей, которым должна соответствовать система, были выявлены нефункциональные требования:

1) в организации, занимающейся разработкой СИМИ, для написания кода используется язык программирования Java, поэтому система также должна быть реализована на Java;
2) для тестирования в организации используется фреймворк JUnit, поэтому тестирование также должно быть реализовано с помощью JUnit.

Для визуализации функциональных требований был использован унифицированный язык моделирования для объектно-ориентированного проектирования UML [5]. UML является открытым стандартом использующим графические обозначения для создания абстрактной модели системы.

В ходе работы была построена модель взаимодействия внешнего актера с системой автоматизированного тестирования в виде диаграммы вариантов использования. В ходе анализа требований к разрабатываемой системе был определен актер, а также были выявлены варианты использования (рисунок 2).

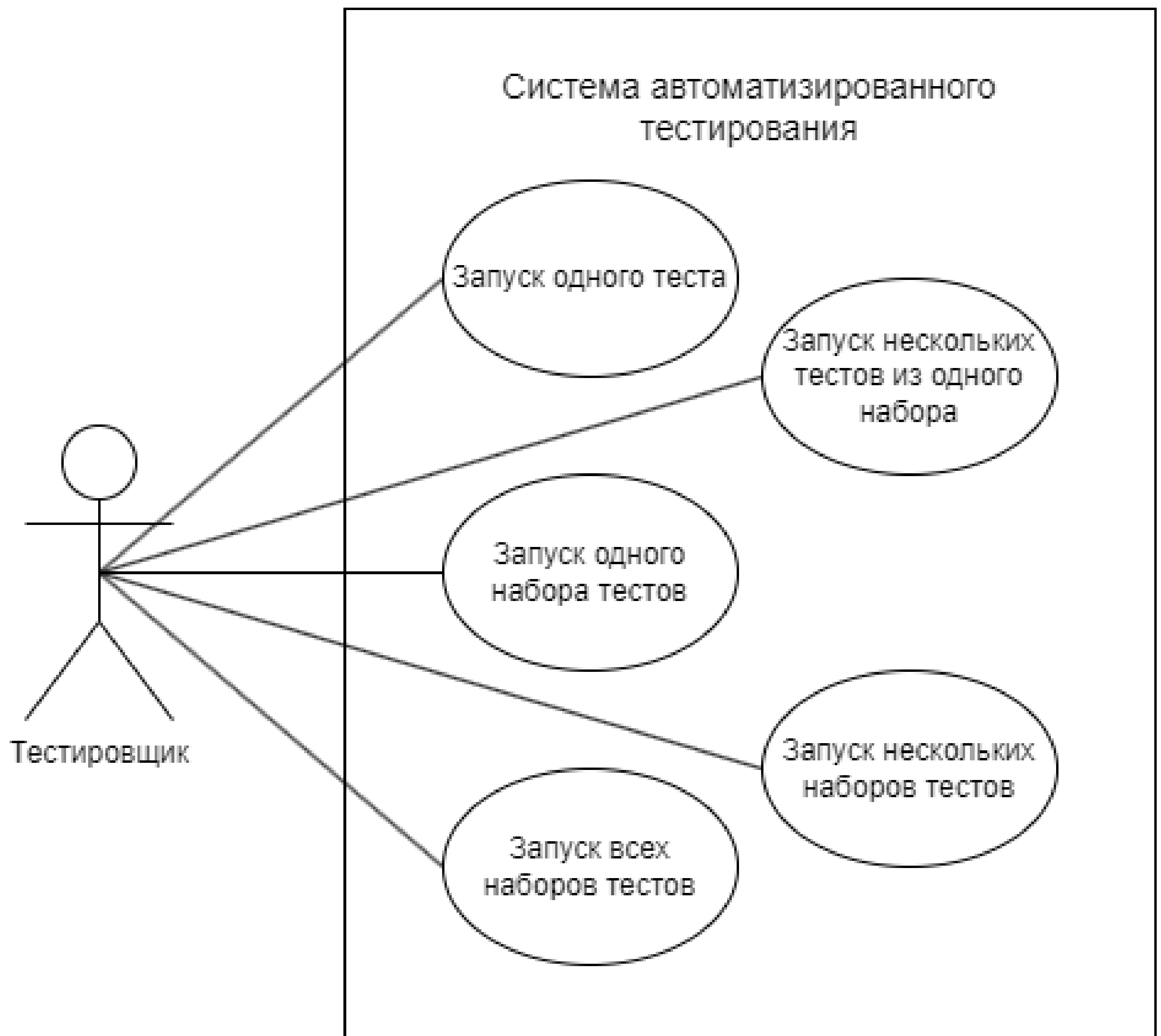

Рисунок 2 – Диаграмма вариантов использования

Для данной системы определен актер – тестировщик. Актеру доступны следующие варианты использования системы:

1) запуск одного теста;
2) запуск нескольких тестов из одного набора;
3) запуск одного набора тестов;
4) запуск нескольких наборов тестов;
5) запуск всех наборов тестов.

2.3. Проектирование системы

При определении нефункциональных требований для реализации системы автоматизированного тестирования было установлено, что необходимо использовать объектно-ориентированный язык программирования Java. В связи с этим, для определения основных классов программы, определения взаимосвязей между этими классами, а также для понимания процесса взаимодействия между ними были спроектированы диаграмма классов и диаграмма взаимодействия компонентов.

Диаграмма классов

При проектировании диаграммы классов было определено 11 основных Java классов:

1. Класс `Runner` – основной компонент системы автоматизированного тестирования. С помощью него тестировщик будет взаимодействовать с программой, выбирать необходимый вариант использования, выбирать нужные наборы тестов и тесты, а после их выполнения получать отчет о проведенном тестировании.

2. Класс `ApplicationManager` – центральный компонент системы, с помощью которого тесты будут взаимодействовать с остальными частями программы.

3. Класс `SimiServiceClient` – компонент, в котором будет инициироваться делегат сервиса СИМИ. Также в этом компоненте к делегату сервиса будет добавляться обработчик WS-Security заголовков.

4. Класс `AddUserNameToken` – компонент, с помощью которого будет реализовываться обработчик WS-Security заголовков. Его будет использовать класс SimiServiceClient.

5. Класс `SimiServiceHelper` – компонент, в котором будут реализованы функции для составления и отправки запросов к сервису СИМИ. С помощью этого класса будет достигнуто удобство использования запросов с разным набором параметров.

6. Класс `DBHelper` – похожий на предыдущий компонент, однако в этом классе работа будет строиться вокруг взаимодействия с базой данных.

7. Класс `TestSettings` – компонент, реализующий взаимодействие с конфигурационными параметрами проекта, такими как адреса сервиса и баз данных.

8. Класс `SignatureSerializer` – вспомогательный компонент, который используется для сериализации подписи из XML формата в Java объект.

9. Классы `BaseTests`, `OptionsTests` и `DifferentTests` содержат в себе наборы тестов, спроектированные в пункте 2.1. Каждый из них взаимодействует с системой с помощью класса ApplicaitonManager.

Спроектированные классы и их связи представлены на рисунке 3.

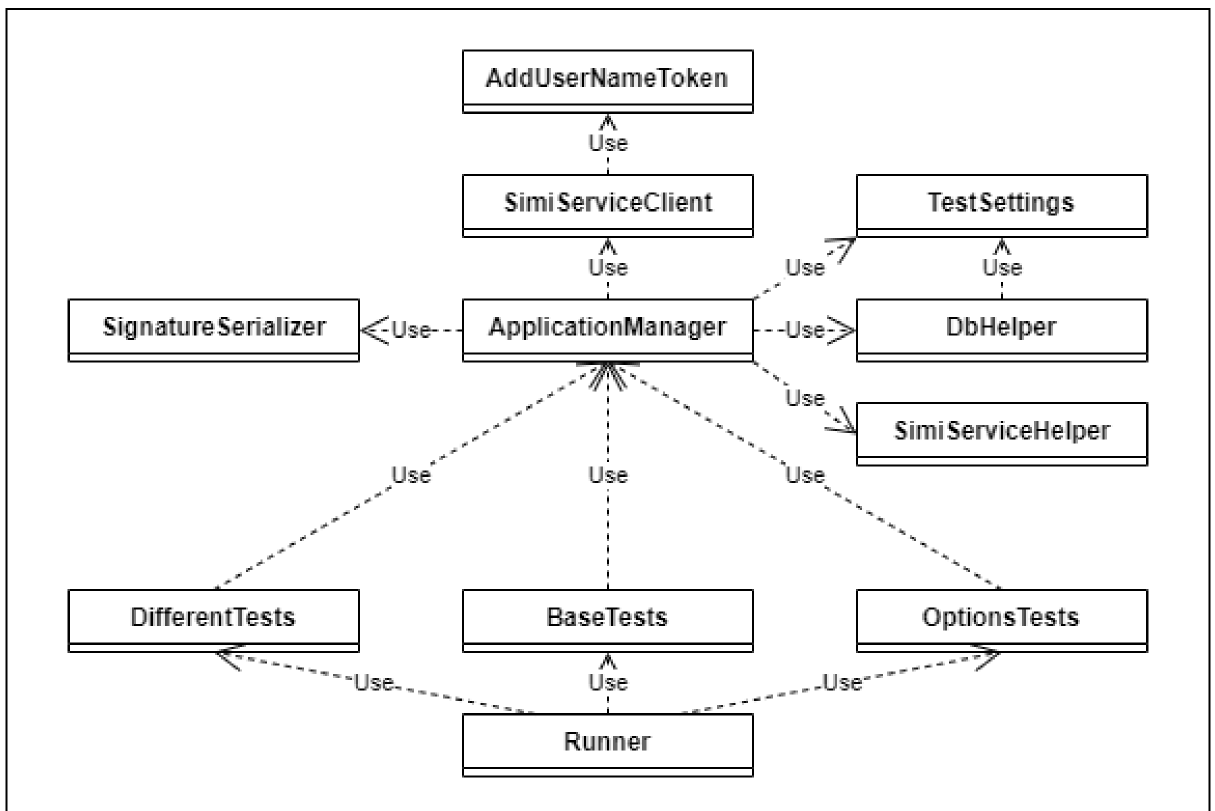

Рисунок 3 – Диаграмма классов

Диаграмма взаимодействия компонентов

Для визуализации и понимания процесса взаимодействия компонентов системы автоматизированного тестирования была спроектирована соответствующая диаграмма (рисунок 4).

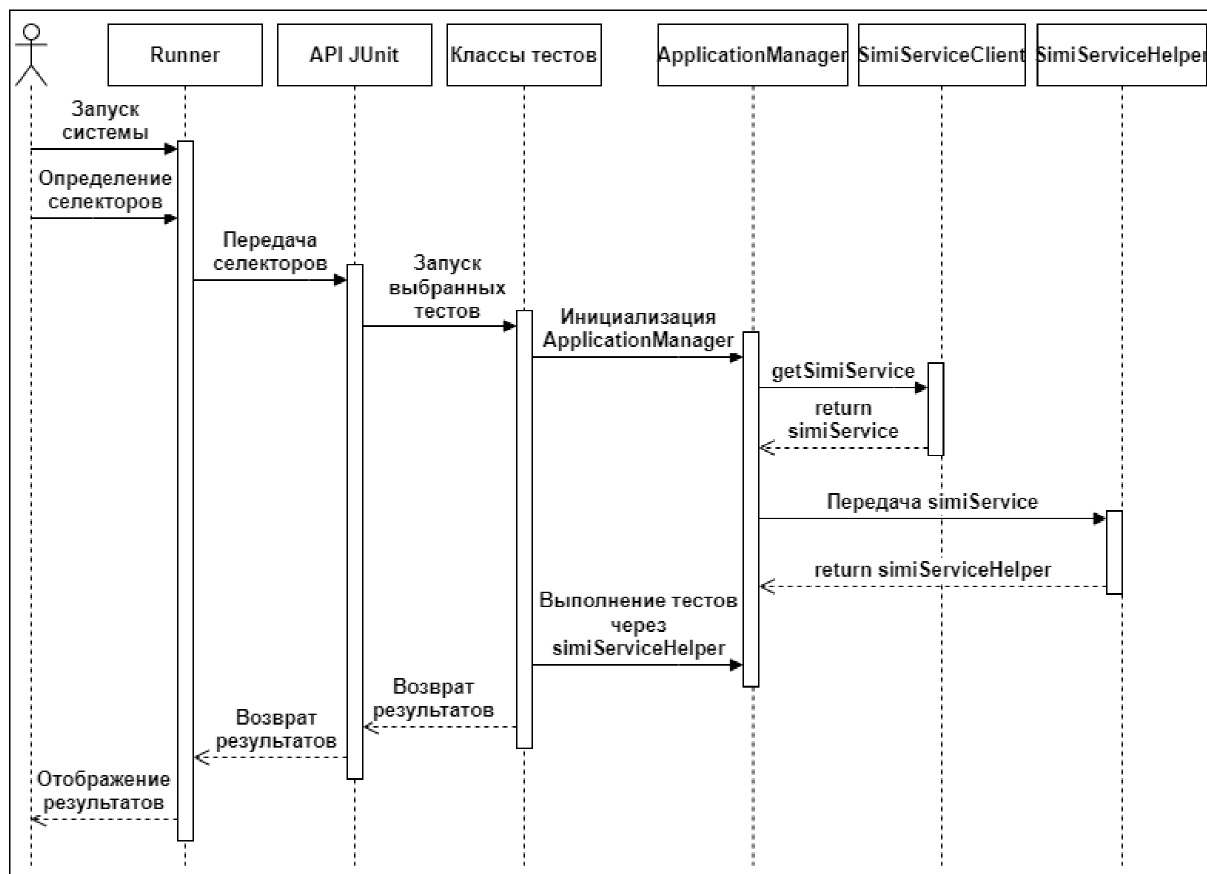

Рисунок 4 – Диаграмма взаимодействия компонентов

Пользователь программы, в данном случае тестировщик, инициализирует выполнение программы с помощью класса Runner. Далее внутри этого класса определяется набор селекторов – список тестов, которые необходимо выполнить. Набор селекторов также определяет тестировщик с помощью консольного интерфейса.

Следующим этапом является передача сформированного списка селекторов в Junit API Launcher. Лаунчер, в свою очередь, инициализирует работу каждого выбранного класса с тестами и после их выполнения возвращает статистику результатов тестирования.

Каждый класс с тестами в первую очередь инициализируют экземпляр класса ApplicationManager, а затем с его помощью взаимодействует со всеми остальными компонентами системы. Тесты могут использовать этот экземпляр класса для создания делегата СИМИ, составления и отправки запросов, добавления подписи или обращения к базам данных.

На диаграмме взаимодействия компонентов (рисунок 4) приведен пример взаимодействия тестов с классом SimiServiceHelper, который реализует функцию составления и отправки запросов к сервису СИМИ. В данном примере в классе `ApplicationManager` происходит обращение к классу `SimiServiceClient`, который возвращает делегата сервиса СИМИ и передает его в `SimiServiceHelper`. С помощью переданного в хелпер делегата классы с тестами буду взаимодействовать с сервисом – отправлять запросы и получать ответы.

После выполнения всех выбранных тестов пользователь программы получает отчет о результатах тестирования в виде текстовой таблицы в консольном интерфейсе программы.

Вывод по второй главе

В данной главе представлено описание проектирования наборов тестов. Общее количество спроектированных наборов тестов равняется 3. Общее количество тестов, включенных в эти наборы – 21.

Также в главе спроектирована система автоматизированного тестирования. В ходе работы были определены актер программы и варианты использования системы. Для визуализации и понимания структуры программы и процесса взаимодействия между компонентами системы были построены диаграмма классов и диаграмма взаимодействия компонентов.

3. РЕАЛИЗАЦИЯ

3.1. Выбор инструментов реализации

При определении нефункциональных требований было выявлено, что для разработки системы автоматизированного тестирования необходимо использовать язык программирования Java. Java – это широко используемый объектно-ориентированный язык программирования, разработанный компанией Sun Microsystems в 1995 году [6]. Java славится своей стабильностью, производительностью и обширной экосистемой библиотек и фреймворков, что делает ее популярным выбором для разработки как серверных приложений, так и клиентских.

Java также отлично подходит для написания автотестов благодаря наличию множества фреймворков, облегчающих создание и выполнение тестов. Среди таких фреймворков – JUnit [1], TestNG [10] и Selenium [11]. JUnit, будучи одним из самых популярных фреймворков для тестирования в Java, предоставляет удобный и мощный инструментарий для написания и организации тестов. Фреймворк поддерживает аннотации для определения тестовых методов, механизмы setup и teardown для подготовки и очистки окружения, а также интеграцию с различными системами сборки и CI/CD.

Для создания проекта выбрана IDE IntelliJ IDEA. IntelliJ IDEA – это многофункциональная интегрированная среда разработки, которая поддерживает множество языков программирования и технологий [7]. Она обеспечивает тесную интеграцию с JUnit, позволяя легко создавать, запускать и отлаживать тесты.

Для сборки проекта выбран инструмент Gradle [8]. Gradle – это современная система автоматизации сборки, которая поддерживает декларативный стиль описания сборочного процесса и конфигураций, а также предоставляет гибкость и расширяемость за счет использования языка программирования Groovy [9]. Gradle позволяет управлять зависимостями, компиляцией, тестированием и упаковкой проекта, что делает его мощным инструментом для реализации комплексных процессов сборки.

3.2. Реализация системы

Далее будет приведено описание реализации основных классов системы:

1) класс `TestSettings`;
2) класс `ApplicationManager`;
3) класс `Runner`.

Класс TestSettings

Класс `TestSettings` – важный компонент системы, обеспечивающий загрузку и управление конфигурационными параметрами из файла свойств формата «.properties» (рисунок 5).

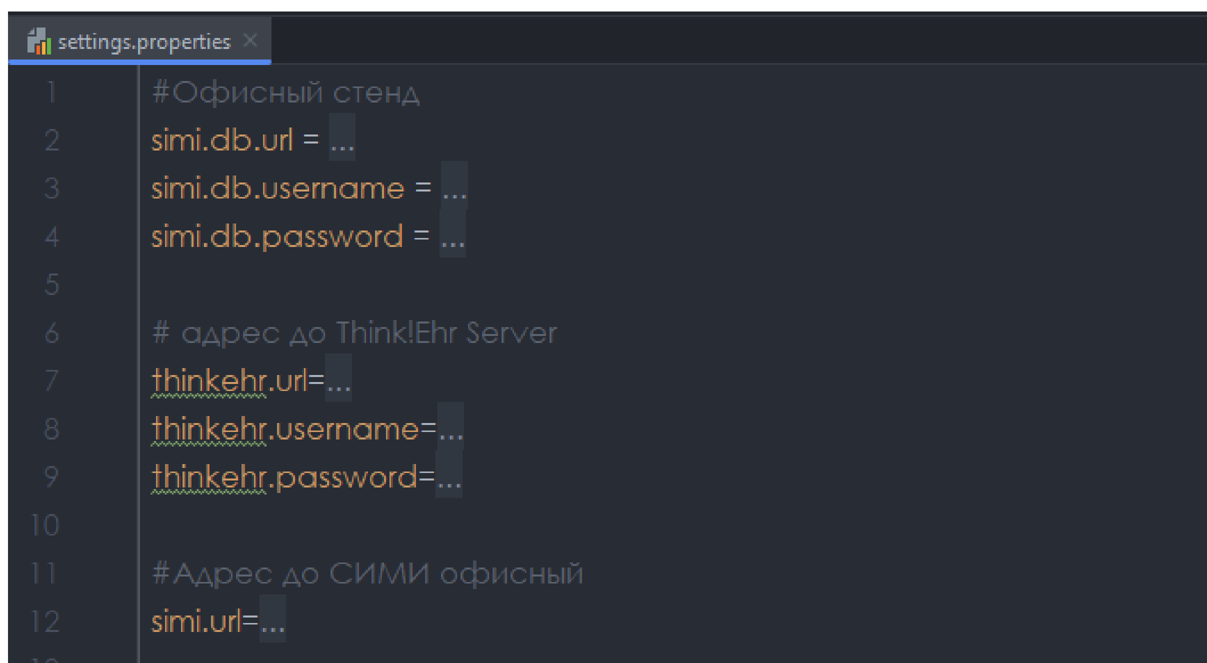

Рисунок 5 – Файл свойств «settings.properties»

Файл свойств представляет собой текстовый файл, который содержит набор пар «ключ-значение», где каждый ключ ассоциирован с определенным параметром настройки. Использование файлов свойств позволяет легко модифицировать параметры конфигурации без необходимости изменения исходного кода приложения, что значительно упрощает процесс поддержки и настройки системы.

Поля класса `TestSettings` представлены в листинге 1.

Листинг 1 – Поля класса TestSettings

```
private static final String SYSTEM_PROPERTIES_FILE_NAME = "/settings.prop-
erties";
private final Properties properties;
```

Переменная SYSTEM_PROPERTIES_FILE_NAME – константа, определяющая имя файла свойств по умолчанию. В данном случае файл находится в корневом каталоге ресурсов. Указание имени файла в виде константы позволяет централизованно управлять этим параметром, упрощая его изменение в будущем и обеспечивая единообразие в использовании конфигурационных параметров во всех частях программы.

Переменная properties – экземпляр одноименного класса Properties, который используется для хранения загруженных параметров конфигурации [12]. Класс Properties из стандартной библиотеки Java предоставляет удобные методы для работы с конфигурационными данными, включая загрузку данных из файла, их сохранение и доступ к значениям по ключам.

Класс TestSettings имеет два конструктора, которые обеспечивают гибкость и удобство при инициализации объектов класса. Код конструкторов представлен в листинге 2.

Листинг 2 – Конструкторы класса TestSettings

```
public TestSettings() {
    this(null);
}
public TestSettings(String filename) {
    if (filename == null) {
        filename = SYSTEM_PROPERTIES_FILE_NAME;
    }
    properties = new Properties();
    try (InputStream inputStream = TestSettings.class.getResource-
AsStream(filename);
         InputStreamReader inputStreamReader = new InputStreamReader(in-
putStream, "UTF8")) {
        properties.load(inputStreamReader);
        System.out.println(properties);
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Первый конструктор используется в случае, если в проекте должны использоваться свойства конфигурации по умолчанию. Он вызывает второй конструктор, передавая в него значение `null`, и в результате используется файл свойств, путь до которого был задан константой `SYSTEM_PROPERTIES_FILE_NAME`. Такое решение позволяет создавать экземпляр класса `TestSettings` без необходимости явно указывать имя файла, что делает использование класса более удобным и интуитивно понятным в случае, когда настройки хранятся в отдельном конфигурационном файле.

Второй конструктор принимает имя файла свойств в качестве переменной. Если переменная `filename` равна `null`, используется путь, указанный константой `SYSTEM_PROPERTIES_FILE_NAME`. Далее конструктор открывает поток для чтения файла с помощью метода `getResourceAsStream`. Он позволяет загружать файлы из ресурсов приложения. `InputStreamReader` с указанием кодировки «UTF-8» обеспечивает корректное чтение файла, что особенно важно при работе с текстовыми данными, содержащими символы в различных кодировках.

Далее конструктор загружает содержимое файла свойств в переменную `properties` с помощью метода `load` класса `Properties`. В случае возникновения ошибки ввода-вывода `IOException`, конструктор выбрасывает исключение, что позволяет быстро выявить и обработать проблему на этапе выполнения программы.

Класс `TestSettings` содержит набор методов для получения значений различных параметров конфигурации. Эти методы обеспечивают удобный доступ к настройкам, хранящимся в `properties`. Пример получения URL, логина и пароля от базы данных СИМИ приведен в листинге 3.

Листинг 3 – Методы получения свойств

```
public String getSimiDbUrl() {
    return properties.getProperty("simi.db.url");}
public String getSimiDbUsername() {
    return properties.getProperty("simi.db.username");}
public String getSimiDbPassword() {
    return properties.getProperty("simi.db.password");
}
```

Каждый из этих методов извлекает значение соответствующего параметра конфигурации по заданному ключу из переменной `properties` с помощью метода `getProperty`. Например, метод `getSimiDbUrl` возвращает URL для подключения к базе данных, а методы `getSimiDbUsername` и `getSimiDbPassword` возвращают имя пользователя и пароль для доступа к этой базе данных. Аналогично, методы `getThinkehrUrl`, `getThinkehrUsername` и `getThinkehrPassword` обеспечивают доступ к параметрам подключения и учетным данным для сервиса «ThinkEHR», а метод `getSimiUrl` возвращает URL для подключения к сервису «СИМИ».

Класс ApplicationManager

Класс `ApplicationManager` предназначен для инициализации и управления компонентами системы. Код этого класса представлен в листинге 4.

Листинг 4 – Класс `ApplicationManager`

```
public class ApplicationManager {
    private static final TestSettings testSettings = new TestSettings();
    private SimiServiceHelper simiServiceHelper;
    public SimiServicePortType simiService;
    private DbHelper dbHelper;
    public simi.document.v5.Signature signatureV5Example;
    public simi.document.v5.Signature signatureForSigning;
    public ApplicationManager() {
    }
    public void init() throws IOException {
        dbHelper = new DbHelper();
        signatureV5Example = SignatureSerializer.deserializeSigna-
tureV5FromFile("/signatureV5_sample.xml");
        signatureForSigning = SignatureSerializer.deserializeSigna-
tureV5FromFile("/signForSIGNING.xml");
        simiService = SimiServiceClient.getSimiService(testSet-
tings.getSimirUrl());
        simiServiceHelper = new SimiServiceHelper(signatureV5Example, si-
miService);
    }
    public DbHelper db() {
        return dbHelper;
    }
    public SimiServiceHelper simiServiceHelper () {
        return simiServiceHelper;
    }
}
```

В этом классе реализуются следующие необходимые для выполнения тестов переменные.

1. Подписи signatureV5Example и signatureForSigning, которые будут использоваться при необходимости подписать документ.

2. Переменная testSettings, в которой хранятся все необходимые конфигурационные данные.

3. SimiServiceHelper и DbHelper – компоненты, с помощью которых тесты будут взаимодействовать с сервисом СИМИ и другими. Перед определением simiServiceHelper также происходит присваивание делегата службы СИМИ в переменную simiService. Для этого используется метод getSimiService класса SimiServiceClient, в параметрах которого передается URL нужного сервиса. Далее сформированный делегат передается в соответствующий хелпер, и дальнейшее взаимодействие тестов происходит именно с хелперами.

Класс Runner

Runner – класс, с помощью которого происходит взаимодействие пользователя с системой. В этом классе реализуется определение наборов тестов, их запуск, а также вывод результатов тестирования в консоль.

Всего в классе реализовано 3 функции:

1) main;
2) getTestsStructure;
3) runTests.

Функция getTestsStructure вызывается один раз в начале выполнения функции main. Реализация этой функции представлена в листинге 5.

Листинг 5 – Функция getTestsStructure

```
public Map<String, Map<Integer, String>> getTestsStructure() {
    Map<String, Map<Integer, String>> testsStructure = new HashMap<>();
    Reflections reflections = new Reflections(ClasspathHelper
            .forPackage("simiV5.tests"), new SubTypesScanner(false));
    Set<Class<?>> subTypes = reflections.getSubTypesOf(Object.class);

    for (Class<?> clazz : subTypes) {
        String className = clazz.getName();
        if (className.contains("Tests")) {
            Method[] methods = clazz.getDeclaredMethods();
            Map<Integer, String> methodsMap = new HashMap<>();
            int index = 1;
            for (Method method : methods) {
                if (method.isAnnotationPresent(Test.class)) {
```

```
                    methodsMap.put(index++, method.getName());
                    testsStructure.put(StringUtils.substringAfter(class-
Name, "simiV5.tests."), methodsMap);
                }
            }
        }
    }
    return testsStructure;
}
```

В этой функции используется библиотека `Reflections` для получения структуры определенного пакета. В данном случае метод считывает классы, находящиеся в указанном пакете, проверяет, что названия классов содержат слово «Tests», и добавляет эти классы в словарь `testsStructure`, который в итоге будет содержать всю структуру пакета в формате {«Название класса» : {1 : «название метода»}}. Эта структура будет возвращена в основной метод main и будет использоваться для вывода и определения тестов, которые необходимо выполнить.

Функция `main` реализует взаимодействие пользователя с системой, то есть выбор варианта действия, затем сформированная с помощью метода `getTestsStructure` структура используется для выбора одного набора или нескольких наборов тестов (если необходимо), а также выбора одного или нескольких тестов (если необходимо). После того, как пользователь выбрал все необходимые данные, формируется набор селекторов, который передается в функцию `runTests`. Пример реализации варианта действия «Запуск одного набора тестов» представлен в листинге 6.

Листинг 6 – Реализация запуска одного набора тестов в функции `main`

```
System.out.println("Choose set: ");
Map<Integer, String> indexedSets = new HashMap<>();
int setIndex = 1;
while (structureIterator.hasNext()) {
    Map.Entry<String, Map<Integer, String>> entry = structureItera-
tor.next();
    indexedSets.put(setIndex, entry.getKey());
    System.out.println(setIndex++  + ". " + entry.getKey());
}
int executableSet = 0;
while (executableSet < 1 || executableSet > structure.size()) {
    if (in.hasNextInt()) {
        executableSet = in.nextInt();
    } else {
        in.next();
        System.out.println("Choose set: ");
```

```
    }
}

String executableSetName = indexedSets.get(executableSet);
selectorsMap.put(executableSetName, new ArrayList<>(structure.get(executa-
bleSetName).values()));
runTests(selectorsMap);
```

Функция `runTests` использует класс `Launcher` из JUnit для запуска выбранных тестов. Она получает на вход структуру формата {«Название класса» : [«Название теста 1», «Название теста 2», …]}, далее эта структура преобразуется в `ArrayList`, состоящий из данных типа `MethodSelector`. С помощью сформированного `ArrayList'а` формируется запрос типа `LauncherDiscoveryRequest` и выполняется с помощью класса `Launcher`. После выполнения всех выбранных тестов результаты записываются в переменную `summary` типа `TestExecutionSummary` – этот класс позволяет составлять краткое описание результатов тестирования и отображать их в нужной форме. В данном случае использовались методы для отображения количества найденных, запущенных, успешно пройденных, проваленных и пропущенных тестов. Более подробное описание проваленных тестов отображается при помощи метода `printFailuresTo`. Код функции `runTests` приведен в листинге 7.

Листинг 7 – Функция `runTests`

```
public void runTests(Map<String, ArrayList<String>> selectorsMap) {
    ArrayList<MethodSelector> selectors = new ArrayList<>();

    for (Map.Entry<String, ArrayList<String>> entry : selectorsMap.en-
trySet()) {
        String className = entry.getKey();
        for (String testName : entry.getValue()) {
        selectors.add(selectMethod("simiV5.tests." + className, testName));
        }
    }

    SummaryGeneratingListener listener = new SummaryGeneratingListener();
    LauncherDiscoveryRequest request = LauncherDiscoveryRequestBuilder.re-
quest()
            .selectors(selectors)
            .build();
    Launcher launcher = LauncherFactory.create();
    launcher.registerTestExecutionListeners(listener);
    launcher.execute(request);
    TestExecutionSummary summary = listener.getSummary();
    System.out.println(summary.getTestsFoundCount() + " tests found");
    System.out.println(summary.getTestsStartedCount() + " tests started");
```

```
  System.out.println(summary.getTestsSucceededCount()+" tests successful");
  System.out.println(summary.getTestsFailedCount() + " tests failed");
  System.out.println(summary.getTestsSkippedCount() + " tests skipped");
  summary.printFailuresTo(new PrintWriter(System.out));
}
```

3.3. Реализация тестов

Для того чтобы иметь возможность использовать компоненты системы, такие как `SimiServiceHelper`, в каждом классе с тестами был инициализирован экземпляр класса `ApplicationManager` и выполнен его метод `init`. Такой код в дальнейшем позволит классу использовать для построения тестов все необходимые компоненты системы. Код представлен в листинге 8.

Листинг 8 – Инициализация компонентов системы

```
ApplicationManager app = new ApplicationManager();

@BeforeEach
public void setApp() {
    try {
        app.init();
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Ввиду большого количества тестов для демонстрации реализации были выбраны 3 теста – по одному из каждого набора тестов.

Тест из набора BaseTests

Для первого примера был выбран тест «удаление черновика xml» из набора BaseTests. Код тестового метода представлен на листинге 9.

Листинг 9 – Тест «удаление черновика xml»

```
@Test
public void deletingDraftXML() throws Exception {
    Document createDocument = app.simiServiceHelper().createDocument(pa-
tientId, cctXML);

    createDocument.getContent().setData(Utils.decodeDataFromFiles(scan-
DataPath));
    Document saveDocumentDraft = app.simiServiceHelper().saveDocument(cre-
ateDocument);

    Assertions.assertEquals(Status.DRAFT, saveDocumentDraft.get-
Metadata().getStatus());
    Assertions.assertEquals("application/xml", saveDocumentDraft.getCon-
tent().getMimeType());
```

```
    Document getDocument = app.simiServiceHelper().getDocument(saveDocu-
mentDraft.getMetadata().getDocumentId());
    Assertions.assertEquals(Status.DRAFT, getDocument.getMetadata().getSta-
tus());

    app.simiServiceHelper().deprecateDocument(saveDocumentDraft.get-
Metadata().getDocumentId());

    Fault fault = Assertions.assertThrows(Fault.class, () -> app.simiS-
erviceHelper().getDocument(saveDocumentDraft.getMetadata().getDocumen-
tId()));
    Assertions.assertEquals("Document registry item not found.", fault.get-
Message());
}
```

Аннотация `@Test`, предоставленная фреймворком JUnit, указывает на то, что метод является тестовым [13].

В первую очередь метод отправляет запрос на создание документа. В качестве параметров указываются `patientId`, а также `cctXML` – переменные-константы, хранящие данные для создания документов. В случае успеха, в ответе мы получаем созданный документ, на что указывает тип переменной `createDocument` – `Document`. Это сгенерированный из WSDL Java-класс, с помощью которого далее можно будет взаимодействовать с полученным документом. В случае, если после отправки запроса нам не вернется документ, тест провалится и JUnit выведет соответствующее сообщение. Далее в ранее полученный документ устанавливается необходимый контент документа и выполняется запрос на сохранение документа. В случае успеха, также, как и при создании, вернется документ.

`Assertions` – утверждения, которые предоставляет JUnit для тестирования различных сценариев [14]. В случае, если хотя бы одно утверждение не получит ожидаемого результата, весь тест будет считаться непройденным. В данном случае использовано утверждение `assertEquals` – оно проверяет, что ожидаемое значение (параметр 1) и фактическое значение (параметр 2) равны. В первом случае проверяется, что статус возвращенного документа «черновик», а во втором случае, что тип контента документа «XML».

Далее по порядку выполняются следующие действия.

1. Запрос на получение документа.
2. Проверка статуса документа после его получения.
3. Запрос на удаление документа – в данном случае в качестве ответа ничего не возвращается, так как документ должен быть удален.

Последними утверждениями является комбинация `assertThrows` и `assertEquals`. Первое утверждение ожидает, что в качестве ответа вернется ошибка. Сообщение ошибки можно получить с помощью метода `getMessage`, который используется далее в качестве фактического результата проверки `assertEquals`.

Тест из набора OptionsTests

Для второго примера был выбран тест «сохранение документа с удаленной ассоциацией» из набора OptionsTests. Код представлен в листинге 10.

Листинг 10 – Тест «сохранение документа с удаленной ассоциацией»

```
@Test
public void savingWithDeletedAssociation() throws Exception {
    Document createDocument1 = app.simiServiceHelper().createDocument(pa-
tientId, cctTDD);

    String docIdForAssociation = "…";

    Options optionsAddAssociation = new Options();
    optionsAddAssociation.getEntry().add(Utils.createKeyValuePairV5(
            "addAssociation","{…}"));
    createDocument1.getSignature().add(app.signatureV5Example);
    createDocument1.getContent().setData(Utils.decodeDataFrom-
Files(tddDataPath));

    Fault fault = Assertions.assertThrows(Fault.class, () -> app.simiS-
erviceHelper().saveDocument(createDocument1, optionsAddAssociation));
    Assertions.assertEquals("Can't get class code for associated docu-
ment.", fault.getMessage());
}
```

Суть данного теста заключается в добавлении опции с удаленной ассоциацией к запросу на сохранение документа:

1) выполняется запрос на создание документа;
2) формируется id документа с удаленной ассоциацией;
3) создается экземпляр класс `Options`;
4) добавляется опция, содержащая в себе удаленную ассоциацию;

5) на созданный документ добавляется подпись;

6) устанавливается необходимый контент документа;

7) с помощью утверждения `assertThrows` выполняется запрос на сохранение документа с указанной выше опцией;

8) полученное сообщение об ошибке сравнивается с ожидаемым сообщением.

Данный тест-кейс будет считаться пройденным, если при отправке запроса на сохранение в ответе вернется ошибка.

Тест из набора DifferentTests

Для третьего примера был выбран тест «удаление композиции» из набора DifferentTests. Код тестового метода представлен в листинге 11.

Листинг 11 – Тест «удаление композиции»

```
@Test
public void checkDeletingComposition() throws Exception {
    Document createDocument = app.simiServiceHelper().createDocument(pa-
tientId, cctTDD);

    createDocument.getContent().setData(Utils.decodeDataFrom-
Files(tddDataPath));
    Document saveDocumentDraft = app.simiServiceHelper().saveDocument(cre-
ateDocument);
    Assertions.assertEquals(Status.DRAFT, saveDocumentDraft.get-
Metadata().getStatus());
    Assertions.assertEquals(saveDocumentDraft.getContent().getMimeType(),
"application/tdd");

    Document getDocument = app.simiServiceHelper().getDocument(saveDocu-
mentDraft.getMetadata().getDocumentId());
    Assertions.assertEquals(Status.DRAFT, saveDocumentDraft.get-
Metadata().getStatus());

    getDocument.getSignature().add(app.signatureV5Example);
    Document saveDocumentSigned = app.simiServiceHelper().saveDocument(get-
Document);
    Assertions.assertEquals(Status.SIGNED, saveDocumentSigned.get-
Metadata().getStatus());

    String storageDocumentId = app.db().getStorageDocumentId(saveDocu-
mentSigned.getMetadata().getDocumentId()).substring(0, 36);
    app.db().checkComposition(storageDocumentId, true);

    Thread.sleep(deprecationTimeoutMs);

    app.simiServiceHelper().deprecateDocument(saveDocumentSigned.get-
Metadata().getDocumentId());
    app.db().getStorageDocumentId(saveDocumentSigned.getMetadata().getDocu-
mentId());
    app.db().checkComposition(storageDocumentId, false);
}
```

В первую очередь в тесте выполняются следующие действия:

1) создается документ;
2) документ сохраняется в статусе черновика;
3) документ сохраняется в статусе подписанного.

Далее отправляется запрос в базу данных, ответом на который является id композиции документа, выполняется метод `checkComposition`, который проверяет, должна ли композиция существовать в базе данных. Параметр 2 этого метода передает необходимое состояние композиции: `true` – композиция должна существовать, `false` – композиции не должно быть в базе данных. В данном случае во втором параметре указано значение `true`, поэтому проверка будет пройдена при наличии в базе переданного в метод `storageDocumentId`.

Далее выполняется запрос на удаление документа и выполняется метод `checkComposition` с флагом `false` – композиция должна быть удалена.

Вывод по третьей главе

В третьей главе была разработана система автоматизированного тестирования для продукта СИМИ. В главе представлено описание основных классов программы, а именно: `TestSettings`, `ApplicationManager` и `Runner`. Также были разработаны ранее спроектированные наборы тестов и приведены по одному примеру реализаций на каждый из наборов.

4. ТЕСТИРОВАНИЕ

Для тестирования системы использовалось функциональное тестирование – проверка системы на соответствие функциональным требованиям [15]. Результаты функционального тестирования представлены в таблице 1.

Таблица 1 – Функциональное тестирование

№	Название теста	Шаги	Ожидаемый результат	Тест пройден?
1	Запуск одного теста	1. В консольном интерфейсе выбрать первый вариант использования – «Running one test from a set». 2. Выбрать один из предложенных наборов тестов. 3. Выбрать один из предложенных тестов.	Тест выполнился, в консольном интерфейсе появилось описание результатов тестирования с пройденным тестом.	Да
2	Запуск нескольких тестов из одного набора	1. В консольном интерфейсе выбрать пятый вариант использования – «Running multiple tests from one test suite». 2. Выбрать один из предложенных наборов тестов. 3. Выбрать несколько тестов.	Тесты выполнились, в консольном интерфейсе появилось описание результатов тестирования с пройденными тестами.	Да
3	Запуск одного набора тестов	1. В консольном интерфейсе выбрать второй вариант использования – «Running one test set». 2. Выбрать один из предложенных наборов тестов.	Выполнение всех тестов из выбранного набора, в консольном интерфейсе отображаются результаты тестирования.	Да
4	Запуск нескольких наборов тестов	1. В консольном интерфейсе выбрать четвертый вариант использования – «Running multiple test sets». 2. Выбрать несколько наборов тестов.	Выполнение всех тестов из выбранных наборов, в консольном интерфейсе отображаются результаты тестирования.	Да
5	Запуск всех наборов тестов	В консольном интерфейсе выбрать третий вариант использования – «Running all test sets».	Выполнение всех тестов, в консольном интерфейсе отображаются результаты тестирования.	Да

Вывод по четвертой главе

В рамках данной главы было проведено функциональное тестирование системы. Фактические результаты совпадают с ожидаемыми, тестирование пройдено успешно.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была разработана система автоматизированного тестирования для многопользовательского продукта СИМИ с использованием JUnit. В ходе выполнения данной работы были решены следующие задачи.

1. Выполнен анализ предметной области.
2. Спроектирована система и наборы тестов.
3. Разработана система и наборы тестов.
4. Проведено тестирование системы.

В настоящий момент разработанная в ходе выполнения данной работы система уже введена в эксплуатацию, что подтверждается актом о внедрении.

Система интегрированной медицинской информации постоянно обновляется и требует тестирования новых и уже использующихся компонентов, поэтому в будущем планируется проектирование новых наборов тестов, а также расширение и обновление уже существующих. Также планируется интеграция системы автоматизированного тестирования в инструменты CI для более удобного запуска и мониторинга выполнения тестов.

ЛИТЕРАТУРА

1. Junit documentation. [Электронный ресурс] URL: https://junit.org/junit5/ (дата обращения: 31.01.2024 г.).
2. Медицинский портал ЕМИАС.ИНФО. [Электронный ресурс] URL: https://emias.info/ (дата обращения: 01.02.2024 г.).
3. Назначение и функции систем интегрированной медицинской информации. [Электронный ресурс] URL: https://eduherald.ru/ru/article/view?id=12082 (дата обращения: 01.02.2024 г.).
4. Информационный город. [Электронный ресурс] URL: https://budget.mos.ru/budget/gp/passports/12 (дата обращения: 01.02.2024 г.).
5. UML. [Электронный ресурс] URL: https://www.uml-diagrams.org/ (дата обращения: 25.02.2024 г.).
6. Java Software. [Электронный ресурс] URL: https://www.java.com/ru/ (дата обращения: 26.02.2024 г.).
7. IntelliJ IDEA documentation. [Электронный ресурс] URL: https://www.jetbrains.com/help/idea/javadocs.html (дата обращения: 26.02.2024 г.).
8. Gradle documentation. [Электронный ресурс] URL: https://docs.gradle.org/current/userguide/userguide.html (дата обращения: 15.03.2024 г.).
9. Groovy documentation. [Электронный ресурс] URL: https://docs.groovy-lang.org/latest/html/documentation/ (дата обращения: 16.03.2024 г.).
10. TestNG. [Электронный ресурс] URL: https://testng.org/ (дата обращения: 28.03.2024 г.).
11. Selenium Documentation. [Электронный ресурс] URL: https://www.selenium.dev/documentation/ (дата обращения: 28.03.2024 г.).
12. Java Properties. [Электронный ресурс] URL: https://hr-vector.com/java/properties (дата обращения: 10.04.2024 г.).
13. A Guide to JUnit 5. [Электронный ресурс] URL: https://www.baeldung.com/junit-5 (дата обращения 11.04.2024 г.).

14. Junit assertions. [Электронный ресурс] URL: https://www.baeldung.com/junit-assertions (дата обращения: 13.04.2024 г.).

15. Functional Testing. [Электронный ресурс] URL: https://www.geeksforgeeks.org/software-testing-functional-testing/ (дата обращения: 02.05.2024 г.).