

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Программное обеспечение средств вычислительной техники и автоматизированных систем»

БАКАЛАВРСКАЯ РАБОТА

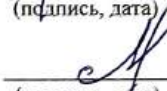
на тему

**Бизнес-проект «Система модульного тестирования
интернет-сервиса для зачисления в образовательные
организации»**

Студент

 24.05.2026 Воробьева М.М.
(подпись, дата)

Руководитель

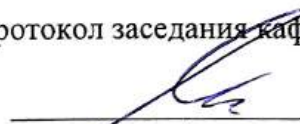
 24.05.2026 Акифьев И.В.
(подпись, дата)

Нормоконтролёр

 24.05.2026 А.С. Бычков
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 22.05.2026 № 14)

Заведующий кафедрой

 М.А. Митрохин
(подпись)

Работа защищена с отметкой  (протокол заседания ГЭК от
02 июня 2026 № 9)

Секретарь ГЭК

 Е.И. Гурин
(подпись)

Пенза 2026

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

Факультет
вычислительной техники

Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
« 7 » апреля 2026г.

**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студентка Воробьева Маргарита Михайловна гр.22ВВП1 факультета вычислительной техники.
2. Тема работы: «Бизнес-проект «Система модульного тестирования интернет-сервиса для зачисления в образовательные организации»

Тема утверждена приказом ПГУ № 347/о от 07.04.2026 г.

3. Руководитель бакалаврской работы к.э.н, доцент Акифьев И.В.

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

Требуется разработать систему модульного тестирования интернет-сервиса для зачисления в образовательные организации, позволяющую автоматизировать процесс проверки ключевых компонентов сервиса. Для реализации системы модульного тестирования требуется реализовать возможность выбора модульных тестов для запуска, реализовать возможность для пользователя запуска тестов в различных выборочных комбинациях, предоставить пользователю возможность видеть результат выполнения тестирования и экспортировать его в виде отчета, реализовать хранение и предоставление истории и статистики запуска тестов, разработать интуитивно-понятный для пользователя интерфейс, позволяющий управлять процессом тестирования и анализировать результаты, разработать систему, включающую в себя набор разработанных и отлаженных модульных тестов для ключевых модулей интернет-сервиса.

5. Перечень подлежащих разработке вопросов

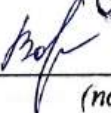
Требуется произвести анализ предметных областей: изучить аналоги существующих систем тестирования, проанализировать и выбор технологий, разработать модульные тесты, разработать приложение для запуска тестов с графическим интерфейсом на Java Swing, обеспечивающее выбор модулей, программный запуск через JUnit Platform Launcher, отображение лога и прогресса, формирование отчётов, ведение истории и статистики, провести тестирование и отладку разработанной системы, описать бизнес-модель проекта.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Изучение предметной области	48	07.04.2026 – 12.04.2026	
2	Анализ требований	16	13.04.2026 – 14.04.2026	
3	Выбор технологий, применяемых при проектировании и разработке	48	15.04.2026 – 20.04.2026	
4	Описание бизнес-модели	40	21.04.2026 – 25.04.2026	
5	Разработка бизнес-проекта «Система модульного тестирования интернет-сервиса для зачисления в образовательные организации»	184	26.04.2026 – 18.05.2026	
6	Оформление пояснительной записки	24	19.05.2026 – 21.05.2026	

Дата выдачи задания "7" апреля 2026 г.

Руководитель бакалаврской работы  Акифьев И.В.
(подпись, дата)

Задание к исполнению принял студент  Воробьева М.М.
(подпись, дата)

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

27.05.2026

Реферат

Бакалаврская работа содержит 161 лист, 31 рисунок, 5 таблиц, 15 источников и 3 приложения.

МОДУЛЬНОЕ ТЕСТИРОВАНИЕ, ИЗОЛЯЦИЯ, ИНТЕРНЕТ-СЕРВИС ЗАЧИСЛЕНИЯ В ОБРАЗОВАТЕЛЬНЫЕ ОРГАНИЗАЦИИ, ПРИЛОЖЕНИЕ ДЛЯ ЗАПУСКА ТЕСТОВ, ИСТОРИЯ ЗАПУСКОВ, СТАТИСТИКА ТЕСТИРОВАНИЯ

Цель работы – разработать систему модульного тестирования для интернет-сервиса зачисления в образовательные организации, обеспечивающую автоматизированный запуск тестов, сбор и анализ результатов, ведение истории запусков и формирование отчётов.

Объектами разработки являются:

– набор модульных тестов для пяти ключевых сервисов интернет-сервиса зачисления;

– приложение для запуска тестов с графическим интерфейсом для выбора тестовых модулей, программного запуска, формирования итогового отчёта с возможностью экспорта, ведения истории запусков с визуальными трендами и расчёта статистики.

В результате проделанной работы разработана система, полностью соответствующая поставленной задаче, а именно: реализованы 30 модульных тестов; создано приложение для запуска тестов, которое позволяет пользователю выбирать тестовые модули, запускать их в отдельном потоке, просматривать и экспортировать итоговые отчёты, хранить историю запусков с отображением трендов и вычислять сводную статистику.

					ПГУ 1.090301.05.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Бизнес-проект «Система модульного тестирования интернет-сервиса для зачисления в образовательные организации» Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Воробьева М.М.		24.05.24					
Провер.		Акифьев И.В.		24.05.24			2		
Н. Контр		Бычков А.С.		24.05.24		ФВТ, каф ВТ гр. 22ВВП1			
Утверд.		Митрохин М.А.		24.05.24					

Содержание

Перечень сокращений и обозначений	8
Введение	9
1 Система модульного тестирования для интернет-сервиса зачисления в образовательные организации	10
1.1 Анализ системной области	10
1.2 Постановка задач	14
1.3 Технические требования	15
1.4 Обзор аналогов	16
1.5 Выбор и настройка инструментов разработки	19
1.5.1 Язык программирования Java	20
1.5.2 Выбор среды разработки IntelliJ IDEA	20
1.5.3 Фреймворк для модульного тестирования JUnit 5	23
1.5.4 Библиотека для создания mock-объектов Mockito	24
1.6 Проектирование системы модульного тестирования	25
1.6.1 Архитектура системы	25
1.6.2 Взаимодействие компонентов системы	27
1.6.3 Реализация модульных тестов	29
1.6.4 Модуль тестирования ApplicationCreationCheckServiceComplexTest	31
1.6.5 Модуль тестирования ExpulsionGroupOrderServiceTest	33
1.6.6 Модуль тестирования ExamGroupServiceTest	37
1.6.7 Модуль тестирования GradeBookServiceTest	40
1.6.8 Модуль тестирования ProfSettingAssertionServiceTest	43
1.6.9 Реализация приложения для запуска модульных тестов	45
1.6.10 Описание графического интерфейса приложения	52

1.7 Тестирование и отладка.....	57
1.7.1 Тестирование модульных тестов.....	58
1.7.2 Тестирование приложения для запуска тестов	65
1.8 Руководство пользователя.....	69
1.8.1 Начало работы	70
1.8.2 Просмотр истории запусков.....	75
1.8.3 Просмотр статистики	77
2 Бизнес-модель.....	79
3 Охрана труда.....	83
Заключение	86
Список использованных источников	87
Приложение А – UML-диаграммы	89
Приложение Б – Листинг программы	100
Приложение В – Презентация.....	153

Перечень сокращений и обозначений

В настоящей работе применяют следующие обозначения и сокращения

БД – база данных

ОР – ожидаемый результат

Рефакторинг кода – процесс улучшения внутренней структуры программного кода без изменения его внешнего поведения

ФР – фактический результат

API (Application Programming Interface) – описание способов, которыми одна компьютерная программа может взаимодействовать с другой программой

GUI (Graphical User Interface) – графический интерфейс пользователя

IDE (Integrated development environment) – интегрированная среда разработки

SDK (Software Development Kit) – комплект разработки программного обеспечения

UML (Unified Modeling Language) – унифицированный язык моделирования

					ПГУ 1.090301.05.001 ПЗ	Лист
						3
Изм.	Лист	№ докум.	Подпись	Дата		

Введение

Современные интернет-сервисы для зачисления в образовательные организации становятся всё сложнее. Они включают множество взаимосвязанных частей: подачу заявлений, формирование приказов, учёт зачётных книжек, экзаменационные группы и интеграцию с другими системами. Ошибки в таких сервисах могут нарушить весь процесс зачисления, поэтому важно находить их на ранних этапах разработки. Один из эффективных способов – модульное тестирование, которое позволяет проверять отдельные компоненты программы автоматически.

Цель данной работы – разработать систему модульного тестирования для интернет-сервиса зачисления в образовательные организации. Система должна состоять из самих модульных тестов и приложения для их запуска, которое должно обеспечивать запуск тестов, показ результатов, ведение истории проверок и формирование отчётов. Также требуется описать и проанализировать бизнес-модель этого проекта.

Практическая ценность работы заключается в том, что разработанную систему можно сразу применять для регулярной проверки образовательного сервиса. Она экономит время и ресурсы на поиск ошибок, повышает надёжность продукта, помогает собирать статистику по тестированию.

1 Система модульного тестирования для интернет-сервиса зачисления в образовательные организации

1.1 Анализ системной области

В данный момент активно происходит цифровая трансформация (внедрение цифровых технологий) во многие сферы нашей жизни, сфера образования не является исключением, в связи с этим надежная работа таких систем, как онлайн-системы для зачисления абитуриентов в учебные заведения, становится одной из важнейших задач.

Во время приемных кампаний такие сервисы испытывают огромные нагрузки, и даже малейший, кратковременный сбой сможет обернуться серьезными последствиями. Массовая подача документов абитуриентами одновременно при сбое может вызвать огромное количество проблем, от нарушения прав на образование и проблем с зачислением до ухудшения репутации учебных заведений.

Высокая нагрузка также сохраняется в течение учебного года, поскольку современные сервисы для зачисления абитуриентов работают не только во время приемных кампаний, они помимо этого занимаются выпуском различных приказов, связанных со студентами, например, о предоставлении академического отпуска, отслеживают процесс обучения, занимаются переводом студентов из одного учебного заведения в другое, выполняют множество другой важной для сферы образования работы.

Современные образовательные платформы представляют собой сложные программные продукты, состоящие из множества взаимосвязанных компонентов, где даже небольшое изменение кода может привести к непредсказуемым последствиям. Поэтому специалистам постоянно приходится тестировать код с целью выявления и исключения возможных недостатков.

Тестирование – это процесс исследования и оценки программного продукта с целью проверки его соответствия заявленным требованиям и выявления потенциальных дефектов (багов или ошибок) до того, как с ними столкнутся

					ПГУ 1.090301.05.001 ПЗ	Лист
						5
Изм.	Лист	№ докум.	Подпись	Дата		

конечные пользователи. Этот системный подход помогает обеспечить качество, надежность и соответствие ожиданиям пользователей.

Тестирование сопровождает весь цикл разработки от начала до конца. Оно включает в себя планирование, анализ требований, проектирование тестов, реализацию, выполнение и анализ результатов. В зависимости от типа тестируемых объектов и целей существуют различные виды тестирования.

Модульное тестирование или по-другому unit-тестирование – это процесс проверки программного продукта, при котором тестируются отдельные компоненты и модули в изолированном состоянии. В качестве тестируемых единиц выступают функции, методы, процедуры, класс или небольшие модули – все что можно рассматривать самостоятельную, не зависящую от других объектов единицу кода [1]. Ключевой особенностью данного метода тестирования является изоляция проверяемого компонента от остальной системы, то есть тестируется конкретный компонент, не полагаясь на работу внешних зависимостей – баз данных, веб сервисов или других модулей приложения, чтобы имитировать эти зависимости используются специальные объекты-заглушки (mock objects).

Основная цель модульного тестирования – предотвращение накопления ошибок на ранних стадиях разработки, это дает данному подходу преимущества, такие как:

Экономия средств и времени. Поскольку ошибку легче и быстрее исправить, если она была выявлена на ранней стадии тестирования;

Повышение понимания кодовой базы. Если код был написан давно или другим человеком, по сценариям модульных тестов можно определить, как правильно должен работать тестируемый модуль и особенности его использования;

Поддержка рефакторинга. Наличие модульных тестов позволяет безопасно изменять внутреннюю структуру кода, не опасаясь нарушить существующую функциональность.

Интеграционное тестирование проверяет взаимодействие различных компонентов системы и связь с другими внешними продуктами, из-за этого может

					ПГУ 1.090301.05.001 ПЗ	Лист
						6
Изм.	Лист	№ докум.	Подпись	Дата		

быть не всегда стабильным. Эти тесты требуют развертывание части инфраструктуры, время выполнения у них больше, чем у модульных, они могут занимать минуты или даже часы, имеют высокую стоимость. При неуспешном выполнении бывает сложно определить точное место проблемы, поскольку сбой может быть вызван одной из сотен зависимостей.

Сквозное тестирование проверяет полностью всю систему, как единое целое, от начала до конца, от пользовательского интерфейса до базы данных и внешних связей, интеграций, из-за этого его называют End-to-End или E2E. Тест выполняет различные пользовательские действия, тем самым проверяет различные бизнес-сценарии, и что все компоненты системы взаимодействуют корректно. В данном типе тестирования требуется развертывание полного стека технологий. Из минусов можно выделить долгое время выполнения, большая вероятность сбоя и сложности в локализации ошибки. Обычно сквозным тестированием пользуются перед выпуском релиза, чтобы окончательно удостовериться в отсутствии проблем в работе сервиса.

Нагрузочное тестирование – это проверка работы системы с различными степенями нагрузки. Главная цель метода – определить, как система справляется с реальными объемами работы и найти проблемные места, в которых производительность хуже и возможны утечки памяти.

Регрессионное тестирование – это проверка корректной работы системы при изменениях в различных местах кода, например, при исправлении багов или внедрении новых функций и возможностей, поэтому эти тесты очень важны при частых релизах.

Помимо этих видов тестирования в разработке применяются и другие, важно понимать, что у каждого вида свои задачи и возможности. Также тестирование можно классифицировать по степени автоматизации. Выделяют 3 основных типа: автоматизированное, ручное, полуавтоматическое.

Автоматизированное тестирование – это проверка работы функционала, при котором все действия (запуск самих тестов, сравнение ожидаемых и фактических

					ПГУ 1.090301.05.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

результатов, формирование отчетов и др.) выполняются автоматически с помощью специальных инструментов. В этом типе участие человека минимально, он занимается только настройкой и запуском тестов, благодаря этому у них достаточно высокая скорость выполнения и высокая точность. Применяется при регрессионном, нагрузочном, модульном тестировании.

Ручное тестирование проверяет работу без использования автоматизированных инструментов. В данном случае человек, следуя определенным сценариям, проверяет работу системы. Тестировщик выполняет аналогичные обычному пользователю действия и сравнивает ожидаемый результат с фактическим. Это очень гибкий подход к проверке функций сервиса, он позволяет вычислить множество проблем различных компонентов и оценить целостное состояние. У этого вида тестирования присутствуют и недостатки, в первую очередь – это человеческий фактор, тестировщик может не заметить проблему или наоборот воспринять корректную работу системы за ошибку, также это занимает большое количество времени. Обычно применяется в начале разработки, при тестировании новых функций или на проектах с частыми изменениями интерфейса.

Полуавтоматическое тестирование – это гибридный подход, который совмещает в себе 2 предыдущих. Часть действий выполняется автоматически, например, сбор данных, выполнение функций, а другая часть (анализ результатов, формирование отчетов) – вручную. Этот подход часто находит свое место в ситуациях, когда полная автоматизация затруднена. Скорость выполнения у этого вида средняя, как и точность из-за человеческого фактора [2].

На рисунке 1.1 представлена пирамида видов тестирования, предложенная Майком Коном в книге «Succeeding with Agile: Software Development Using Scrum». Она прекрасно показывает соотношение разных видов тестов на проекте. Модульные являются фундаментом для всех остальных тестов. Они выполняются за считанные секунды, позволяют быстро определить место проблемы и соответственно исправить ошибку, у них достаточно низкая стоимость поддержки, а автоматизация этих тестов многократно увеличивает их эффективность.

Система модульного тестирования, обеспечивающая автоматизированный запуск тестов, сохраняющая историю и предоставляющая статистику тестирования является полезным инструментом для контроля качества интернет-сервиса зачисления в образовательные организации, где любая ошибка может привести к серьезным последствиям.



Рисунок 1.1 – Пирамида видов тестирования

1.2 Постановка задач

Требуется разработать систему модульного тестирования интернет-сервиса для зачисления в образовательные организации, позволяющую автоматизировать процесс проверки ключевых компонентов сервиса.

Для реализации системы модульного тестирования требуется выполнить следующие задачи:

- реализовать возможность выбора модульных тестов для запуска;
- реализовать возможность для пользователя запуска тестов в различных выборочных комбинациях;

- предоставить пользователю возможность видеть результат выполнения тестирования и экспортировать его в виде отчета;
- реализовать хранение и предоставление истории и статистики запуска тестов;
- разработать интуитивно-понятный для пользователя интерфейс, позволяющий управлять процессом тестирования и анализировать результаты;
- разработать систему, включающую в себя набор разработанных и отлаженных модульных тестов для ключевых модулей интернет-сервиса.

1.3 Технические требования

Для функционирования разработанной системы модульного тестирования необходимо соблюдение определённых требований к аппаратному и программному обеспечению. Данные требования обусловлены используемым стеком технологий.

Операционная система. Система должна быть кроссплатформенной и работать на всех операционных системах, поддерживающих Java 21: Microsoft Windows, macOS, Linux;

Аппаратные требования. Минимальная конфигурация компьютера должна включать процессор с тактовой частотой не менее 2,0 ГГц (2 ядра), 4 ГБ оперативной памяти и 1 ГБ свободного дискового пространства. Рекомендуемая конфигурация: процессор от 2,5 ГГц (4 ядра), 8 ГБ оперативной памяти и 2 ГБ свободного места на диске.

Программные требования. Основным требованием является наличие установленной Java версии 21 (JRE или JDK). Рекомендуется сборка Eclipse Temurin 21.0.10, но допустимы любые совместимые реализации OpenJDK. Для управления зависимостями требуется Apache Maven версии 3.9 или выше. Среда разработки IntelliJ IDEA (Community Edition).

Библиотечные зависимости. Все необходимые библиотеки автоматически загружаются Maven.

					ПГУ 1.090301.05.001 ПЗ	Лист
						10
Изм.	Лист	№ докум.	Подпись	Дата		

1.4 Обзор аналогов

Для глубокого погружение в тему были рассмотрены существующие системы, позволяющие осуществлять модульное тестирование программного продукта.

Maven Surefire Plugin [3] – плагин для системы сборки Maven, который запускает тесты в процессе сборки проекта. Он привязан к фазе тестирования по умолчанию, поддерживает JUnit и TestNG. Плагин генерирует отчеты о тестировании после его выполнения в различных форматах, поддерживает параллельное выполнение тестов, их автоматизацию при сборке, является бесплатным и открытым инструментом. Из недостатков можно отметить отсутствие графического интерфейса, невозможность выборочного запуска тестов без изменения конфигурации, не сохраняет историю и статистику тестирования продукта.

JUnit 5 Console Launcher [4] – это консольное приложение фреймворка JUnit 5, предназначенное для обнаружения и запуска тестов, результаты тестирования выводятся в консоль, есть возможность их генерации и экспорта отчетов в различных форматах (.xml, .txt, json). JUnit 5 Console Launch независим от IDE и систем сборки, прост в интеграции, автономен. Минусы у данного инструмента также присутствуют – это сложность отладки, отсутствие возможности компиляции кода, проблемы с производительностью при большом количестве тестов, и сложность в анализе результатов. На рисунке 1.2 представлен результат выполнения тестов в среде IntelliJ IDEA с использованием JUnit 5



Рисунок 1.2 – Результат выполнения тестов в среде IntelliJ IDEA с использованием JUnit 5

Jenkins [5] – открытая система непрерывной интеграции, позволяющая запускать тесты и анализировать их результаты. В ней поддерживается полная автоматизация запуска тестов после каждого коммита и по расписанию, благодаря этому у разработчики экономят время и обнаруживают ошибки на ранних стадиях разработки. Имеется поддержка популярных тестовых фреймворков (JUnit, TestNG и др.), генерация наглядных отчетов о тестировании, интеграция с популярными системами (email, Telegram, Slack) для отправки уведомлений. Минусы системы – это сложность установки и настройки, ресурсоемкость, сложность отладки, подходит для крупных проектов, для небольших будет избыточной. На рисунке 1.3 представлен результат выполнения тестов с использованием Jenkins.

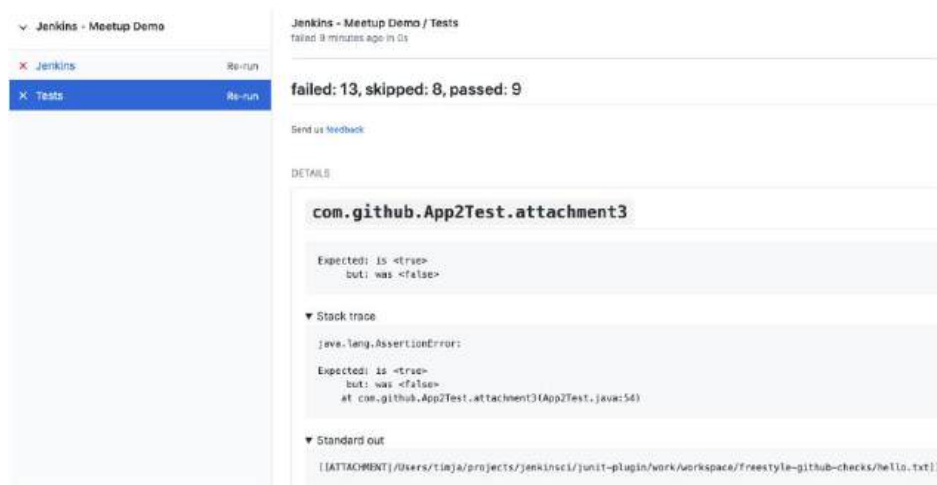


Рисунок 1.3 – Результат выполнения тестов с использованием Jenkins

GitLab CI [6] – это инструмент платформы GitLab, позволяющий автоматизировать процесс запуска тестов при каждом изменении кода. Он включает в себя инструменты генерации отчетов о тестировании, поддерживает различные фреймворки для тестирования и множество языков программирования, интегрирован с многими другими функциями GitLab. Из минусов – зависимость от GitLab, сложность локальной отладки и запуска, невысокая скорость выполнения. На рисунке 1.4 отображена сводка результатов тестирования в GitLab CI.

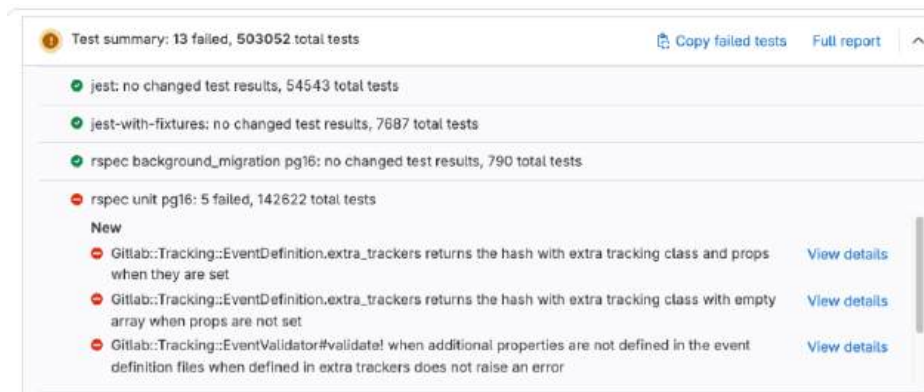


Рисунок 1.4 — Отображение сводки результатов тестирования в GitLab CI

Gradle Test [7] – плагин для автоматического обнаружения, компиляции и запуска модульных тестов в системе Gradle. Поддерживает инкрементальный подход, если в коде не было изменений, то повторный запуск тестов не выполняется, тем самым экономятся время и ресурсы. Плагин формирует наглядные отчеты о тестировании, поддерживает фреймворк JUnit 5 и параллельное выполнение тестов. К минусам можно отнести отсутствие графического интерфейса, сложность выборочного запуска тестов, зависимость от Gradle, отсутствие истории и аналитики запусков.

TeamCity [8]– система непрерывной интеграции от компании JetBrains. Может работать локально и в облаке. Предназначена для автоматизации сборки, тестирования и развертывания программных продуктов. Поддерживает множество технологий и инструментов, интегрируется с популярными системами контроля версий и отслеживания ошибок. Для небольших проектов будет избыточной. Имеет бесплатную и платную версии. Потребляет большое количество ресурсов при работе.

CicleCI [9] – облачная платформа непрерывной интеграции, автоматизирующая сборку, тестирование и развертывание приложений. Интегрируется с различными Git-хостингами. Поддерживает параллельное выполнение тестов, кеширование зависимостей, подробные отчеты по тестированию. Есть платная и бесплатная версия. Из минусов можно выделить отсутствие возможности выбора тестов для запуска, сложность локальной отладки,

зависимость от облачного сервиса, требуется постоянное подключение к сети. На рисунке 1.5 показана статистика тестирования в CircleCI.

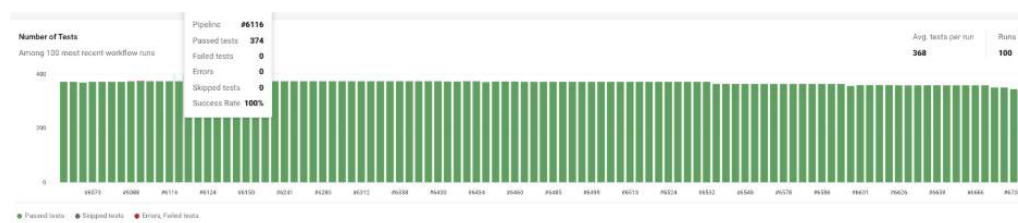


Рисунок 1.5 — Статистика тестирования в CircleCI

У существующих аналогов системы модульного тестирования есть свои плюсы и минусы. На основе этого можно сделать вывод, что существующие аналоги не предоставляют полного набора функций, необходимого для эффективного тестирования интернет-сервиса для зачисления в образовательные организации. Разработанная система сочетает в себе лучшие черты аналогов — отчетность о выполнении тестов, удобный графический интерфейс, автоматизацию запуска, а также имеет уникальные функции — история запусков тестов, сохранение статистики.

1.5 Выбор и настройка инструментов разработки

Выбор инструментов разработки для системы модульного тестирования зависел от следующих критериев:

- совместимость. Инструменты должны быть совместимы с имеющейся кодовой базой проекта, чтобы не возникало конфликтов;
- популярность. Использование популярных инструментов облегчает поиск информации о них, исправление ошибок и решение вопросов. Популярные инструменты используются многими командами, это означает, что они тщательно протестированы и удобны в использовании;
- документация. Наличие качественной документации у инструмента позволяет изучить его, понять как с ним работать;

– кроссплатформенность. Возможность работы на разных операционных системах (Windows, Linux, macOS) без потери функционала является не менее важным фактором;

– поддержка версий. Наличие долгосрочно поддерживаемых версий говорит о стабильной работе продукта и его актуальности.

1.5.1 Язык программирования Java

Поскольку кодовая база (тестируемые сервисы) была написана на языке программирования Java, то выбор этого языка для написания системы модульного тестирования обеспечил полную совместимость.

Java – объектно-ориентированный язык программирования общего назначения. Был разработан и выпущен компанией Sun Microsystems 23 мая 1995 года.

Java обеспечивает кроссплатформенность, благодаря этому не нужно поддерживать отдельные версии кода для разных платформ. Для этого языка программирования существует множество проверенных инструментов для модульного тестирования (JUnit, Mockito). Современные версии Java показывают высокую производительность, для модульного тестирования это очень важно.

У Java большое сообщество разработчиков и обилие документации, тысячи статей и курсов для изучения, что помогает при возникновении вопросов и сложностей. Многолетняя история говорит о том, что он стабильный и надежный, на данный момент стабильно выходят новые версии. Благодаря строгой обратной совместимости, написанная система модульного тестирования будет продолжать работать продолжительное время, не нуждаясь в рефакторинге [10].

1.5.2 Выбор среды разработки IntelliJ IDEA

IntelliJ IDEA Community Edition – это бесплатная среда разработки с открытым кодом, в которой есть все необходимые функции для написания и отладки системы модульного тестирования на языке программирования Java. Выбор это IDE (Integrated Development Environment) обусловлен ее глубокой

интеграцией с Java и инструментами тестирования. IDE понимает синтаксис Java, управляет classpath, автоматически импортирует зависимости из Maven.

Данная среда включает в себя такие инструменты, как редактор кода с интеллектуальной помощью, который может производить анализ, находить дубликаты и показывать возможные решения ошибок. Инструменты для упрощения и автоматизации кода являются также частью IDE. Встроенный отладчик дает возможность просматривать значения переменных в исходном коде, ставить точки останова, что не менее важно для обнаружения ошибок при написании тестов и приложений.

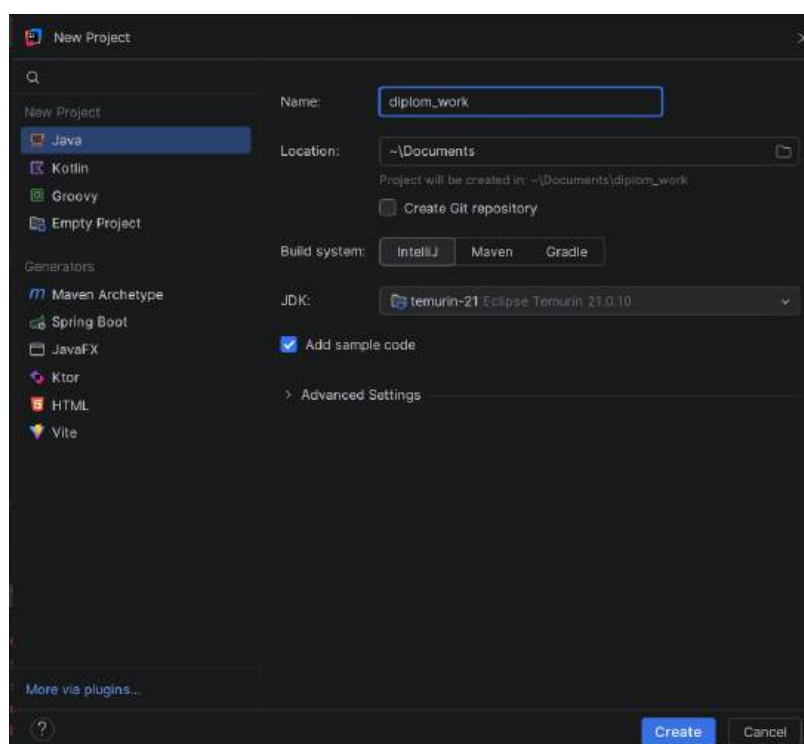


Рисунок 1.6 – Создание проекта в IntelliJ IDEA

IntelliJ IDEA идеально подходит для написания тестов, поскольку предусматривает запуск отдельного тестового метода или всего класса одной кнопкой, после выполнения тестовых сценариев можно по цветовой индикации понять какие тесты были пройдены успешно и какие завершились с ошибками, благодаря быстрой навигации от ошибки в тесте к соответствующей строке кода и древовидному отображению результатов тестирования локализовать и

исправить ошибку становится проще [11]. На рисунке 1.6 показано создание проекта в IntelliJ IDEA.

Интеграция с Git позволяет отслеживать изменения в проекте и упрощает командную разработку.

Настройка среды включала в себя выбор и установку SDK – temurin-21, выбор обусловлен тем, что тестируемые сервисы написаны под Java 21. На рисунке 1.7 показан выбор SDK – temurin-21.

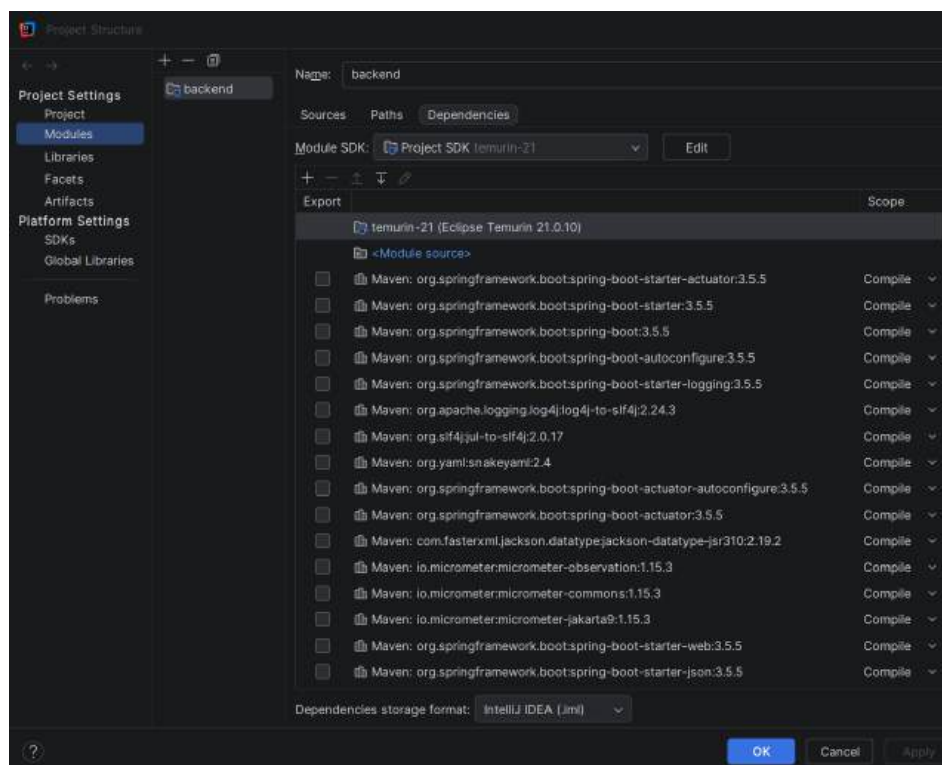


Рисунок 1.7 – Выбор SDK – temurin-21

В файл pom.xml были добавлены строки с описанием необходимых библиотек и зависимостей (например spring-boot-starter-test), и с помощью функции «Reload All Maven Projects» они были импортированы в проект. На рисунке 1.8 показан импорт зависимостей в IntelliJ IDEA.

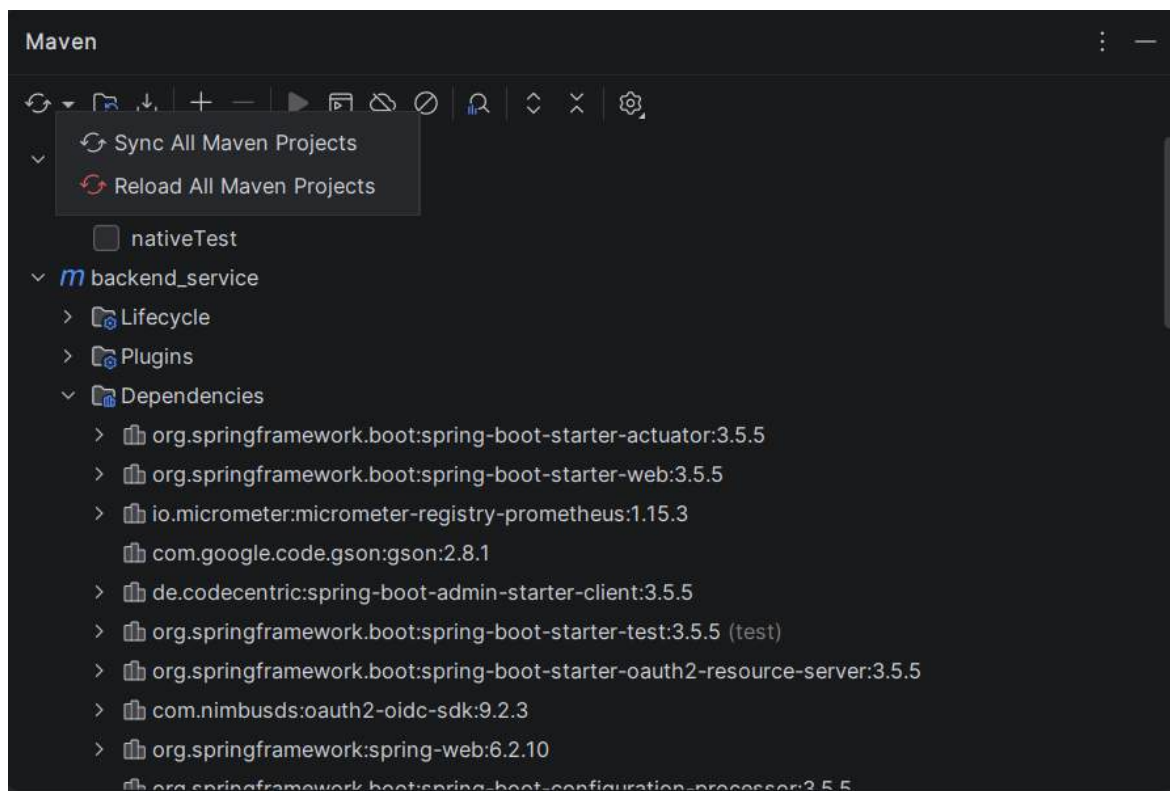


Рисунок 1.8 – Импорт зависимостей в IntelliJ IDEA

1.5.3 Фреймворк для модульного тестирования JUnit 5

JUnit 5 – современный, активно поддерживаемый фреймворк, ставший стандартом для модульного тестирования в Java-экосистеме, он был выбран основным инструментом для написания модульных тестов.

Фреймворк предоставляет возможность группировать тесты по значению с помощью аннотации `@Nested`, это делает тесты более читаемыми и понятными, также аннотация `@DisplayName` позволяет давать тестам различные описания, которые очень важны для наглядности отчетов. В проекте так же используется аннотация `@Test` для обозначения тестового метода, `@BeforeEach` обозначает метод, который происходит перед каждым тестом, `@ExtendWith(MockitoExtension.class)` подключает интеграцию с Mockito. Метод `assertThrows` проверяет, что при выполнении определенного кода выбрасывается ожидаемое исключение – это необходимо для тестирования негативных сценариев и проверки корректной обработки ошибок. Метод `assertDoesNotThrow` проверяет,

что код выполняется без исключений, и используется для позитивных сценариев [12].

1.5.4 Библиотека для создания mock-объектов: Mockito

Для изоляции тестируемых компонентов от внешних зависимостей была выбрана библиотека Mockito Core версии 5.17.0. Mockito – популярная библиотека для создания mock-объектов в Java. Она интегрируется с JUnit 5 с помощью аннотации `@ExtendWith(MockitoExtension.class)`, это позволяет использовать `@Mock` и `@InjectMocks` для внедрения моков в тесты. У Mockito есть функция `verify()`, которая позволяет понять вызвался ли метод нужное количество раз с правильными параметрами.

В проекте используются следующие возможности Mockito. `mock()` создает mock-объект, имитирующий поведение реального объекта. `when().thenReturn()` настраивает возвращаемое значение при вызове метода мока. `when().thenThrow()` настраивает выбрасывание исключения. `doNothing().when()` используется для void-методов, которые не должны ничего делать. `verify()` проверяет, что метод был вызван с определенными аргументами. `spy()` позволяет создать частичный мок реального объекта, когда часть методов вызывается реально, а часть – мокается. `InOrder` проверяет, что методы вызывались в определенной последовательности. Метод `assertThrows` проверяет, что при выполнении определенного кода выбрасывается ожидаемое исключение – это необходимо для тестирования негативных сценариев и проверки корректной обработки ошибок. Метод `assertDoesNotThrow` проверяет, что код выполняется без исключений, и используется для позитивных сценариев [13].

Графический интерфейс: Java Swing

Для реализации графического интерфейса пользователя была выбрана библиотека Java Swing, входящая в стандартную поставку Java. Выбор Swing обоснован следующими соображениями. Swing является встроенной библиотекой и не требует подключения дополнительных зависимостей, что упрощает сборку и распространение приложения. Swing обеспечивает кроссплатформенность –

приложение будет одинаково работать на всех операционных системах, поддерживающих Java. Несмотря на то, что Swing считается несколько устаревшим, его функциональности вполне достаточно для реализации требуемого интерфейса, включающего список тестов с чекбоксами, вкладки, таблицы и текстовые области для вывода логов.

В разработанном приложении используются следующие компоненты Swing. JFrame является главным окном приложения. JPanel служат контейнерами для группировки других компонентов. JButton используются для кнопок «Запустить выбранные тесты», «Очистить логи», «Экспорт отчёта» и «Выход». JTextArea применяются для вывода лога выполнения тестов и итогового отчета. JProgressBar отображает прогресс выполнения тестов. JTable используется для отображения истории запусков. JTabbedPane реализует вкладки «Запуск тестов», «История запусков» и «Статистика». JCheckBox позволяют выбирать отдельные модули для тестирования [14].

1.6 Проектирование системы модульного тестирования

1.6.1 Архитектура системы

Разработанная система модульного тестирования представляет собой программный комплекс, предназначенный для автоматизации процесса тестирования интернет-сервиса зачисления в образовательные организации. Архитектура системы построена по модульному принципу, что обеспечивает гибкость, расширяемость и удобство сопровождения. Модульность позволяет независимо развивать и тестировать каждый компонент, а также заменять или дополнять отдельные части без нарушения целостности всей системы.

В основе архитектуры лежит трехуровневая модель, включающая тестируемые сервисы, модульные тесты и приложение-раннер. Такое разделение соответствует принципу разделения ответственности: каждый уровень решает свои задачи и взаимодействует с соседними через четко определённые интерфейсы. Тестируемые сервисы находятся на нижнем уровне и представляют собой целевые

					ПГУ 1.090301.05.001 ПЗ	Лист
						20
Изм.	Лист	№ докум.	Подпись	Дата		

компоненты интернет-сервиса, подлежащие проверке. Они реализованы на языке Java с использованием фреймворка Spring Boot версии 3.5.5, что обеспечивает масштабируемость, удобство конфигурирования и встроенные механизмы управления зависимостями. Данные сервисы содержат всю бизнес-логику и правила валидации, связанные с приёмом заявлений, формированием приказов, управлением зачётными книжками, экзаменационными группами и другими процессами, характерными для образовательной организации.

Модульные тесты составляют второй уровень архитектуры. Они представляют собой набор тестовых классов и методов, написанных с использованием фреймворка JUnit 5 (Jupiter). Каждый тест проверяет отдельную, строго определённую функциональность тестируемых сервисов. Для изоляции зависимостей в тестах применяется библиотека Mockito, позволяющая создавать мок-объекты и замещать реальные зависимости сервисов (например, репозитории для работы с базой данных, клиенты внешних API, сервисы отправки уведомлений). Благодаря этому достигается высокая скорость выполнения тестов, их воспроизводимость и независимость от внешних факторов, таких как наличие сети или состояние базы данных.

Приложение для запуска тестов (раннер) является управляющим компонентом верхнего уровня. Оно отвечает за обнаружение тестов, программный запуск выбранных тестовых классов, сбор и агрегацию результатов выполнения, формирование итоговых отчётов и ведение истории запусков. Приложение реализовано на Java с использованием библиотеки Swing для построения графического интерфейса. Для взаимодействия с тестами раннер использует официальный программный интерфейс JUnit Platform Launcher, который обеспечивает стандартизированный способ запуска тестов JUnit 5 независимо от среды выполнения. Результаты тестирования сохраняются в текстовом формате, а история всех запусков хранится в файле test_history.txt, что позволяет анализировать динамику качества продукта и выявлять регрессии.

Коммуникация между компонентами организована следующим образом. Пользователь через графический интерфейс выбирает один или несколько тестовых модулей. Раннер, используя JUnit Platform Launcher, инициирует выполнение соответствующих тестов. Тесты, в свою очередь, вызывают методы тестируемых сервисов, подменяя их зависимости моками. Результаты выполнения каждого теста возвращаются раннеру в структурированном виде (статистика и ошибки), после чего раннер агрегирует данные, формирует отчёт и сохраняет информацию в историю. Вся цепочка работает асинхронно: выполнение тестов не блокирует пользовательский интерфейс, что достигается за счёт использования многопоточности. На рисунке 1.9 представлена диаграмма развертывания.

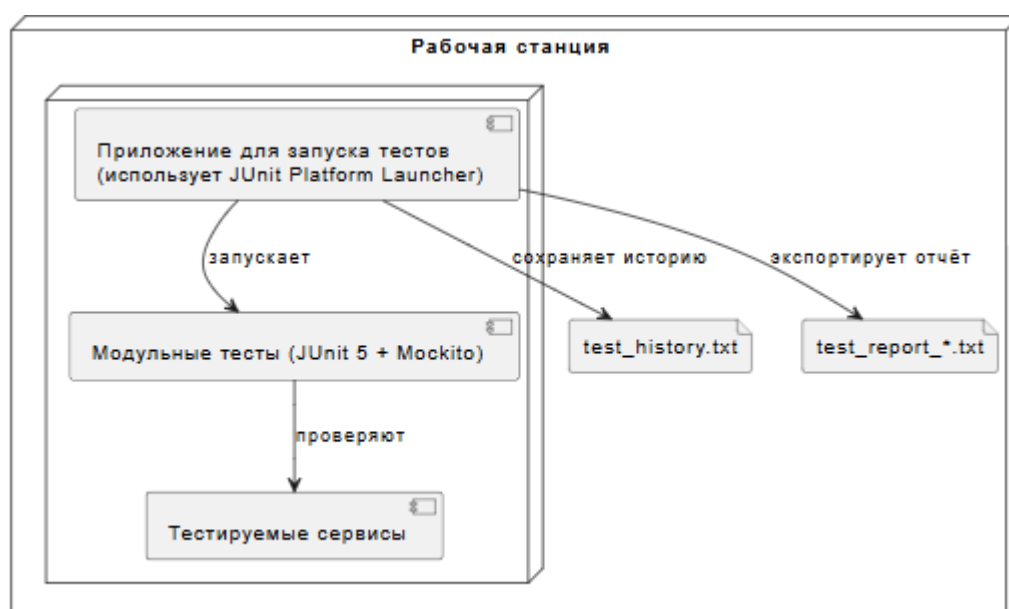


Рисунок 1.9 –Диаграмма развертывания

1.6.2 Взаимодействие компонентов системы

Взаимодействие начинается с того, что пользователь через графический интерфейс приложения для запуска тестов выбирает один или несколько тестовых модулей, отмечая соответствующие флажки. Интерфейс, построенный на Java Swing, отображает список модулей с их названиями и краткими описаниями. После нажатия кнопки запуска приложение блокирует интерфейс на время выполнения, чтобы избежать повторных действий, и формирует список Java-классов,

соответствующих выбранным модулям. Затем для каждого такого класса раннер с помощью JUnit Platform Launcher программно создаёт запрос на выполнение тестов, включая в него все тестовые методы, аннотированные стандартными маркерами JUnit 5. Запуск тестов происходит в отдельном потоке, что не мешает пользователю наблюдать за ходом выполнения через обновляемый индикатор прогресса и лог.

По мере выполнения тестов их результаты (статус прохождения, сообщения об ошибках, время выполнения) собираются специальным слушателем и передаются обратно в приложение. После завершения всех тестов раннер формирует итоговый отчёт, который сразу отображается в графическом интерфейсе в виде текстовой области с детализацией по каждому модулю и сводной статистикой. Кроме того, информация о запуске (дата, количество тестов, число ошибок, процент успешности) сохраняется в файл истории и отображается на отдельной вкладке в табличном виде, снабжённая индикаторами тренда. Таким образом, графический интерфейс выступает не только средством инициализации тестирования, но и центральным узлом визуализации результатов, обеспечивая полный цикл «выбор – запуск – анализ – сохранение». На рисунке 1.10 приведена диаграмма компонентов системы модульного тестирования.

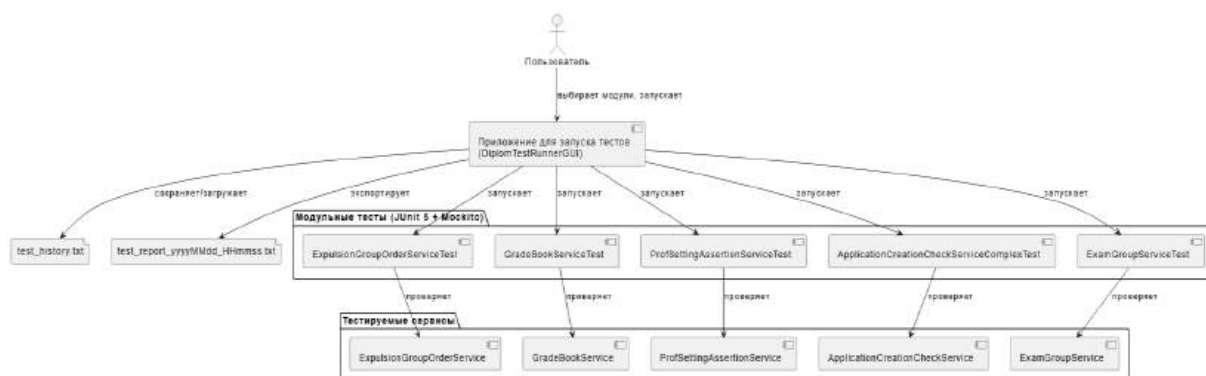


Рисунок 1.10 –Диаграмма компонентов системы модульного тестирования

1.6.3 Реализация модульных тестов

Модульные тесты разработаны с использованием фреймворка JUnit 5 и библиотеки Mockito для изоляции зависимостей. Тесты организованы по принципу «один тестовый класс – один тестируемый сервис», что обеспечивает чёткую структуру и упрощает навигацию по коду.

Каждый тестовый класс создаётся для проверки конкретного сервиса и имеет имя, совпадающее с именем тестируемого класса с суффиксом «Test».

Тестовые классы аннотированы `@ExtendWith(MockitoExtension.class)`, что интегрирует Mockito с JUnit 5 и обеспечивает автоматическую инициализацию мок-объектов, созданных с помощью аннотации `@Mock`. Аннотация `@DisplayName` используется для указания понятного имени тестового класса, которое отображается в отчётах о тестировании. Перед выполнением каждого теста метод `setUp`, аннотированный `@BeforeEach`, инициализирует тестируемый сервис, передавая в его конструктор все необходимые мок-зависимости. Это позволяет каждому тесту выполняться в изолированном окружении с чистыми экземплярами моков. Вспомогательные методы вынесены в отдельные приватные методы, такие как `createApplication()` для создания тестового объекта заявления или `createExamGroupRequest()` для создания запроса экзаменационной группы, что устраняет дублирование кода и упрощает поддержку тестов.

В разработанных тестах используются следующие аннотации JUnit 5. Аннотация `@ExtendWith(MockitoExtension.class)` подключает расширение Mockito для JUnit 5, позволяющее использовать аннотации `@Mock`, `@InjectMocks` и автоматически закрывать моки после выполнения тестов. Аннотация `@DisplayName` задаёт человекочитаемое имя тестового класса или метода; например, `@DisplayName("ApplicationCreationCheckService - unit-тесты")` или `@DisplayName("Не бросает исключение, если тип заявления не ADMISSION")`. Эти имена отображаются в отчётах и упрощают понимание назначения теста. Аннотация `@Nested` используется для группировки связанных тестов во внутренние классы. Группировка улучшает организацию кода и позволяет

выборочно запускать только определённые группы тестов. Аннотация `@Test` указывает, что метод является тестовым. Аннотация `@BeforeEach` обозначает метод, который выполняется перед каждым тестом, и используется для инициализации тестируемого сервиса и настройки начального состояния.

Для изоляции тестируемых сервисов от их реальных зависимостей широко используется библиотека Mockito. Все зависимости тестируемого класса заменяются мок-объектами, создаваемыми с помощью аннотации `@Mock`. Это позволяет тестировать только бизнес-логику сервиса без обращения к базам данных, внешним API или другим реальным компонентам. В каждом тестовом классе все зависимости объявляются с аннотацией `@Mock`.

Поведение моков настраивается с помощью метода `when(...).thenReturn(...)` для возвращаемых значений и `doThrow(...).when(...).check(...)` для выбрасывания исключений. Для проверки того, что методы зависимостей были вызваны с правильными параметрами и в правильном порядке, используются методы `verify(...)` и `InOrder`. Для проверки отсутствия вызовов используется `verify(..., never()).method()`. Для методов, возвращающих `void`, используется `doNothing()` (по умолчанию) или `doThrow()` для имитации ошибки. Для гибкой настройки поведения используются аргументы-матчеры `any()`, `any(Class.class)`, `eq()`, например: `verify(applicationToExamGroupRepo, never()).save(any())`. Таким образом, использование Mockito позволяет полностью изолировать тестируемый сервис, замещая все его внешние зависимости контролируемыми мок-объектами, что обеспечивает высокую скорость выполнения тестов и их независимость от внешних систем.

В рамках дипломной работы разработаны модульные тесты для пяти ключевых сервисов интернет-сервиса зачисления в образовательные организации (заявления, зачетные книжки, настройка функционала, приказы, экзаменационные группы). Общее количество тестовых методов составляет 30, каждый из которых проверяет определённый сценарий работы сервиса.

1.6.4 Модуль тестирования ApplicationCreationCheckServiceComplexTest

Модуль тестирования ApplicationCreationCheckServiceComplexTest предназначен для проверки корректности работы сервиса ApplicationCreationCheckService, который реализует бизнес-логику валидации заявлений на поступление в образовательные организации.

Структура модуля организована с использованием вложенных классов, помеченных аннотацией @Nested. Это позволяет логически сгруппировать тесты по проверяемым методам и сценариям, улучшая читаемость и поддержку кода. Внутри основного класса определены три вложенных класса: CanCreateAdmissionApplicationCheckTest, CheckAdmissionApplicationTest и BoundaryTests. Перед выполнением каждого теста срабатывает метод setUp, аннотированный @BeforeEach, который создаёт реальный экземпляр тестируемого сервиса, передавая в его конструктор все необходимые мок-объекты. Таким образом, каждый тест получает изолированное окружение с контролируемым поведением зависимостей.

Все зависимости сервиса заменены мок-объектами, создаваемыми с помощью аннотации @Mock. В данном тестовом классе замockаны девять компонентов.

Первая группа тестов, CanCreateAdmissionApplicationCheckTest, проверяет метод canCreateAdmissionApplicationCheck, который определяет, разрешено ли создание заявления на поступление (ADMISSION) с учётом системного бана и временных ограничений. Внутри этой группы реализовано четыре тестовых метода. Первый тест, shouldNotThrowWhenTypeIsNotAdmission, создаёт заявление с типом, отличным от ADMISSION (например, FINANCING_CHANGE), и проверяет, что метод не бросает исключение, а также что сервис настроек profSettingService вообще не вызывается – это проверяется с помощью verify(profSettingService, never()).isBanAppAdmission(). Вторым тестом, shouldNotThrowWhenBanIsDisabled, создаёт заявление типа ADMISSION, настраивает мок profSettingService так, что метод isBanAppAdmission() возвращает

false, и убеждается, что исключение не возникает. Третий тест, `shouldThrowWhenBanEnabledAndDateOutOfPeriod`, моделирует ситуацию, когда бан включён (возврат true) и текущая дата находится вне разрешённого периода приёма – мок `isCurrentDateOutOfAppAdmissionDates()` возвращает true. В этом случае ожидается выброс `ApplicationException`, что проверяется через `assertThrows`. Четвёртый тест, `shouldNotThrowWhenBanEnabledButDateInPeriod`, повторяет конфигурацию с включённым баном, но дата находится в пределах периода (false от `isCurrentDateOutOfAppAdmissionDates()`), и метод не должен бросать исключение.

Вторая группа тестов, `CheckAdmissionApplicationTest`, проверяет метод `checkAdmissionApplication`, который последовательно вызывает все восемь проверочных блоков для заявления типа `ADMISSION`. Первый тест, `shouldExecuteAllChecksInCorrectOrder`, создаёт заявление и вызывает метод сервиса. Затем с помощью класса `InOrder` фиксируется порядок вызовов моков: проверяется, что сначала вызывается `skipCheck`, затем `toBasicGeneralProgramBlockCheck`, далее `toPpLevelBlockCheck` и так далее до `hasNpoOrSpoEducationBeforeBlockCheck`. Если порядок нарушен, тест упадёт. Это важно, так как последовательность проверок влияет на корректность бизнес-логики. Второй тест, `shouldStopWhenFirstCheckThrowsException`, настраивает мок `toPpLevelBlockCheck` таким образом, что при вызове его метода `check` выбрасывается `ApplicationException`. После этого проверяется, что вызов `service.checkAdmissionApplication` приводит к тому же исключению. Дополнительно с помощью `verify` проверяется, что `skipCheck` и `toBasicGeneralProgramBlockCheck` были вызваны, а все остальные блоки (`hasAllowedDocsApprovalCheck` и последующие) не вызываются ни разу – для этого используется `verify(..., never())`. Этот тест гарантирует, что после первой же ошибки дальнейшая проверка прекращается.

Третья группа, `BoundaryTests`, содержит один тест `shouldNotThrowNPEWhenApplicationTypeIsNull`, который проверяет граничный

случай: заявление не имеет установленного типа (type = null). Тест убеждается, что метод canCreateAdmissionApplicationCheck не бросает NullPointerException, то есть сервис корректно обрабатывает отсутствие типа. Такая проверка важна для защиты от неожиданных сбоев на реальных данных.

Для удобства в классе определён вспомогательный метод createApplication, который принимает тип заявления и возвращает объект Application с установленным полем type. Этот метод используется во всех тестах, избегая дублирования кода. На рисунке 1.11 приведена диаграмма классов для модуля ApplicationCreationCheckServiceComplexTest

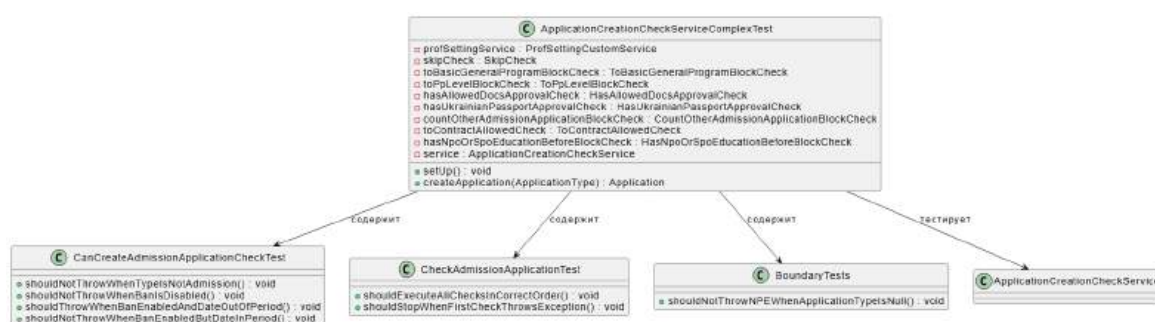


Рисунок 1.11 - Диаграмма классов для модуля ApplicationCreationCheckServiceComplexTest

1.6.5 Модуль тестирования ExpulsionGroupOrderServiceTest

Модуль тестирования ExpulsionGroupOrderServiceTest предназначен для проверки корректности работы сервиса ExpulsionGroupOrderService, который реализует бизнес-логику управления групповыми приказами об отчислении студентов в интернет-сервисе зачисления в образовательные организации.

Все зависимости тестируемого сервиса заменены мок-объектами, создаваемыми с помощью аннотации @Mock. В данном классе замockаны пять компонентов. Сам тестируемый сервис ExpulsionGroupOrderService создаётся с помощью аннотации @InjectMocks, которая автоматически внедряет все перечисленные моки в экземпляр сервиса. Благодаря этому каждый тест работает с реальным объектом сервиса, но все его зависимости полностью контролируются через моки.

Перед выполнением тестов в методе setUp, аннотированном @BeforeEach, создаются общие мок-объекты mockOrder (сущность приказа) и mockParams (параметры запроса) с помощью вызова mock(StudOrder.class) и mock(ExpulsionGroupOrderParams.class). Никакие стабы в этом методе не настраиваются, чтобы избежать появления лишних заглушек, которые могут вызвать UnnecessaryStubbingException. Все необходимые поведения задаются непосредственно внутри каждого тестового метода.

В классе также определена константа ID = 123124124L, используемая во всех тестах как идентификатор приказа, что обеспечивает единообразие тестовых данных.

Структура модуля организована с помощью вложенных классов, помеченных аннотацией @Nested. Это позволяет логически сгруппировать тесты по проверяемым операциям: получение, создание, обновление и удаление приказа.

Первая группа, GetOrderTest, содержит один тест shouldGetOrderSuccessfully, который проверяет метод getOrder, получающий информацию о групповом приказе об отчислении по его идентификатору. В начале теста настраивается поведение мока mockOrder: вызов getOrderItems() возвращает пустой список. Затем создаётся мок-объект ExpulsionGroupOrderContentResponse, который будет имитировать ожидаемый ответ. Далее настраиваются моки зависимостей: orderService.findByIdAndType(ID, OrderType.EXPULSION_GROUP) возвращает mockOrder; orderConvertService.toExpulsionGroupOrderContentResponse(mockOrder, new ArrayList<>()) возвращает мок ответа; securityUtilService.securityCheckOrder(mockOrder) ничего не делает (doNothing()). После этого вызывается реальный метод expulsionGroupOrderService.getOrder(ID). Проверки осуществляются с помощью assertNotNull (результат не равен null) и assertEquals (фактический результат соответствует ожидаемому моку). Дополнительно с помощью verify(orderConvertService).toExpulsionGroupOrderContentResponse(mockOrder, new

ArrayList<>()) проверяется, что метод конвертации был вызван ровно один раз с правильными аргументами.

Вторая группа, CreateOrderTest, содержит тест shouldCreateOrderSuccessfully, проверяющий создание нового группового приказа. Создаётся мок-объект ExpulsionGroupOrderResult, имитирующий результат операции, и настраивается мок orderTransactionHandler.runInTransaction(any()) так, чтобы он возвращал этот мок. Затем вызывается метод expulsionGroupOrderService.createOrder(mockParams). Проверяется, что результат не null и соответствует ожидаемому моку, а также что orderTransactionHandler.runInTransaction(any()) был вызван ровно один раз. Этот тест гарантирует, что операция создания приказа выполняется внутри транзакции, а сам вызов сервиса делегирует работу обработчику транзакций.

Третья группа, UpdateOrderTest, содержит тест shouldUpdateOrderSuccessfully, проверяющий обновление существующего приказа. Настраивается orderService.findOrderByIdAndType(ID, OrderType.EXPULSION_GROUP), чтобы возвращать mockOrder а orderTransactionHandler.runInTransaction(any()) – возвращать мок-объект ExpulsionGroupOrderResult. После вызова метода expulsionGroupOrderService.updateOrder(ID, mockParams) проверяется, что результат не null и соответствует ожидаемому. С помощью verify(orderAssertService).assertOrderStatusForUpdate(mockOrder) убеждаемся, что перед обновлением была выполнена проверка допустимости статуса приказа для операции обновления. Также проверяется вызов orderTransactionHandler.runInTransaction(any()). Этот тест подтверждает, что обновление приказа защищено проверкой статуса и также выполняется в транзакции.

Четвёртая группа, DeleteOrderTest, содержит тест shouldDeleteOrderSuccessfully, проверяющий удаление приказа. Настраивается orderService.findOrderByIdAndType(ID, OrderType.EXPULSION_GROUP), чтобы возвращать mockOrder, после чего вызывается метод

					ПГУ 1.090301.05.001 ПЗ	Лист
						30
Изм.	Лист	№ докум.	Подпись	Дата		

`expulsionGroupOrderService.deleteOrder(ID)`. Поскольку метод не возвращает значения, проверка ограничивается вызовом `assertDoesNotThrow`, который убеждает, что исключение не возникает. Затем с помощью `verify(orderAssertService).assertOrderStatusForDelete(mockOrder)` проверяется, что перед удалением была выполнена проверка допустимости статуса для операции удаления, и `verify(orderTransactionHandler).runInTransaction(any())` – что удаление выполняется в транзакции.

Таким образом, модуль `ExpulsionGroupOrderServiceTest` полностью покрывает все основные CRUD-операции над групповыми приказами об отчислении: получение, создание, обновление и удаление. Каждый тест изолирован благодаря мокам, не требует реальной базы данных или внешних сервисов, выполняется быстро и надёжно. Использование вложенных классов `@Nested` и понятных имён методов делает структуру тестов чистой и удобной для поддержки. На рисунке 1.12 приведена диаграмма классов тестового модуля `ExpulsionGroupOrderServiceTest`.

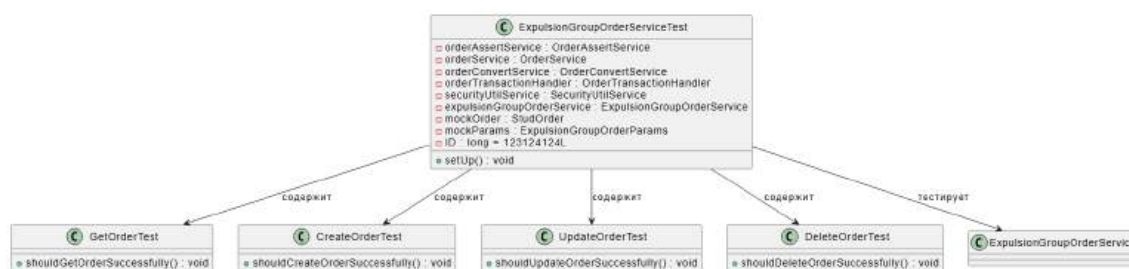


Рисунок 1.12 – Диаграмма классов тестового модуля
`ExpulsionGroupOrderServiceTest`

1.6.6 Модуль тестирования ExamGroupServiceTest

Модуль тестирования ExamGroupServiceTest предназначен для проверки корректности работы сервиса ExamGroupService, который реализует бизнес-логику управления экзаменационными группами в интернет-сервисе зачисления в образовательные организации. Все зависимости тестируемого сервиса заменены мок-объектами, создаваемыми с помощью аннотации Mock. В данном классе замockаны двенадцать компонентов. Сам тестируемый сервис ExamGroupService создаётся в методе setUp, который выполняется перед каждым тестом, где явно вызывается конструктор с передачей всех перечисленных моков. Это позволяет каждому тесту работать с реальным экземпляром сервиса, но все его внешние связи полностью контролируются через моки.

В классе определены константы: EXAM_GROUP_ID равный 1L, GROUP_NAME и EXAM_DATE (текущая дата плюс семь дней), которые используются для формирования тестовых данных. Также присутствует вспомогательный метод createExamGroupRequest, создающий стандартный запрос на создание экзаменационной группы с заполненными полями groupName, examDateTime, addressAreaId и examRoom. Этот метод помогает избежать дублирования кода и обеспечивает единообразие данных в тестах, где требуется объект ExamGroupReq.

Структура модуля организована с помощью четырёх вложенных классов, каждый из которых отвечает за отдельную функциональность сервиса: получение группы по идентификатору, удаление экзаменационной группы, проверка возможности закрытия группы и выставление оценок студентам.

Первая группа, GetByIdTest, содержит два тестовых метода. Первый метод проверяет успешное получение экзаменационной группы по её идентификатору. В начале теста создаются моки сущности ExamGroup и ожидаемого ответа ExamGroupResp. Затем настраивается поведение репозитория: вызов findById с аргументом EXAM_GROUP_ID возвращает Optional с моком группы, а вызов конвертера examGroupToRespConverter.convert с этой группой возвращает мок

ответа. После вызова метода `getExamGroup` сервиса проверяется, что результат не равен `null`, что он совпадает с ожидаемым моком, и что метод конвертера был вызван ровно один раз. Второй тест этой группы проверяет ситуацию, когда группа с указанным идентификатором отсутствует: репозиторий возвращает `Optional.empty`, и вызов сервиса должен выбросить исключение `ExamGroupNotFoundException`, что подтверждается с помощью `assertThrows`.

Вторая группа, `DeleteExamGroupTest`, включает три теста, покрывающих различные сценарии удаления. Первый тест моделирует успешное удаление группы. Для этого создаётся мок `ExamGroup`, у которого метод `getStatus` возвращает `INITIAL`, а репозиторий `findById` возвращает `Optional` с этим моком. Также настраивается, что счётчик `applicationToExamGroupRepo.countAllByExamGroup` возвращает `0`, то есть в группе нет студентов. Удаление не должно выбросить исключение, что проверяется через `assertDoesNotThrow`, и затем `verify` проверяет, что метод `delete` репозитория был вызван ровно один раз. Второй тест проверяет, что удаление невозможно, если группа не находится в статусе `INITIAL`. Здесь статус мока возвращает `COMPLETE`, и при вызове `deleteExamGroup` ожидается выброс `ApplicationException`. Дополнительно `verify` с `never` подтверждает, что метод `delete` не вызывался. Третий тест аналогично проверяет, что удаление запрещено, если в группе есть студенты: статус `INITIAL`, но счётчик `countAllByExamGroup` возвращает `5`, и снова ожидается исключение `ApplicationException`, а `delete` не вызывается.

Третья группа, `CanCloseExamGroupTest`, содержит два теста для проверки возможности закрытия экзаменационной группы. Первый тест проверяет случай, когда группу можно закрыть. Создаётся мок `ExamGroup`, у которого статус `INITIAL`, дата экзамена установлена на вчерашний день (`LocalDateTime.now().minusDays(1)`), а репозиторий `findById` возвращает `Optional` с этим моком. Также настраивается, что вызов `countAllByExamGroupAndMarkIsNull` возвращает `0`, то есть все студенты получили оценки. Метод `canCloseExamGroup` возвращает объект `CanDoResp`, у которого `isAllowed` должно быть `true`. Тест

проверяет, что результат не null и что isAllowed истинно. Второй тест проверяет случай, когда дата экзамена ещё не наступила (завтрашний день), тогда isAllowed должно быть false. Остальные условия аналогичны, но проверяется отрицательный результат.

Четвёртая группа, SetExamGroupApplicationMarkTest, содержит два теста для выставления оценки студенту. Первый тест проверяет успешное выставление оценки. Создаётся объект запроса ApplicationExamMarkReq, у которого устанавливается applicationToExamGroupId равный 10L и mark равный 1. Мокируется репозиторий applicationToExamGroupRepo: вызов findById с идентификатором 10L возвращает Optional с моком объекта ApplicationToExamGroup. Вызов save возвращает тот же мок. Тест проверяет, что вызов setExamGroupApplicationMark не бросает исключений, а затем последовательно проверяются вызовы на моке: setMark с аргументом 1, setMarkedByCurrentUser, setMarkedAt с любым объектом LocalDateTime, и save на репозитории. Второй тест проверяет ситуацию, когда запись о привязке студента к группе не найдена: findById возвращает Optional.empty, и ожидается выброс ApplicationException, при этом save никогда не вызывается.

Таким образом, модуль ExamGroupServiceTest покрывает основные сценарии работы сервиса экзаменационных групп: получение информации о группе, удаление с проверкой статуса и наличия студентов, определение возможности закрытия группы в зависимости от даты экзамена и наличия оценок, а также выставление оценок студентам с обработкой случая отсутствия записи. Каждый тест изолирован благодаря использованию моков, не требует реальной базы данных и выполняется быстро. На рисунке 1.13 показана диаграмма классов тестового модуля ExamGroupServiceTest.

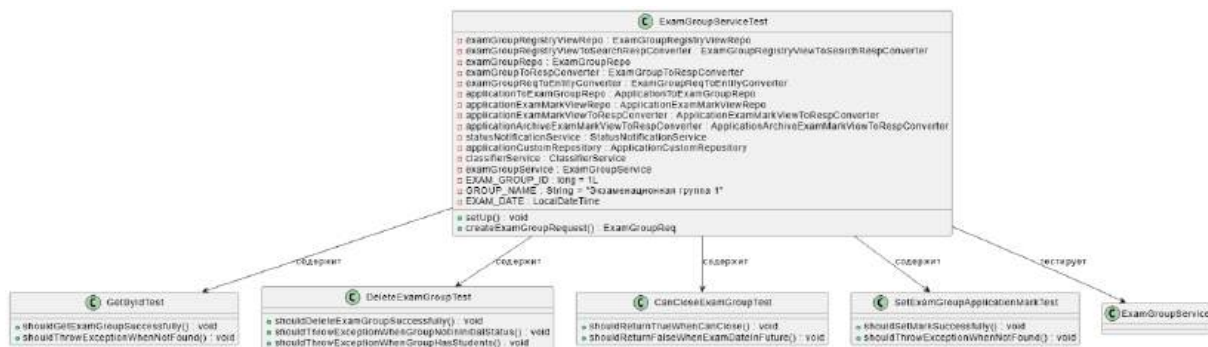


Рисунок 1.13 – Диаграмма классов тестового модуля ExamGroupServiceTest

1.6.7 Модуль тестирования GradeBookServiceTest

Модуль тестирования GradeBookServiceTest предназначен для проверки корректности работы сервиса GradeBookService, который реализует бизнес-логику управления зачётными книжками студентов в интернет-сервисе зачисления в образовательные организации. Все зависимости тестируемого сервиса заменены мок-объектами, создаваемыми с помощью аннотации Mock. В данном классе замockаны семь компонентов. Сам тестируемый сервис GradeBookService создаётся в методе setUp, который выполняется перед каждым тестом, где вызывается конструктор с передачей всех перечисленных моков. Это обеспечивает полную изоляцию тестируемого сервиса от реальных баз данных и внешних API.

В классе определены константы: STUDENT_ID равный 1L, GRADE_BOOK_NUM равный "123456" и GRADE_BOOK_DATE равный 15 января 2026 года. Эти значения используются во всех тестах для единообразия данных. Вспомогательные методы отсутствуют, однако в тестах создаются объекты запросов StudGradeBookReq с использованием указанных констант. С помощью аннотации Nested тесты сгруппированы в пять вложенных классов, каждый из которых проверяет отдельную операцию или группу сценариев: получение зачётной книжки, добавление, обновление, удаление, а также граничные случаи и ошибки.

Первая группа, GetGradeBookTest, содержит один тестовый метод, проверяющий успешное получение зачётной книжки по идентификатору студента. В этом тесте создаются моки студента и ожидаемого ответа StudGradeBookResp.

Настраивается поведение репозитория: вызов `getStudentById` со `STUDENT_ID` возвращает мок студента, конвертер `studentToStudGradeBookRespConverter.convert` с этим студентом возвращает мок ответа, а сервис проверки настроек `assertAllowFunctionalGradeBook` ничего не делает. После вызова метода `getGradeBook` сервиса проверяется, что результат не равен `null` и соответствует ожидаемому ответу. Затем с помощью `verify` последовательно проверяется вызов `assertAllowFunctionalGradeBook`, `getStudentById` и `convert`. Тест демонстрирует, что успешное получение зачётной книжки проходит все необходимые проверки и преобразования.

Вторая группа, `AddGradeBookTest`, содержит один тест, проверяющий добавление новой зачётной книжки студенту. В начале теста создаётся мок студента и его подчинённого объекта `StudentEducation`, причём вызов `getEducation` возвращает мок образования. Затем создаётся объект запроса `StudGradeBookReq` с номером и датой зачётной книжки и мок ожидаемого ответа. Настраиваются моки: репозиторий возвращает студента, конвертер возвращает ответ, сервис проверки настроек и сервис проверки отсутствия зачётной книжки (`assertNoGradeBook`) ничего не делают, репозиторий при сохранении возвращает того же студента, сервис проверки студента `assertHasMeshId` ничего не делает. Для синхронизации с внешней системой мокируются вызовы `getCurrentPersonEducationSync`, `getPersonEducations` и `updatePersonEducation`. После вызова метода `addGradeBook` проверяется, что результат не `null` и соответствует ожидаемому, а также что сервис `assertNoGradeBook` был вызван и репозиторий сохранил студента как минимум один раз. Тест охватывает не только локальное добавление зачётной книжки, но и последующую синхронизацию данных с внешним сервисом через `ContingentEducationApi`.

Третья группа, `UpdateGradeBookTest`, содержит один тест, проверяющий обновление существующей зачётной книжки. По структуре он почти идентичен тесту добавления, но вместо `assertNoGradeBook` используется `assertHasGradeBook`, который проверяет, что зачётная книжка уже существует. Также после вызова

метода `updateGradeBook` дополнительно проверяется, что `assertHasGradeBook` был вызван. Все остальные настройки моков и проверки аналогичны добавлению, что говорит о схожести бизнес-логики, но с разными условиями.

Четвёртая группа, `DeleteGradeBookTest`, содержит один тест, проверяющий удаление зачётной книжки. Здесь создаётся мок студента с образованием, настраивается возврат студента репозиторием, сервис проверки настроек ничего не делает, при сохранении возвращается студент, выполняется проверка `assertHasMeshId`. Также настраиваются моки синхронизации с внешней системой. После вызова метода `deleteGradeBook` используется `assertDoesNotThrow`, чтобы убедиться, что удаление не сопровождается исключениями, и проверяется, что студент был сохранён в репозитории (поскольку удаление зачётной книжки предполагает изменение сущности студента). Тест не проверяет вызов каких-либо специфических `assert`-сервисов, связанных с зачётной книжкой, потому что основная логика удаления, вероятно, выполняется внутри самого сервиса через установку `null` в поле `education` студента.

Пятая группа, `BoundaryAndErrorTest`, содержит два теста, проверяющих обработку ошибок. Первый тест проверяет, что при отсутствии студента (репозиторий выбрасывает `NotFoundException`) метод `addGradeBook` также выбрасывает это исключение, и оно не перехватывается внутри сервиса. Вторым тестом проверяется, что при обновлении зачётной книжки, если студент существует, но у него нет зачётной книжки (сервис `assertHasGradeBook` выбрасывает `RuntimeException`), то вызов `updateGradeBook` также выбрасывает исключение. Оба теста демонстрируют корректную обработку ошибок и проброс исключений из вспомогательных сервисов наружу.

Таким образом, модуль `GradeBookServiceTest` покрывает все основные операции управления зачётными книжками: получение, добавление, обновление и удаление, а также два важных граничных случая (отсутствие студента и отсутствие зачётной книжки при обновлении). Все зависимости изолированы с помощью моков, тесты не требуют реальной базы данных, выполняются быстро и позволяют

надёжно проверять бизнес-логику сервиса в различных сценариях. На рисунке 1.14 показана диаграмма классов тестового модуля `GradeBookServiceTest`



Рисунок 1.14 – Диаграмма классов тестового модуля GradeBookServiceTest

1.6.8 Модуль тестирования ProfSettingAssertionServiceTest

Модуль тестирования ProfSettingAssertionServiceTest предназначен для проверки корректности работы сервиса ProfSettingAssertionService, который отвечает за валидацию включения или выключения определённых функциональных возможностей интернет-сервиса зачисления в образовательные организации, а именно возможности запуска межведомственных проверок и функционала зачётной книжки. Единственная зависимость тестируемого сервиса – ProfSettingCustomService – заменена мок-объектом с помощью аннотации Mock. Сам тестируемый сервис ProfSettingAssertionService создаётся в методе setUp, который выполняется перед каждым тестом, путём вызова конструктора с передачей мока. Это обеспечивает полную изоляцию тестируемой логики от реальной реализации сервиса настроек.

Структура модуля организована с помощью двух вложенных классов, аннотированных `Nested`. Первая группа, `AssertAgencyEnabledTest`, проверяет метод `assertAgencyEnabled`, который выбрасывает исключение, если функционал межведомственных проверок выключен. Внутри этой группы содержатся два теста. Первый тест проверяет сценарий, когда функционал включён: мок сервиса настроек возвращает `false` при вызове `isSettingFalse` с константой `AGENCY_ENABLED`, и вызов метода `assertAgencyEnabled` не должен бросать исключение, что проверяется с помощью `assertDoesNotThrow`. Дополнительно `verify` убеждается, что метод `isSettingFalse` был вызван. Второй тест проверяет

противоположный сценарий, когда функционал выключен: `isSettingFalse` возвращает `true`, и ожидается выброс `ApplicationException`. С помощью `assertThrows` перехватывается исключение, после чего проверяется, что текст сообщения совпадает с ожидаемой строкой «Функционал запуска межведомственных проверок выключен». Также `verify` подтверждает вызов метода `isSettingFalse`.

Вторая группа, `AssertAllowFunctionalGradeBookTest`, проверяет метод `assertAllowFunctionalGradeBook`, который контролирует доступность функционала зачётной книжки. Аналогично первой группе, здесь также два теста. Первый тест моделирует ситуацию, когда зачётная книжка включена: `isSettingFalse` с константой `ALLOW_FUNCTIONAL_GRADE_BOOK` возвращает `false`, `assertDoesNotThrow` подтверждает отсутствие исключения, `verify` фиксирует вызов. Второй тест проверяет выключенное состояние: `isSettingFalse` возвращает `true`, `assertThrows` перехватывает `ApplicationException`, сообщение исключения должно равняться «Функционал зачетной книжки выключен», и снова `verify` проверяет вызов метода `isSettingFalse`.

Таким образом, модуль `ProfSettingAssertionServiceTest` полностью покрывает оба метода сервиса `ProfSettingAssertionService`, проверяя как положительные (функционал включён, исключения нет), так и отрицательные (функционал выключен, исключение с правильным сообщением) сценарии. Благодаря использованию моков тесты не зависят от реальной конфигурации настроек, выполняются быстро и позволяют надёжно убедиться в правильности логики выбрасывания исключений в зависимости от состояния функциональных флагов. На рисунке 1.15 показана диаграмма классов тестового модуля `ProfSettingAssertionServiceTest`.

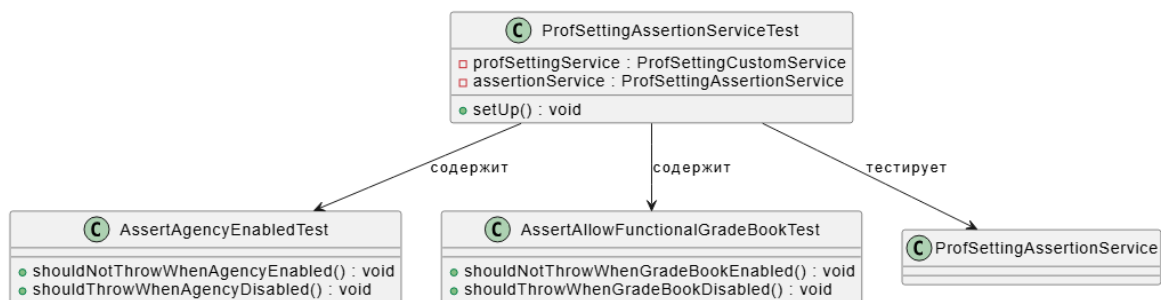


Рисунок 1.15 – Диаграмма классов тестового модуля
ProfSettingAssertionServiceTest

1.6.9 Реализация приложения для запуска модульных тестов

В данном разделе описывается реализация приложения для запуска тестов, которое обеспечивает графический интерфейс для выбора и запуска модульных тестов, а также отображение и сохранение результатов. Приложение написано на Java с использованием библиотеки Swing для пользовательского интерфейса и JUnit Platform Launcher для программного запуска тестов. На рисунке 1.16 показана диаграмма последовательности для сценария запуска тестов.

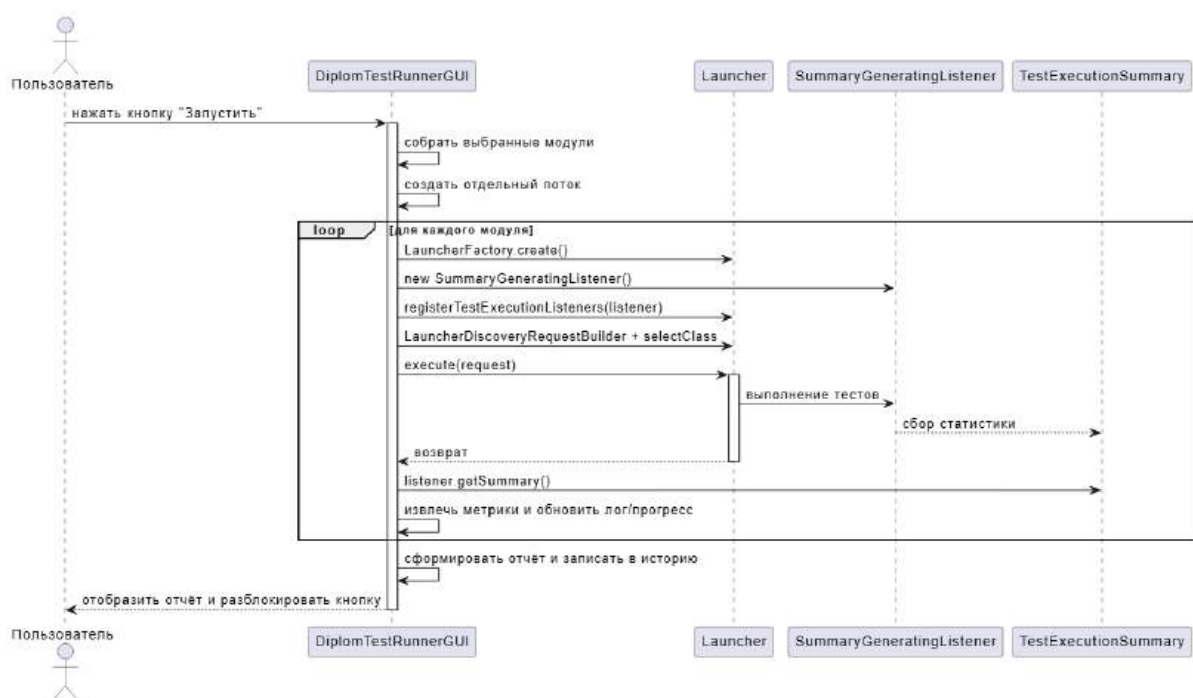


Рисунок 1.16 – Диаграмма последовательности для запуска тестов

Приложение-раннер реализовано в единственном классе DiplomaTestRunnerGUI, который наследуется от JFrame и содержит поля для хранения компонентов интерфейса, списков тестовых модулей, истории запусков, а также константы. Все операции по управлению тестированием, обработке результатов и ведению истории сосредоточены в методах этого класса.

Конструктор и инициализация. При создании экземпляра класса вызывается конструктор DiplomaTestRunnerGUI. В нём устанавливаются размеры окна, его расположение по центру экрана, задаётся операция закрытия. Сначала вызывается метод loadHistory, который загружает историю предыдущих запусков из файла. Затем создаются панели интерфейса с помощью методов createHeaderPanel, createTestRunnerPanel, createHistoryPanel, createStatisticsPanel, createButtonPanel. Эти методы строят графический интерфейс, но их детальное описание выходит за рамки данного раздела, поскольку они касаются исключительно визуального представления. После сборки интерфейса вызывается setLookAndFeel, который пытается установить системную тему оформления, и updateHistoryTable, заполняющая таблицу истории данными из загруженного списка.

Методы работы с историей. Метод loadHistory отвечает за загрузку записей о предыдущих запусках из текстового файла test_history.txt. Он проверяет существование файла, и если тот присутствует, читает все строки с помощью Files.readAllLines. Каждая строка разбивается по символу вертикальной черты, и из полученных семи частей извлекаются идентификатор запуска (целое число), временная метка (строка), общее количество тестов, количество пройденных, упавших, пропущенных (целые числа) и процент успешности (число с плавающей точкой). На основе этих данных создаётся объект внутреннего класса TestRunRecord, который добавляется в список testRunHistory. Если файла нет, список остаётся пустым. В случае ошибок ввода-вывода список также инициализируется пустым, а сообщение об ошибке выводится в стандартный поток ошибок.

Метод `saveHistory` выполняет обратную операцию: он открывает файл `test_history.txt` на запись (используя `FileWriter` в блоке `try-with-resources`) и для каждого объекта `TestRunRecord` из списка формирует строку, в которой поля разделены вертикальной чертой, и записывает её в файл. Порядок полей соответствует порядку при загрузке. Этот метод вызывается после каждого успешного завершения тестов, а также при очистке истории.

Метод `clearHistory` предназначен для удаления всей истории. Он показывает диалог подтверждения с помощью `JOptionPane.showConfirmDialog`, и при положительном ответе очищает список `testRunHistory`, вызывает `saveHistory` (что приводит к перезаписи файла пустым содержимым), обновляет таблицу истории вызовом `updateHistoryTable`, пересчитывает статистику через `updateStatistics(null)` и добавляет в лог сообщение об очистке.

Метод `updateHistoryTable` обновляет содержимое таблицы `historyTable`. Он выполняется в потоке обработки событий с помощью `SwingUtilities.invokeLater`, чтобы гарантировать корректное обновление интерфейса. Сначала модель таблицы очищается вызовом `setRowCount(0)`. Затем список `testRunHistory` перебирается в обратном порядке (чтобы последние запуски были вверху). Для каждой записи формируется строка, содержащая идентификатор, отформатированную дату и время, общее количество тестов, пройденные, ошибки, процент успешности со знаком процента и символ тренда. Тренд вычисляется сравнением успешности текущей записи с предыдущей: стрелка вверх при увеличении, стрелка вниз при уменьшении, горизонтальная стрелка при неизменной успешности; для первой записи ставится дефис. Строка добавляется в модель, после чего таблица отображает новые данные.

Методы экспорта и генерации данных. Метод `exportReportToTXT` экспортирует текущий итоговый отчёт в текстовый файл. Он проверяет, что текстовая область `reportArea` не пуста; если пуста – выводит предупреждение. Иначе создаёт экземпляр `JFileChooser` с предложением имени файла по шаблону `test_report_ГГГГММДД_ЧЧММСС.txt`. После выбора пользователем файла

содержимое reportArea записывается в файл через FileWriter. При возникновении ошибок выводится соответствующее сообщение.

Метод exportHistoryToTXT экспортирует всю историю запусков. Он проверяет, что список testRunHistory не пуст, затем предлагает сохранить файл с именем по шаблону test_history_ГГГГММДД_ЧЧММСС.txt. Для формирования содержимого используется вспомогательный метод generateHistoryTXT, который строит многострочную строку. В этой строке записи перебираются в обратном порядке, для каждой выводятся номер запуска, дата и время, количество тестов, пройденные, ошибки, пропущенные, успешность, а также текстовое описание тренда (Улучшение, Ухудшение, Стабильно или прочерк). В конце добавляется общее количество запусков. Полученная строка записывается в выбранный файл.

Метод обновления статистики – updateStatistics(JTextArea statsArea). Он вычисляет агрегированные показатели на основе всего списка testRunHistory. Если список пуст, метод завершается. Для расчётов используется потоковый API: средняя успешность вычисляется как среднее арифметическое через mapToDouble().average(), лучший и худший результаты – через max() и min() с компаратором по полю successRate, общее количество выполненных тестов и пройденных – через mapToInt().sum(). Все значения форматируются в виде многострочной строки с выравниванием. Если переданный параметр statsArea не равен null, текст устанавливается в эту текстовую область (с синхронизацией через SwingUtilities.invokeLater). Этот метод вызывается после каждого тестового запуска, после очистки истории, а также вручную по нажатию кнопки «Обновить статистику».

Вспомогательные методы для лога. Метод appendLog(String message) добавляет сообщение в лог. Он выполняется в потоке обработки событий, добавляет строку в logArea и перемещает курсор в конец документа с помощью setCaretPosition, что обеспечивает автоматическую прокрутку. Метод clearLogs очищает содержимое logArea и reportArea, сбрасывает метку статуса, счётчик

выбранных модулей и индикатор прогресса, а затем записывает в лог сообщение о готовности к запуску.

Основной метод запуска тестов – `runSelectedTests`. Это наиболее сложный метод, выполняющий всю логику обнаружения тестов, их выполнения, сбора результатов и формирования отчёта. При вызове метод сначала собирает список выбранных тестовых модулей, проходя по всем флажкам из списка `testCheckboxes` и добавляя соответствующие объекты `TestInfo` из `testInfos` в новый список `selectedTests`. Если список пуст, выводится предупреждение с помощью `JOptionPane` и метод завершается.

Затем происходит блокировка кнопки запуска, сброс индикатора прогресса, обновление меток статуса и счётчика выбранных модулей. В лог выводятся стартовые сообщения с текущим временем. Далее создаётся новый поток `Thread`, в котором выполняется основная работа – это нужно, чтобы графический интерфейс не «зависал» на время выполнения тестов.

Внутри потока инициализируются переменные-аккумуляторы: `totalTests`, `totalPassed`, `totalFailed`, `totalSkipped` и флаг `hasErrors`. Создаётся строковый построитель `fullReport` для итогового отчёта, в который сразу добавляются заголовок и дата запуска.

Затем метод последовательно перебирает все выбранные модули. Для каждого модуля в лог записываются его имя и описание. В блоке `try` выполняется следующее: через `Class.forName(info.className)` загружается тестовый класс по полному имени. Создаётся экземпляр `Launcher` с помощью статического метода `LauncherFactory.create`. Создаётся объект `SummaryGeneratingListener` – это слушатель, который собирает статистику выполнения тестов, и регистрируется в лаунчере. С помощью `LauncherDiscoveryRequestBuilder.request` и метода `selectors(selectClass(...))` формируется запрос `LauncherDiscoveryRequest`, указывающий, какой класс нужно выполнить. Запрос передаётся в метод `launcher.execute(request)`, который запускает все тесты указанного класса в среде `JUnit Platform`.

После выполнения вызывается `listener.getSummary`, возвращающий объект `TestExecutionSummary`. Из него извлекаются количество найденных тестов (`getTestsFoundCount`), успешно пройденных (`getTestsSucceededCount`), упавших (`getTestsFailedCount`) и пропущенных (`getTestsSkippedCount`). Эти значения прибавляются к общим переменным. Если количество упавших тестов больше нуля, флаг `hasErrors` устанавливается в `true`. В лог выводится результат в формате «пройдено / всего». Если есть упавшие тесты, метод `getFailures` возвращает список ошибок, каждая из которых содержит идентификатор теста (`getTestIdentifier().getDisplayName`) и исключение (`getException().getMessage`). Эта информация добавляется в отчёт: для модуля выводится строка с символом X, затем сообщения об ошибках. Если упавших тестов нет, в отчёт добавляется строка «ОК название модуля - УСПЕШНО».

Если при загрузке класса возникает `ClassNotFoundException`, в лог выводится ошибка, переменная `totalFailed` увеличивается, флаг `hasErrors` устанавливается, а в отчёт добавляется строка о том, что класс не найден. Аналогично обрабатываются любые другие исключения (`Exception`).

После обработки каждого модуля вычисляется процент выполнения как отношение текущего индекса ($i+1$) к общему числу выбранных модулей. С помощью `SwingUtilities.invokeLater` обновляется значение и текст индикатора прогресса.

Когда все модули обработаны, в лог выводится сообщение о завершении. В отчёт добавляется блок «СТАТИСТИКА» с общими количествами тестов, пройденных, упавших, пропущенных, а также целочисленным процентом успешности (если общее число тестов больше нуля, иначе 0). После этого создаётся объект `TestRunRecord` с полями: идентификатор (текущий размер списка истории + 1), временная метка, накопленные значения и процент успешности. Запись добавляется в список `testRunHistory`. Вызывается `saveHistory`, затем `updateHistoryTable` для обновления таблицы и `updateStatistics(null)` для пересчёта агрегированных показателей.

Наконец, снова через `SwingUtilities.invokeLater` текст итогового отчёта устанавливается в `reportArea`, обновляется статусная метка (в зависимости от наличия ошибок и количества тестов), разблокируется кнопка запуска, и на индикаторе прогресса отображается текст «Готово». Все операции с компонентами интерфейса из рабочего потока выполняются через `invokeLater`, чтобы избежать проблем многопоточности.

Вспомогательные внутренние классы. Класс `TestInfo` служит контейнером для описания тестового модуля: содержит поля `name`, `description`, `className` и статический фабричный метод `from`. Класс `TestRunRecord` хранит данные об одном запуске: `id`, `timestamp` (в виде строки ISO), `totalTests`, `passed`, `failed`, `skipped`, `successRate`. Конструктор преобразует `LocalDateTime` в строку, а метод `getTimestamp` выполняет обратное преобразование для удобного форматирования.

Точка входа. Метод `main` запускает приложение через `SwingUtilities.invokeLater`, что гарантирует создание и отображение окна в потоке обработки событий. В нём создаётся экземпляр `DiplomaTestRunnerGUI` и вызывается `setVisible(true)`.

Таким образом, класс `DiplomaTestRunnerGUI` реализует полный цикл работы приложения: загрузка истории, выбор модулей, параллельный запуск тестов через `JUnit Platform Launcher`, сбор и агрегацию результатов, формирование отчёта, сохранение истории, экспорт данных и отображение статистики. Все методы взаимодействуют через разделяемые поля, а синхронизация с графическим интерфейсом обеспечивается механизмом `invokeLater`. На рисунке 1.17 показана диаграмма классов приложения для запуска тестов.

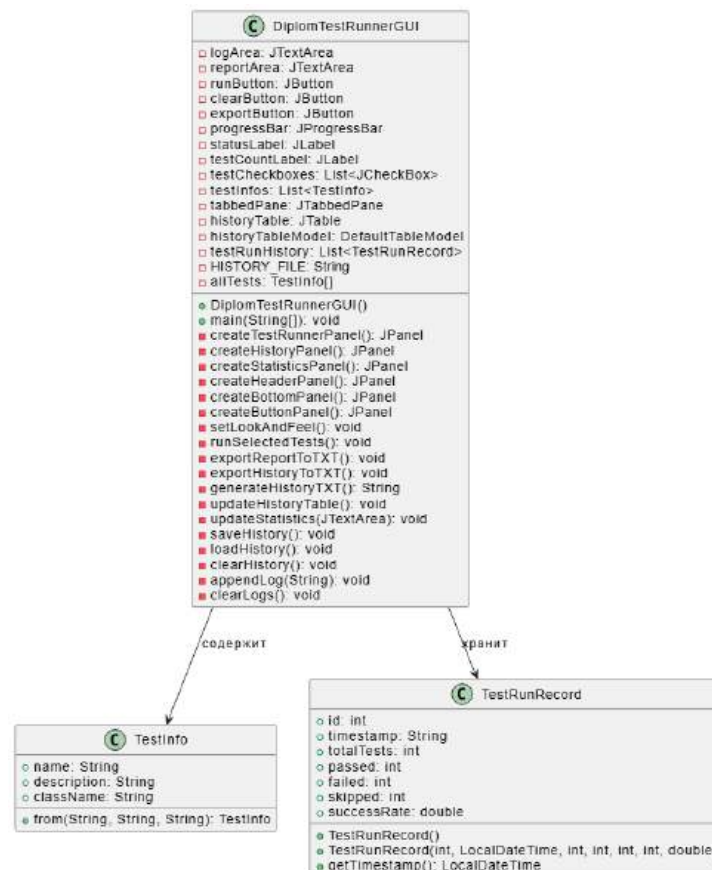


Рисунок 1.17 – Диаграмма классов приложения для запуска тестов

1.6.10 Описание графического интерфейса приложения

Экран запуска тестов является основным рабочим пространством приложения и предназначен для выбора тестовых модулей, управления процессом тестирования и наблюдения за его ходом. Он разделён на три функциональные области: панель выбора тестов, панель лога выполнения и панель итогового отчёта. Панель выбора тестов располагается в левой части экрана и содержит список всех доступных тестовых модулей в виде флажков (JCheckBox). Каждый флажок снабжён текстовой подписью с названием модуля и всплывающей подсказкой (ToolTipText) с кратким описанием проверяемой функциональности. По умолчанию все модули отмечены для запуска. Над списком расположены кнопки «Выбрать все» и «Снять все», позволяющие быстро изменить выбор. Под списком отображаются статусная информация: выбранное количество модулей и текущий статус выполнения.

Панель лога выполнения располагается в правой части экрана и предназначена для вывода текстовых сообщений в реальном времени. Лог выполнен в виде текстовой области (JTextArea) с чёрным фоном и зелёным шрифтом, что имитирует консольный вывод и обеспечивает хорошую читаемость. В лог выводятся сообщения о начале и окончании выполнения каждого тестового модуля, количестве пройденных тестов, а также сообщения об ошибках. Лог автоматически прокручивается вниз при добавлении новых сообщений, что позволяет пользователю наблюдать за ходом выполнения без необходимости ручной прокрутки. Под логом располагается панель статуса, содержащая индикатор прогресса (JProgressBar), отображающий процент выполнения, и текстовую метку с текущим состоянием («Готов к запуску», «Выполнение...», «Выполнено успешно», «Ошибка выполнения»).

Панель итогового отчёта расположена в нижней части экрана и отображается после завершения выполнения тестов. Она содержит сводную информацию о результатах тестирования: дату и время запуска, перечень проверенных модулей с указанием статуса («УСПЕШНО» или «ОШИБКА ВЫПОЛНЕНИЯ»), список ошибок с сообщениями исключений, а также общую статистику (всего тестов, пройдено, ошибок, пропущено, процент успешности). Отчёт представлен в текстовой области с голубым фоном для визуального отделения от лога выполнения. В нижней части основного окна расположена панель управления с кнопками: «Запустить выбранные тесты» – инициирует выполнение отмеченных модулей; «Очистить логи» – очищает лог выполнения и панель отчёта; «Экспорт отчёта» – сохраняет текущий отчёт в текстовый файл; «Выход» – завершает работу приложения.

В разработанном приложении основной экран просмотра результатов совмещён с экраном запуска тестов – итоговый отчёт отображается в нижней панели сразу после завершения тестирования. Это решение позволяет пользователю не переключаться между вкладками для получения результатов.

Однако для детального анализа предусмотрены две дополнительные вкладки. На рисунке 1.18 показан экран запуска тестов

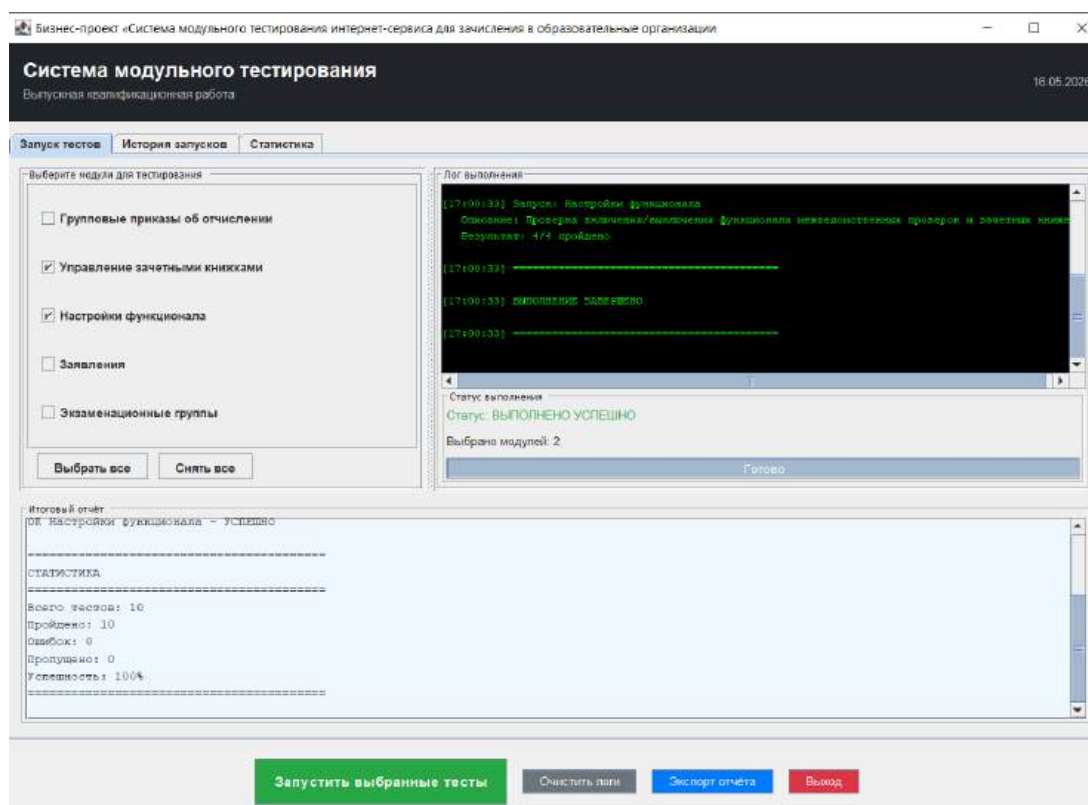


Рисунок 1.18 – Экран запуска тестов

Экран истории запусков реализован в виде отдельной вкладки «История запусков» и предназначен для хранения и просмотра результатов всех предыдущих сессий тестирования. Основным элементом вкладки является таблица (JTable), построенная на основе модели DefaultTableModel с отключённым редактированием ячеек. Таблица содержит следующие колонки: номер запуска (N), дата и время выполнения в формате dd.MM.yyyy HH:mm:ss, общее количество тестов, количество пройденных тестов, количество упавших тестов, процент успешности и визуальный индикатор тренда. Тренд вычисляется автоматически при добавлении новой записи: если успешность текущего запуска выше предыдущего, отображается символ «↑» (улучшение); если ниже – символ «↓» (ухудшение); если осталась неизменной – символ «→» (стабильность). Это позволяет быстро оценить динамику качества тестируемого сервиса.

Хранение истории осуществляется в текстовом файле test_history.txt, расположенном в рабочей директории приложения. Формат хранения — строки с разделителем «|», содержащие все поля записи. При запуске приложения данные автоматически загружаются из файла и отображаются в таблице. При каждом успешном завершении тестирования новая запись добавляется в таблицу и сохраняется в файл. На рисунке 1.19 показан экран просмотра истории поиска.

N	Дата и время	Всего тестов	Пройдено	Ошибок	Успешность	Тренд
9	16.05.2026 17:00:33	10	10	0	100.0%	↑
8	15.05.2026 23:13:14	30	29	1	96.6%	→
7	15.05.2026 21:17:17	30	29	1	96.6%	→
6	15.05.2026 21:17:04	30	29	1	96.6%	→
5	15.05.2026 21:16:52	30	29	1	96.6%	↑
4	15.05.2026 21:16:29	0	0	1	0.0%	↓
3	15.05.2026 21:16:20	6	6	1	100.0%	↑
2	15.05.2026 21:16:11	0	0	1	0.0%	↓
1	15.05.2026 21:16:15	26	25	1	96.0%	→

Рисунок 1.19 – Экран просмотра истории поиска

На вкладке истории также расположены кнопки управления: «Очистить историю» — удаляет все записи из таблицы и из файла с предварительным подтверждением пользователя; «Экспорт истории» — сохраняет все записи истории в текстовый файл для последующего анализа или архивирования. Для обеспечения наглядности таблица имеет настраиваемую высоту строк (25 пикселей) и шрифт заголовков, выделенный полужирным начертанием. Данные в таблице отображаются в порядке убывания — самые свежие запуски показываются первыми, что упрощает просмотр последних результатов. Дополнительно в приложении реализована вкладка «Статистика», где в текстовом виде представлена агрегированная информация по всей истории запусков: общее количество

выполненных тестов, средняя успешность, лучший и худший результаты. Эта информация обновляется при каждом изменении истории (новый запуск или очистка).

Вкладка «Статистика» предназначена для отображения агрегированной аналитики по всем выполненным запускам. Данные на этой вкладке представлены в виде текстовой области (JTextArea) с моноширинным шрифтом и голубым фоном, что обеспечивает единый стиль с остальными экранами приложения. Вкладка содержит следующую информацию: общее количество выполненных запусков; общее количество выполненных тестов за все запуски; общее количество пройденных тестов за все запуски; средняя успешность (среднее арифметическое процентов успешности всех запусков); лучший результат (максимальный процент успешности среди всех запусков); худший результат (минимальный процент успешности среди всех запусков). На рисунке 1.20 показан экран просмотра статистики запуска тестов.

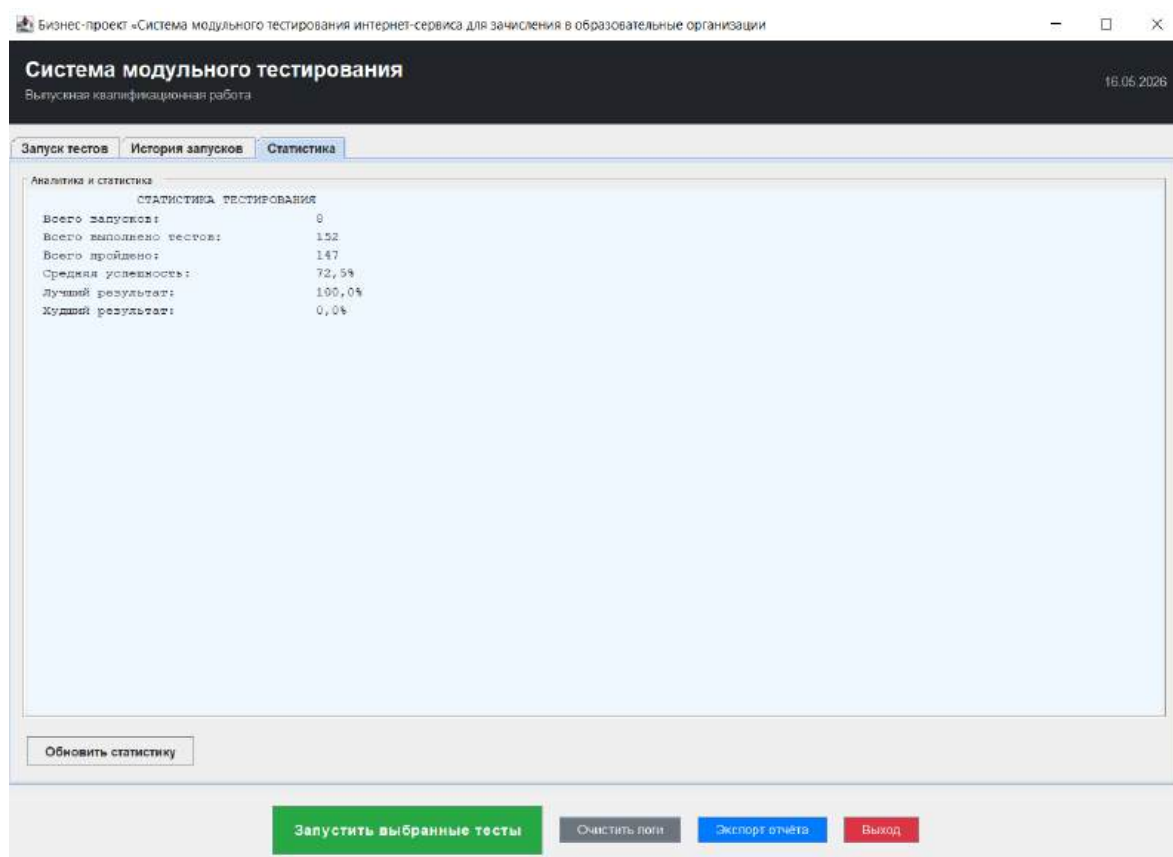


Рисунок 1.20 – Экран просмотра статистики запуска тестов

Данные на вкладке «Статистика» обновляются автоматически при каждом изменении истории запусков: после успешного завершения нового тестирования или после очистки истории. Для ручного обновления предусмотрена кнопка «Обновить статистику», которая пересчитывает все показатели на основе текущих данных истории. Это позволяет пользователю всегда видеть актуальную аналитику без необходимости перезапуска приложения.

1.7 Тестирование и отладка

В ходе разработки дипломного проекта проводилось тестирование двух основных компонентов: модульных тестов сервисов интернет-сервиса зачисления в образовательные организации и приложения для их запуска. Тестирование преследовало следующие цели: проверка корректности работы самих тестов, а также функциональная проверка приложения-раннера. Отладка проводилась параллельно с разработкой с использованием среды IntelliJ IDEA и встроенных инструментов отладки (точки останова, пошаговое выполнение, просмотр переменных).

План тестирования состоит из 5 основных частей:

- проверка модульных тестов. Нужно убедиться, что каждый тест действительно проверяет свой сервис, правильно обрабатывает позитивные и негативные сценарии, не обращается к реальной базе данных и не вызывает внешние зависимости (благодаря Mockito);
- проверка приложения-раннера. Здесь требуется уделить внимание графическому интерфейсу и логике самого приложения: выбор модулей, запуск, отображение лога и прогресса, формирование отчёта, экспорт в файл, ведение истории, вычисление статистики;
- комплексная проверка. Запуск нескольких циклов тестирования подряд, проверка сохранения и загрузки истории, устойчивость к отсутствию тестов и некорректным действиям пользователя (например, попытка запустить тесты без выбора модулей);

- анализ результатов прохождения тест-кейсов и исправление возможных ошибок;
- повторное тестирование всех компонентов программы.

1.7.1 Тестирование модульных тестов

Во всех тестовых классах зависимости сервисов (репозитории, API-клиенты, вспомогательные сервисы) были заменены мок-объектами. Для проверки того, что никакие тесты не обращаются к реальной базе данных или внешним API, достаточно того, что при выполнении тестов не возникает никаких попыток установить сетевое соединение или записать данные в реальную БД. Дополнительно использовались `verify(..., never())`, чтобы убедиться, что определённые методы моков не вызываются (например, `verify(applicationToExamGroupRepo, never()).save(any())` в тестах, где операция сохранения не должна происходить). Аннотация `@ExtendWith(MockitoExtension.class)` также гарантирует, что после каждого теста моки автоматически сбрасываются, исключая влияние одного теста на другой.

В процессе отладки модульных тестов были выявлены и устранены следующие недочёты: в `ExpulsionGroupOrderServiceTest` возникла ошибка `UnnecessaryStubbingException` из-за стаба `when(mockOrder.getOrderItems()).thenReturn(new ArrayList<>())`, вынесенного в `@BeforeEach`, но используемого только в одном тесте. Перенос стаба непосредственно в метод `shouldGetOrderSuccessfully` устранил предупреждения Mockito. Также были исправлены ошибки, связанные с использованием матчера `any()` без указания типа (заменено на `any(Student.class)`), и подмена возвращаемого значения для `void`-метода (вместо `when(...).thenThrow(...)` использовано `doThrow(...).when(...).method()`). После этих исправлений все тесты стали стабильно выполняться.

Тестирование проходило по тест кейсам модульных тестов представленным в таблице 1.1.

					ПГУ 1.090301.05.001 ПЗ	Лист
						53
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 1.1 – Тестовые сценарии для модульных тестов

Тестовый класс	Сценарий	ОР	ФР
ApplicationCreationCheckServiceComplexTest	Установить тип заявления – FINANCING_CHANGE	Исключение не бросается, isBanAppAdmission() не вызывается	Соответствует ОР
ApplicationCreationCheckServiceComplexTest	Установить ADMISSION, isBanAppAdmission=false	Исключение не бросается	Соответствует ОР
ApplicationCreationCheckServiceComplexTest	Установить ADMISSION, isBanAppAdmission=true, isCurrentDateOutOfAppAdmissionDates=true	Бросается ApplicationException	Соответствует ОР
ApplicationCreationCheckServiceComplexTest	Установить ADMISSION, isBanAppAdmission=true, isCurrentDateOutOfAppAdmissionDates	Исключение не бросается	Соответствует ОР

Тестовый класс	Сценарий	ОР	ФР
	missionDates =false		
ApplicationCreationCheckServiceComplexTest	Проверить заявление без типа (type=null)	NullPointerException не бросается	Соответствует ОР
ApplicationCreationCheckServiceComplexTest	Проверить цепочку из восьми проверок	Проверки вызываются в правильном порядке (InOrder)	Соответствует ОР
ApplicationCreationCheckServiceComplexTest	Вызвать ошибку toPpLevelBlockCheck	Остальные проверки не вызываются (never)	Соответствует ОР
ExamGroupServiceTest	Проверить получение, существующей группы по ID	Возвращается ExamGroupResponse	Соответствует ОР
ExamGroupServiceTest	Проверить получение, несуществующей группы по ID	ExamGroupNotFoundException	Соответствует ОР

Тестовый класс	Сценарий	ОР	ФР
ExamGroupServiceTest	Проверить удаление группы со статусом INITIAL, без студентов	Группа удалена (delete вызван 1 раз)	Соответствует ОР
ExamGroupServiceTest	Проверить удаление группы со статусом COMPLETE	ApplicationException, delete не вызывался	Соответствует ОР
ExamGroupServiceTest	Проверить удаление группы, в которой есть студенты (countAllByExamGroup=5)	ApplicationException, delete не вызывался	Соответствует ОР
ExamGroupServiceTest	Проверить возможность закрытия группы с прошедшей датой экзамена, все оценки есть	isAllowed() = true	Соответствует ОР

Тестовый класс	Сценарий	ОР	ФР
ExamGroupServiceTest	Проверить возможность закрытия группы с непрошедше й датой	isAllowed() = false	Соответствует ОР
ExamGroupServiceTest	Проверить выставление оценки, когда запись существует	setMark(1), setMarkedByC urrentUser(), setMarkedAt() вызваны	Соответствует ОР
ExamGroupServiceTest	Проверить выставление оценки, когда записи нет	ApplicationEx ception	Соответствует ОР
GradeBookServiceTest	Проверить получение информации о зачётной книжке	convert() вызван, ответ не null	Соответствует ОР
GradeBookServiceTest	Проверить добавление зачётной книжки	assertNoGrade Book() вызван, save() выполнен	Соответствует ОР

Тестовый класс	Сценарий	ОР	ФР
GradeBookServiceTest	Проверить обновление зачётной книжки	assertHasGradeBook() вызван, save() выполнен	Соответствует ОР
GradeBookServiceTest	Проверить удаление зачётной книжки	save() выполнен (студент изменён)	Соответствует ОР
GradeBookServiceTest	Проверить работу исключения при отсутствии студента	Выпадает NotFoundException	Соответствует ОР
GradeBookServiceTest	Проверить, что при отсутствии зачетной книжки при обновлении выбрасывается исключение	Выпадает RuntimeException	Соответствует ОР
ProfSettingAssertionServiceTest	Установить AGENCY_ENABLED=false	Исключение не бросается	Соответствует ОР

Тестовый класс	Сценарий	ОР	ФР
ProfSettingAssertionServiceTest	Установить assertAgencyEnabled – AGENCY_ENABLED=true	ApplicationException с текстом «Функционал запуска межведомственных проверок выключен»	Соответствует ОР
ProfSettingAssertionServiceTest	Установить assertAllowFunctionalGradeBook – ALLOW_FUNCTIONAL_GRADE_BOOK=false	Исключение не бросается	Соответствует ОР
ProfSettingAssertionServiceTest	Установить assertAllowFunctionalGradeBook – ALLOW_FUNCTIONAL_GRADE_BOOK=true	ApplicationException с текстом «Функционал зачетной книжки выключен»	Соответствует ОР
ExpulsionGroupOrderServiceTest	Проверить получение	Возвращается ExpulsionGroupOrderContent	Соответствует ОР

Тестовый класс	Сценарий	ОР	ФР
	приказа об отчислении	Response, orderConvertService вызван	
ExpulsionGroupOrderServiceTest	Проверить создание приказа об отчислении	orderTransactionHandler.runInTransaction вызван	Соответствует ОР
ExpulsionGroupOrderServiceTest	Проверить обновление приказа об отчислении	orderAssertService.assertOrderStatusForUpdate вызван	Соответствует ОР
ExpulsionGroupOrderServiceTest	Проверить удаление приказа об отчислении	orderAssertService.assertOrderStatusForDelete вызван	Соответствует ОР

1.7.2 Тестирование приложения для запуска тестов

Приложение для запуска тестировалось на предмет выполнения своих основных функций: отображение интерфейса, выбор модулей, запуск тестов, отображение лога и прогресса, формирование отчёта, экспорт, работа с историей и статистикой, обработка ошибочных ситуаций.

В процессе отладки приложения-раннера были исправлены следующие дефекты. Первоначально тесты выполнялись в основном потоке, из-за чего графический интерфейс зависал на время выполнения; проблема решена вынесением запуска в отдельный поток. Попытки напрямую изменять компоненты Swing из рабочего потока вызывали исключения; все обновления GUI были обёрнуты в `SwingUtilities.invokeLater()`. При первом запуске отсутствие файла

test_history.txt приводило к FileNotFoundException; в метод loadHistory() добавлена проверка существования файла. В таблице 1.2 представлены тестовые сценарии для приложения запуска тестов.

					ПГУ 1.090301.05.001 ПЗ	Лист
						61
Изм.	Лист	№ докум.	Подпись	Дата		

Таблица 1.2 – Тестовые сценарии для приложения запуска тестов

Сценарий	Шаги	ОР	ФР
Запуск всех пяти модулей	1. Открыть вкладку «Запуск тестов»; 2. Убедиться, что все флажки установлены; 3. Нажать «Запустить выбранные тесты»	Прогресс-бар растёт, лог содержит 5 модулей, в отчёте отображена информация о тестировании	Соответствует ОР
Запуск одного модуля	1. Нажать «Снять все»; 2. Отметить только «Заявления»; 3. Запустить	Выполняется только один модуль, прогресс с 0 до 100% после него, отчёт содержит информацию про тестирование заявлений	Соответствует ОР
Запуск без выбора модулей	1. Снять все флажки; 2. Нажать «Запустить выбранные тесты»	Появляется предупреждение «Не выбран ни один тест для запуска!», запуск не происходит	Соответствует ОР
Очистка логов	1. Выполнить хотя бы один запуск; 2. Нажать «Очистить логи»	Лог и отчёт очищены, счётчик модулей = 0, прогресс-бар 0%, статус «Готов к запуску»	Соответствует ОР
Экспорт отчёта после успешного запуска	1. Запустить тесты; 2. Нажать «Экспорт отчёта»; 3. Сохранить файл	Файл test_report_*.txt создан, его содержимое совпадает с отчётом в интерфейсе	Соответствует ОР
Экспорт без запуска	1. Не запуская тесты, нажать «Экспорт отчёта»	Предупреждение «Нет данных для экспорта».	Соответствует ОР

Сценарий	Шаги	ОР	ФР
		Сначала запустите тесты.»	
Просмотр истории	1. Выполнить 2–3 запуска с разными результатами; 2. Перейти на вкладку «История запусков»	Таблица содержит строки с номерами запусков, датой/временем, количеством тестов, пройдено, ошибок, успешностью, трендом	Соответствует ОР
Тренды в истории	1. Запуск1: 100%; 2. Запуск2: 80%; 3. Запуск3: 90%	Для запуска2 – ↓, для запуска3 – ↑, для первого – «-»	Соответствует ОР
Очистка истории	1. Перейти на вкладку «История запусков»; 2. Нажать «Очистить историю»; 3. Подтвердить	Таблица очищается, файл test_history.txt становится пустым, в интерфейсе информация исчезает	Соответствует ОР
Экспорт истории	1. Иметь непустую историю; 2. Нажать «Экспорт истории»; 3. Сохранить файл	Файл test_history_*.txt содержит блоки с данными каждого запуска и общее количество запусков	Соответствует ОР
Просмотр статистики	1. Выполнить несколько запусков; 2. Перейти на вкладку «Статистика»	Отображаются: всего запусков, всего выполнено тестов, всего пройдено, средняя успешность, лучший и худший результат	Соответствует ОР

Сценарий	Шаги	ОР	ФР
Обновление статистики	1. На вкладке «Статистика»; 2. Нажать «Обновить статистику»	Данные пересчитаны, совпадают с теми, что были до обновления	Соответствует ОР
Ситуация «класс не найден»	1. Временно изменить в массиве allTests полное имя класса на несуществующее; 2. Запустить этот модуль	В логе «ОШИБКА: Класс не найден», в отчёте «X ... – КЛАСС НЕ НАЙДЕН», статус «ОШИБКА ВЫПОЛНЕНИЯ», программа не падает	Соответствует ОР
Многопоточность (независание)	1. Запустить длительные тесты; 2. Во время выполнения переключаться между вкладками	Интерфейс остаётся отзывчивым, можно переключать вкладки, прогресс-бар обновляется	Соответствует ОР

По результатам тестирования и отладки установлено, что все 30 модульных тестов успешно проходят, обеспечивая проверку бизнес-логики сервисов с полной изоляцией зависимостей. Приложение-раннер корректно выполняет запуск выбранных модулей через JUnit Platform Launcher, отображает ход выполнения в реальном времени, формирует детализированный итоговый отчёт с возможностью экспорта, ведёт историю запусков с правильным вычислением трендов и агрегирует статистику. Ошибочные ситуации обрабатываются без вылета приложения с выводом понятных сообщений. Все выявленные дефекты устранены, приложение готово к эксплуатации.

1.8 Руководство пользователя

Для обеспечения эффективного взаимодействия разработанной системы модульного тестирования с пользователем представлено чёткое и

последовательное описание всех доступных функций, сценариев работы и интерфейсных элементов. Руководство предназначено для пользователей, которые будут выполнять запуск модульных тестов, анализировать результаты и вести историю тестирования интернет-сервиса зачисления в образовательные организации. На рисунке 1.21 показана диаграмма вариантов использования системы модульного тестирования.

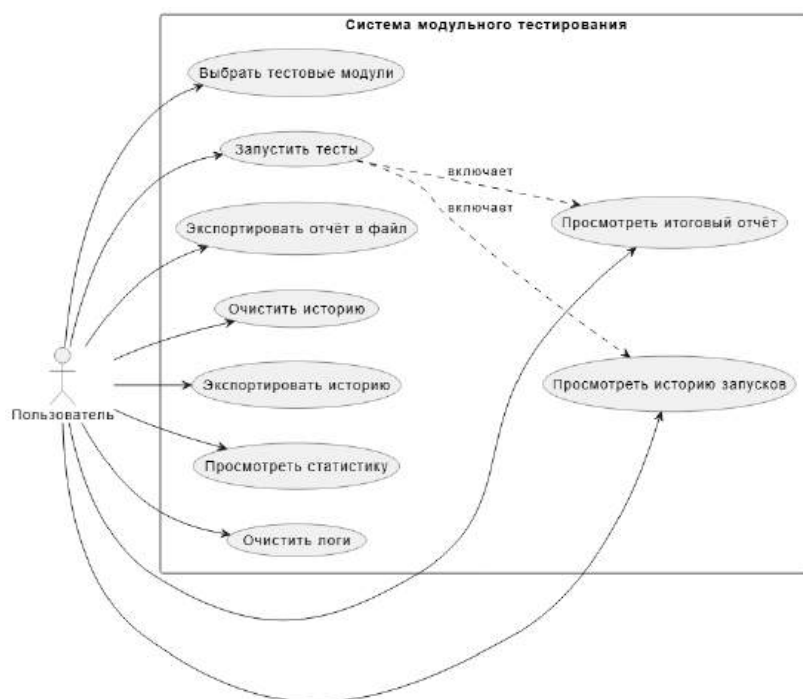


Рисунок 1.21 – Диаграмма вариантов использования системы модульного тестирования

1.8.1 Начало работы

Для запуска системы модульного тестирования необходимо запустить главный класс DiplomaTestRunnerGUI из среды разработки. После запуска на экране появляется главное окно приложения. По умолчанию при запуске приложения активна вкладка «Запуск тестов». В нижней части окна расположена панель с четырьмя кнопками: «Запустить выбранные тесты», «Очистить логи», «Экспорт отчёта» и «Выход». На рисунке 1.22 показано начальное окно приложения для запуска тестов.

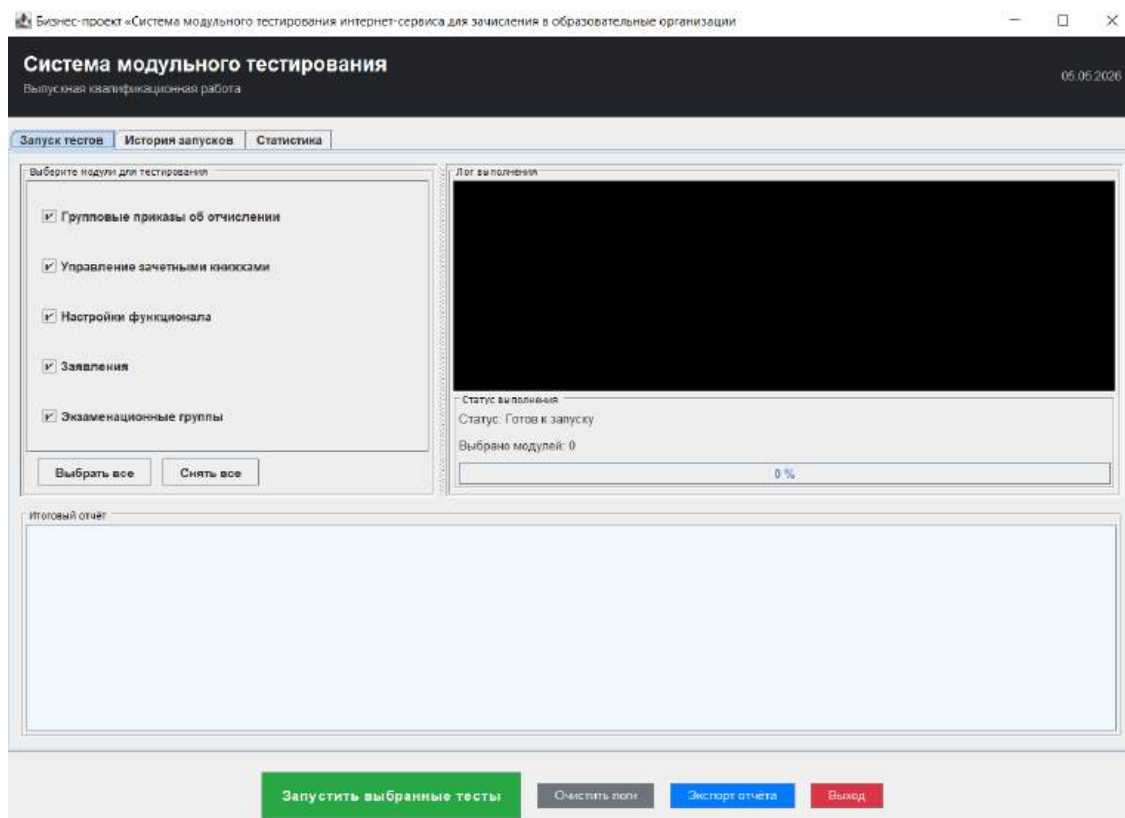


Рисунок 1.22 – Начальное окно приложения для запуска тестов

Вкладка «Запуск тестов» – выбор тестовых модулей

При открытой вкладке «Запуск тестов» пользователь видит разделённое на две части окно. Левая часть для выбора тестовых модулей. Здесь представлен список всех доступных тестовых модулей.

При наведении указателя мыши на название любого модуля появляется всплывающая подсказка (ToolTipText) с кратким описанием проверяемой функциональности, что помогает пользователю сориентироваться. По умолчанию все флажки установлены (выбраны все модули). Справа от списка модулей расположена область «Лог выполнения», где в реальном времени выводятся сообщения о ходе тестирования. Под логом находится блок «Статус выполнения» с индикатором прогресса (0 %), строкой «Статус: Готов к запуску» и счётчиком «Выбрано модулей: 0». В нижней части вкладки размещена область «Итоговый отчёт». В самом низу окна расположены четыре кнопки: «Запустить выбранные тесты», «Очистить логи», «Экспорт отчета», «Выход».

Для выбора тестов необходимо отметить флажками нужные модули. Пользователь может установить флажок напротив любого модуля щелчком мыши – модуль будет добавлен в список для запуска, снять флажок – модуль исключается, нажать кнопку «Выбрать все» – все пять модулей будут отмечены. Нажать кнопку «Снять все» – все отметки будут сброшены.

После того как нужные модули отмечены, необходимо нажать кнопку «Запустить выбранные тесты». На рисунке 1.23 показано обнаружение ошибки при тестировании.

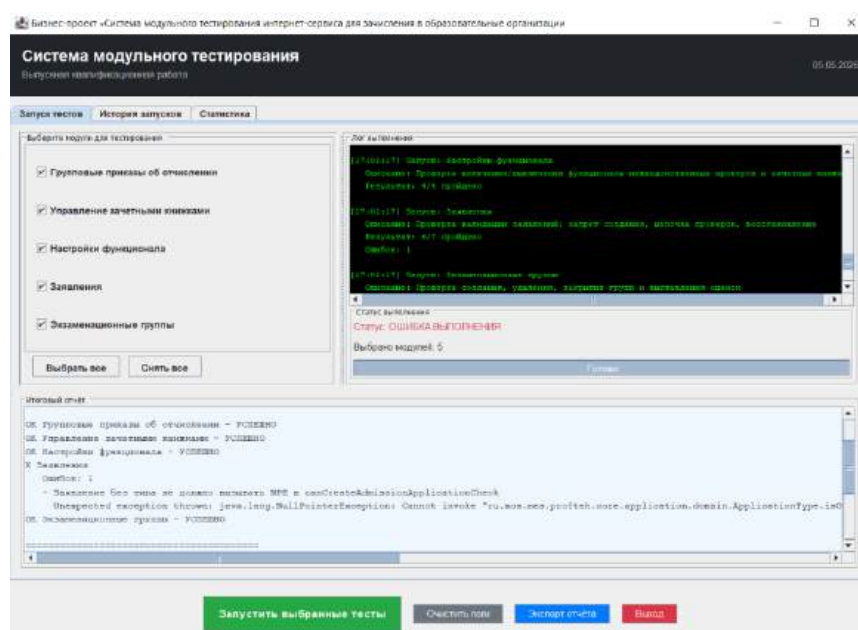


Рисунок 1.23 – Обнаружение ошибки при тестировании

При этом кнопка запуска блокируется (становится неактивной) на время выполнения тестов. Индикатор прогресса обнуляется (0 %) и начинает отображать процент выполнения. Строка статуса меняется на «Статус: Выполнение...». В области «Лог выполнения» начинают выводиться служебные сообщения с временными метками: информация о начале запуска, список выбранных модулей, результаты выполнения каждого модуля. Для каждого модуля в логе отображается его название, описание и результат в формате «количество пройденных тестов / общее количество тестов». Если тесты упали, выводится количество ошибок и текст исключения.

Индикатор прогресса заполняется по мере обработки модулей. Когда все модули выполнены, индикатор показывает 100 % и текст «Готово».

Процесс тестирования выполняется в отдельном потоке, поэтому интерфейс остаётся отзывчивым, пользователь может переключать вкладки, но не может запустить новый тест до завершения текущего.

После завершения всех выбранных модулей в нижней части вкладки «Запуск тестов» в блоке «Итоговый отчёт» появляется текстовый отчёт. Пользователь может скопировать текст отчёта из области или сохранить его в файл.

Кнопка «Очистить логи» удаляет всё содержимое области «Лог выполнения» и области «Итоговый отчёт», сбрасывает счётчик выбранных модулей на 0, обнуляет индикатор прогресса и меняет строку статуса на «Статус: Готов к запуску». После этого в логе появляется сообщение «Лог очищен. Готов к запуску.». Это позволяет начать новый сеанс без данных предыдущего запуска. На рисунке 1.24 показана очистка логов и отчета. Кнопка «Экспорт отчета» сохраняет текущий итоговый отчёт в текстовый файл. Перед сохранением проверяется, что область итогового отчёта не пуста. Если отчёт пуст (тесты не запускались), появляется предупреждение. На рисунке 1.25 показано предупреждение об отсутствии выбранных тестов. Иначе открывается диалог сохранения файла с предложением имени по шаблону test_report_ГГГГММДД_ЧЧММСС.txt. Пользователь может выбрать папку и изменить имя. После подтверждения файл сохраняется, и появляется сообщение об успехе. На рисунке 1.26 показано успешное сохранение отчета в файл.

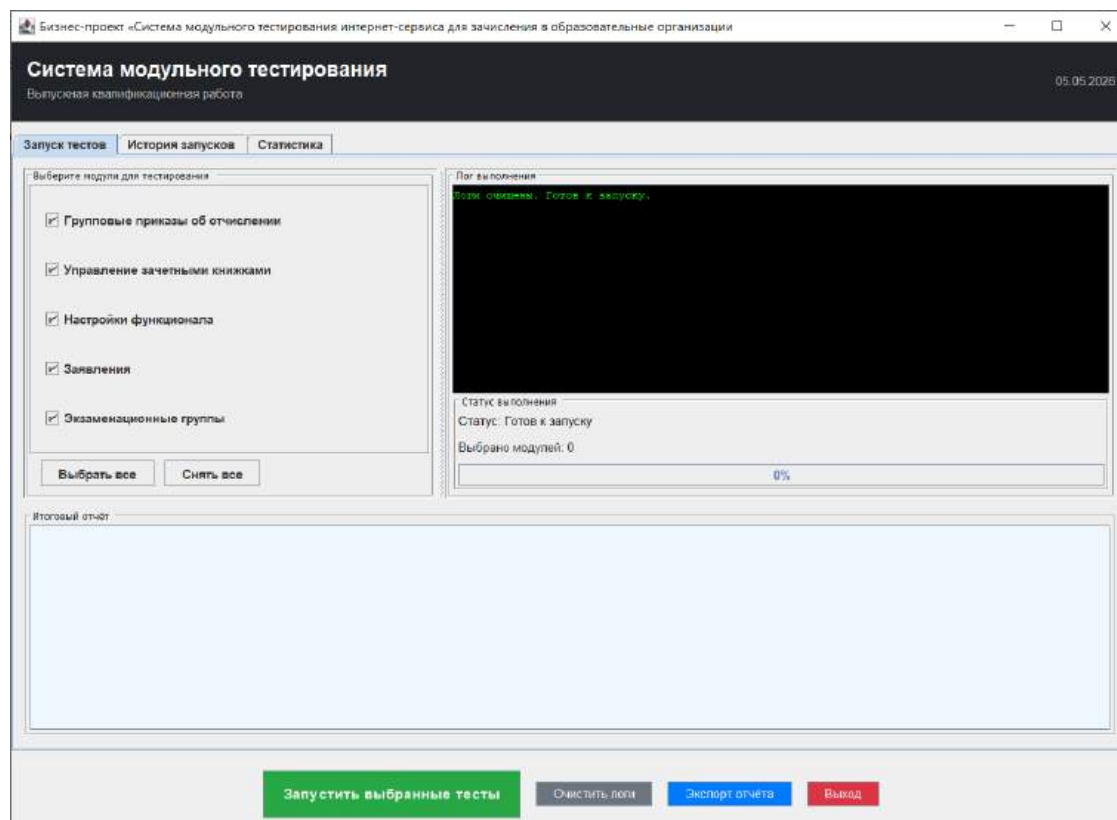


Рисунок 1.24 – Очистка логов и отчета

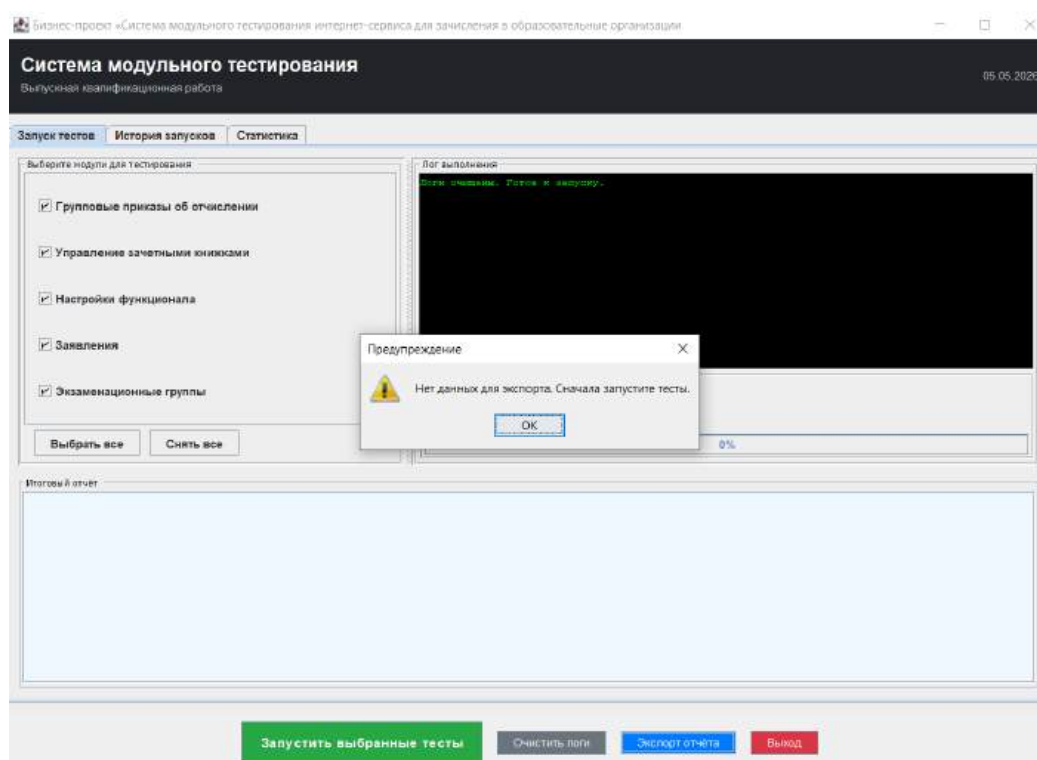


Рисунок 1.25 – Предупреждение об отсутствии выбранных тестов

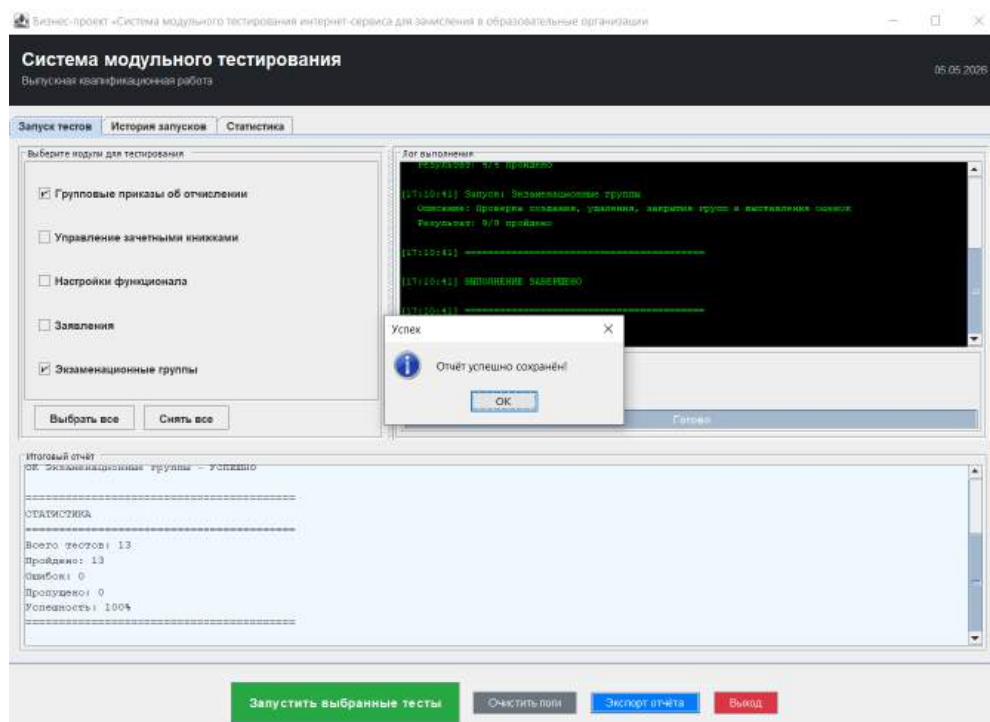


Рисунок 1.26 – Успешное сохранение отчета

1.8.2 Просмотр истории запусков

При переходе на вкладку «История запусков» отображается история тестирования. На рисунке 1.27 показана история запусков тестов.

Система модульного тестирования
Выпускная квалификационная работа

05.05.2026

Запуск тестов История запусков Статистика

История запусков тестов

N	Дата и время	Всего тестов	Пройдено	Ошибок	Успешность	Тренд
52	05.05.2026 17:10:41	13	13	0	100.0%	↑
51	05.05.2026 17:01:17	30	29	1	96.6%	↓
50	05.05.2026 16:56:30	19	19	0	100.0%	↑
49	05.05.2026 16:56:27	30	29	1	96.6%	→
48	05.05.2026 15:09:42	30	29	1	96.6%	→
47	04.05.2026 23:55:09	30	29	1	96.6%	↓
46	04.05.2026 23:01:22	10	10	0	100.0%	→
45	04.05.2026 23:01:21	10	10	0	100.0%	→
44	04.05.2026 23:01:20	10	10	0	100.0%	→
43	04.05.2026 23:01:19	10	10	0	100.0%	→
42	04.05.2026 22:10:54	10	10	0	100.0%	→
41	24.04.2026 16:33:09	23	23	0	100.0%	↑
40	24.04.2026 14:22:10	30	29	1	96.6%	→
39	19.04.2026 15:10:11	30	29	1	96.6%	↓
38	18.04.2026 13:51:09	9	9	0	100.0%	→
37	18.04.2026 13:18:40	13	13	0	100.0%	↑
36	18.04.2026 13:17:51	30	29	1	96.6%	↑
35	18.04.2026 13:10:04	11	10	1	90.9%	↓
34	18.04.2026 00:43:30	4	4	0	100.0%	↑

Очистить историю Экспорт истории

Запустить выбранные тесты Очистить логи Экспорт отчета Выход

Рисунок 1.27 – Просмотр истории запуска

Записи отсортированы от свежих к старым. Под таблицей расположены кнопки «Очистить историю» – удаляет все записи из таблицы и из файла test_history.txt (перед удалением запрашивается подтверждение). На рисунке 1.28 показано подтверждение очистки истории запуска. Результат очистки истории показан на рисунке 1.29.

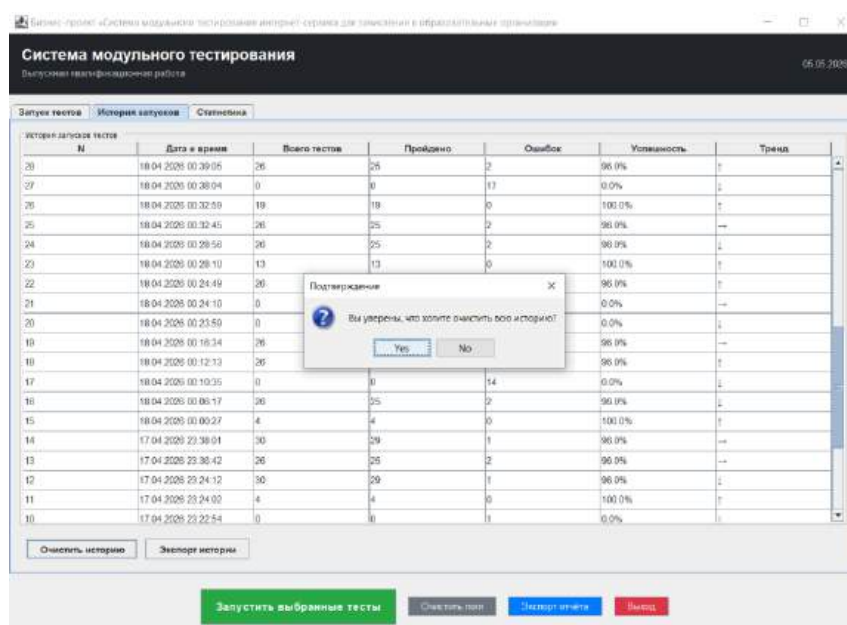


Рисунок 1.28 – Подтверждение очистки истории запуска

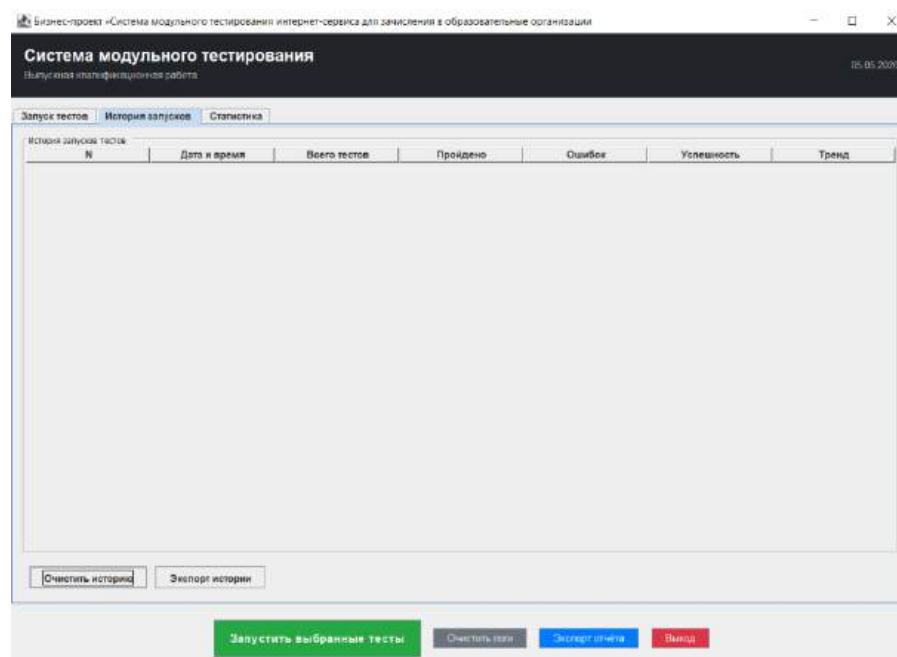


Рисунок 1.29 – История очищена

«Экспорт истории» – сохраняет все записи в текстовый файл. Файл содержит блоки с подробными данными о каждом запуске и общее количество запусков. На рисунке 1.30 показано уведомление о том, что история успешно сохранена в файл.

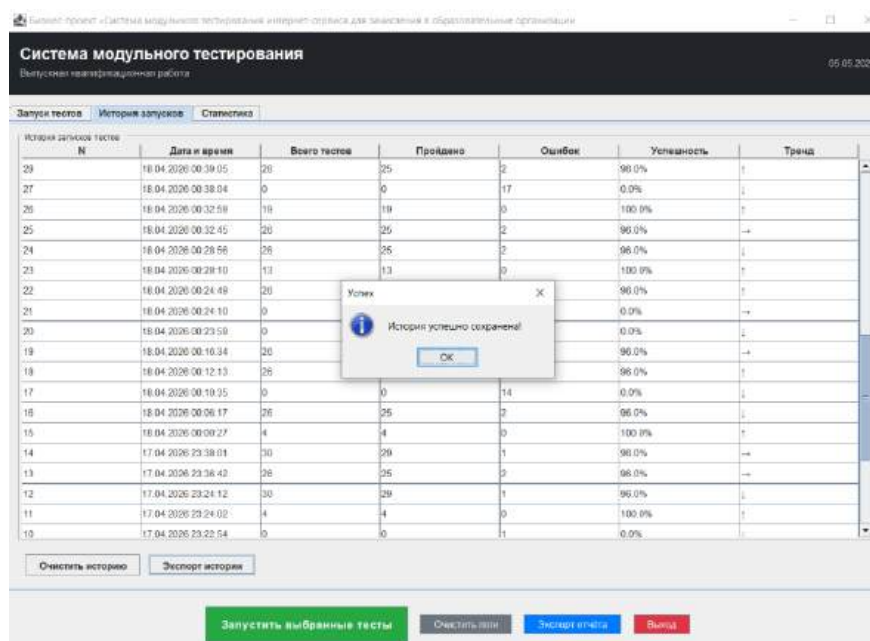


Рисунок 1.30 – История сохранена в файл

1.8.3 Просмотр статистики

Вкладка «Статистика» отображает агрегированные показатели по всей истории запусков. На рисунке 31 показан просмотр статистики в соответствующей вкладке. Данные обновляются автоматически после каждого нового запуска или очистки истории. Кнопка «Обновить статистику» позволяет пересчитать показатели вручную.

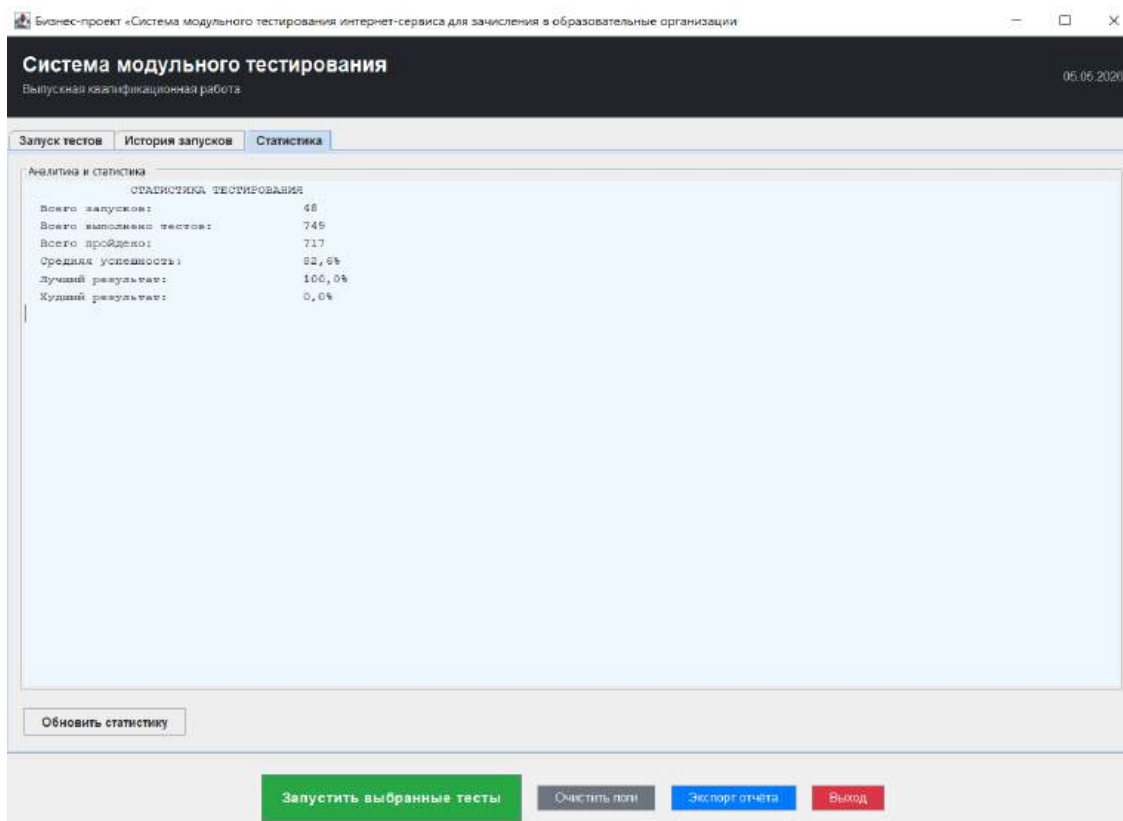


Рисунок 1.31 – Просмотр статистики

2 Бизнес-модель

Бизнес-модель проекта основана на продаже лицензии программного продукта. В неё входит и сам набор модульных тестов, графическое приложение-раннер, и годовая техподдержка с обновлениями. Стоимость одной лицензии – 290 000 рублей. В первый год поддержка идёт в подарок (она уже включена в цену), так что отдельные платежи за неё не вносятся, но расходы на саму поддержку мы всё равно есть – это около 90 000 рублей в год, они заложены в операционные затраты. Оставшиеся 200 тысяч рублей складываются из затрат, которые были уже использованы при создании системы. 1500 рублей в час – это ставка начинающего Java-разработчика, на данный проект было потрачено порядка 126 часов, получается, что разработка приложения выходит в 189 000 рублей. Около 10 000 рублей – это расходы на электроэнергию и за амортизацию ПК, на котором велась разработка

Это приложение для IT-отделов образовательных платформ. Поэтому классическая воронка с рекламой вряд ли сработает. Основной канал – личные контакты, выступления на профильных конференциях, публикация кода на GitHub и общение на форумах вроде Хабра или Stack Overflow. По сути, это B2B-продажа через доверие и демонстрацию.

Требуемые инвестиции исходя из реальных трудозатрат и минимальных рыночных цен на аналогичные работы – 600 тысяч – это сумма, которая позволяет гарантированно довести систему до состояния, пригодного для передачи первому заказчику.

Сейчас написано 30 модульных тестов для пяти ключевых сервисов. Но для серьёзного внедрения нужно покрыть больше сценариев: граничные случаи, негативные тесты, тесты на исключения. На это потребуется примерно 2 месяца работы одного Java-разработчика (средняя ставка 150 000 руб./мес. на рынке зарплаты Java-специалистов с опытом 1–3 года в регионах или на удалёнке). Также сюда входит оплата рефакторинга существующих тестов, чтобы они легко поддерживались. Графическое приложение есть, но его интерфейс и возможности

					ПГУ 1.090301.05.001 ПЗ	Лист
						74
Изм.	Лист	№ докум.	Подпись	Дата		

можно усовершенствовать. Чтобы передать его заказчику, нужны: удобные подсказки, обработка нестандартных разрешений экрана, доработка отчётов (например, возможность сохранять в PDF или сразу отправлять на печать), улучшение логирования и обработки ошибок (чтобы пользователь не видел stack trace, а получал понятные сообщения).

Это примерно 1,5 месяца работы фронтенд-разработчика (можно того же Java-специалиста, который знает Swing). Стоимость работ обойдется около 180 000 рублей. Первичный маркетинг и подготовка к передаче потребителю обойдется в 120 000 руб.

Сначала был составлен консервативный план, который показан в таблице 2.1 для одного пилотного заказчика. По нему видно, что без масштабирования проект не окупается.

Таблица 2.1 - Финансовые показатели пилотного этапа (одна лицензия)

Показатель	1 год	2 год	3 год
Продажа продукта	200 000 рублей	-	-
Обновления	50 000 рублей	50 000 рублей	50 000 рублей
Доработки (средняя цена 20 000 Р)	40 000 рублей	60 000 рублей	80 000 рублей
ИТОГО выручка	290 000 рублей	110 000 рублей	130 000 рублей

Один заказчик даст за три года всего 530 тыс. рублей, а требуемые инвестиции в разработку – 600 тыс.

Если начать продавать лицензии новым клиентам и ежегодно продлевать поддержку действующим, картина меняется. При этом каждый клиент, купивший лицензию, каждый год платит за продление техподдержки. Ожидается, что каждый год продаётся ровно одна новая лицензия за 290 000 рублей. Доход от сопровождения я посчитала не фиксированной суммой, а с ежегодным ростом. У

каждого клиента стоимость продления техподдержки каждый год увеличивается на 20 000 рублей, потому что система дорабатывается, появляются новые функции. В таблице 2.2 показываются какие доходы ожидаются при таких условиях.

Таблица 2.2 – Прогноз выручки при масштабировании

Год	Действующих лицензий (на конец года)	Доход от новых лицензий, руб.	Клиентов на платном сопровождении	Доход от сопровождения, руб.	Общая выручка, руб.
1-й	1	290 000	0	0	290 000
2-й	2	290 000	1	110 000	400 000
3-й	3	290 000	2	240 000 (110 000 + 130 000)	530 000
4-й	4	290 000	3	390 000 (110000+130 000 +150 000)	680 000
5-й	5	290 000	4	560000 (110000 +130000+150000 +170000)	850 000

У продукта низкие операционные расходы – он цифровой, большой команды не требует. Я выделила:

Постоянные расходы – 200 000 рублей в год. Сюда входят: поддержка ядра (исправление багов, совместимость с новыми версиями Java), минимальный маркетинг (публикации на GitHub, ответы на форумах) и администрирование.

Переменные расходы – 40% от выручки. Это типичная доля для программного стартапа: маркетинговые комиссии, платные инструменты для демонстрации, часы техподдержки сверх базовой. В таблице 2.3 рассчитан финансовый результат за первые 5 лет.

Таблица 5 – Финансовый результат при масштабировании

Год	Общая выручка, руб.	Постоянные расходы, руб.	Переменн ые расходы (40% от выручки), руб.	Итого расходы, руб.	Чистая выручка, руб.
1-й	290 000	200 000	116 000	316 000	–26 000
2-й	400 000	200 000	160 000	360 000	+40 000
3-й	530 000	200 000	212 000	412 000	+118 000
4-й	680 000	200 000	272 000	472 000	+208 000
5-й	850 000	200 000	340 000	540 000	+310 000

3 Охрана труда

Охрана труда при работе с компьютерами регулируется несколькими документами, в первую очередь СанПиН 1.2.3685-21 «Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания» [15]. Если говорить простыми словами, законодательство требует, чтобы помещение, где стоят компьютеры, было светлым и сухим. Естественный свет предпочтительнее искусственного, но если его нет, инженеры должны доказать, что работать можно без вреда для глаз. Окна желательно оборудовать жалюзи или плотными шторами, чтобы солнечные лучи не создавали бликов на экране.

Не рекомендуется размещать компьютер во влажном и темном — влага для электроники губительна. Каждое рабочее место должно быть заземлено. Рекомендуется периодически делать влажную уборку. Оптимальная температура воздуха, согласно санитарным нормам, составляет от 19 до 21 градуса.

Стол для работы должен быть удобным, на нём должны помещаться клавиатура, мышь и монитор, причём монитор должен располагаться на расстоянии вытянутой руки (500–700 мм от глаз). Системный блок лучше установить на подставку или так, чтобы между вентиляционными отверстиями и стеной оставалось не менее 10 см — иначе возможен перегрев.

Что касается расстановки рабочих мест, то между столами с мониторами должно быть не менее 2 метров спереди и 1,2 метра сбоку. Нельзя ставить компьютер рядом с батареей или нагревателем — перегрев выводит технику из строя. Также опасны силовые кабели и трансформаторы: они создают электромагнитные помехи, из-за которых компьютер может работать нестабильно. Провода не рекомендуется тянуть по полу, если же это необходимо, их следует прятать в специальные короба, чтобы никто не споткнулся.

Освещение в комнате должно быть равномерным, яркостью около 300–500 люкс на столе. Экран при этом не должен светиться ярче 300 люкс, иначе глаза быстро устают. Лучше, если свет падает сбоку, а не прямо в монитор.

					ПГУ 1.090301.05.001 ПЗ	Лист
						78
Изм.	Лист	№ докум.	Подпись	Дата		

При создании приложения для модульного тестирования использовались контрастные цвета (чёрный фон и зелёный шрифт в логге), крупные кнопки, индикаторы прогресса. Это не отменяет правил охраны труда, но хотя бы снижает нагрузку на зрение при длительной работе.

Помимо уже перечисленных требований к помещению и оборудованию, огромное значение для сохранения здоровья программиста имеет правильная поза во время работы. Длительное пребывание в сидячем положении с неправильной осанкой приводит к болям в спине, напряжению глаз, туннельному синдрому запястья и другим неприятным последствиям. Чтобы избежать этого, рекомендуется соблюдать определенные правила.

Верхняя граница монитора должна находиться на уровне глаз или чуть ниже (не более чем на 15 см). Садиться нужно так, чтобы спина была прямой, плечи расправлены и отведены назад. Голова при этом держится так, чтобы уши находились на одной вертикальной линии с плечами, а не были выдвинуты вперёд. Поясница должна опираться на изгиб спинки кресла — это снижает нагрузку на позвоночник.

Руки должны свободно лежать вдоль корпуса, локти согнуты под прямым или чуть большим углом и находятся примерно в трёх сантиметрах от корпуса. Предплечья и запястья не должны висеть в воздухе; для них предусмотрены подлокотники и подставки. Запястья при работе с клавиатурой и мышью должны быть прямыми, а не согнутыми вверх или вниз, чтобы не травмировать сухожилия. Саму клавиатуру размещают на расстоянии 10–30 см от края стола, чтобы не приходилось тянуться.

Ноги должны стоять на полу или на специальной подставке всей ступнёй. Колени при этом сгибаются под углом 90 градусов и находятся чуть выше уровня бёдер. Ни в коем случае нельзя скрещивать ноги — это ухудшает кровообращение.

Даже при идеально организованном месте статические нагрузки никто не отменял. Поэтому обязательное требование — регулярно менять позу, вставать из-

за стола, разминаться. Например, делать короткие перерывы каждый час, чтобы пройтись или выполнить пару простых упражнений.

					ПГУ 1.090301.05.001 ПЗ	Лист
						80
Изм.	Лист	№ докум.	Подпись	Дата		

Заключение

В ходе выполнения выпускной квалификационной работы была разработана система модульного тестирования интернет-сервиса зачисления в образовательные организации. Цель работы, заключающаяся в создании программного комплекса для автоматизации тестирования ключевых сервисов (обработка заявлений, управление приказами, зачётными книжками и экзаменационными группами), достигнута в полном объёме.

В рамках работы решены следующие задачи. Проведён анализ существующих систем модульного тестирования и средств непрерывной интеграции (Maven Surefire Plugin, JUnit 5 Console Launcher, Jenkins, GitLab CI, Gradle Test, TeamCity, CircleCI), выявлены их преимущества и недостатки, обоснован выбор технологий: Java 21, JUnit 5, Mockito, Maven, Swing. Разработана архитектура системы, включающая тестируемые сервисы, набор модульных тестов и приложение-раннер с графическим интерфейсом. Спроектированы и реализованы модульные тесты для пяти ключевых сервисов (групповые приказы об отчислении, управление зачётными книжками, настройки функционала, заявления, экзаменационные группы). Тесты используют JUnit 5 и Mockito, обеспечивая изоляцию зависимостей и быстрое выполнение. Разработано приложение-раннер на Java Swing, которое позволяет выбирать тестовые модули через флажки, программно запускать их через JUnit Platform Launcher, наблюдать ход выполнения в реальном времени (лог, индикатор прогресса), формировать итоговый текстовый отчёт с экспортом, вести историю запусков с отображением трендов и вычислять агрегированную статистику.

Помимо этого была проанализирована и описана бизнес-модель для дальнейшего использования системы и масштабирования, были рассчитаны затраты и выручка, которую может принести данный проект.

					ПГУ 1.090301.05.001 ПЗ	Лист
						81
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Модульное тестирование [Электронный ресурс] // Wiki ПМИФИ. – Режим доступа: <https://wiki.pmifi.ru/ru/disciplines/testing/lectures/unit-tests>, свободный. – (дата обращения: 14.04.2026).
2. Котляров, В. П. Основы тестирования программного обеспечения : учебное пособие / В. П. Котляров, Т. В. Коликова. — 2-е изд., испр. — Москва : Национальный Открытый Университет "ИНТУИТ", 2016. — 348 с. — (Основы информационных технологий). — ISBN 5-9556-0027-2.
3. Maven Surefire Plugin. [Электронный ресурс] / Режим доступа: <https://maven.apache.org/surefire/maven-surefire-plugin/>, свободный. (дата обращения: 14.04.2026).
4. JUnit 5 Console Launcher. [Электронный ресурс] / Режим доступа: <https://docs.junit.org/5.0.0/user-guide/>, свободный. (дата обращения: 14.04.2026).
5. Jenkins. [Электронный ресурс] / Режим доступа: <https://plugins.jenkins.io/junit/>, свободный. (дата обращения: 14.04.2026).
6. GitLab CI. [Электронный ресурс] / Режим доступа: https://docs.gitlab.com/ci/testing/unit_test_reports/, свободный. (дата обращения: 14.04.2026).
7. Gradle Test. [Электронный ресурс] / Режим доступа: https://docs.gradle.org/current/userguide/java_testing.html, свободный. (дата обращения: 14.04.2026).
8. TeamCity. [Электронный ресурс] / Режим доступа: <https://www.jetbrains.com/ru-ru/teamcity/>, свободный. (дата обращения: 16.05.2026).
9. CircleCI Documentation. [Электронный ресурс] / Режим доступа: <https://circleci.com/docs/>, свободный. (дата обращения: 14.04.2026).
10. Выпуск Java SE 23 и OpenJDK 23 [Электронный ресурс] // OpenNET. – Режим доступа: <https://www.opennet.dev/opennews/art.shtml?num=61884>, свободный. (дата обращения: 20.04.2026).

11. Tutorial: Get started with JUnit [Электронный ресурс] // IntelliJ IDEA. – URL: <https://www.jetbrains.com/idea/junit.html>, свободный. (дата обращения: 21.04.2026).

12. JUnit 5 User Guide [Электронный ресурс] // junit.org. – Режим доступа: <https://junit.org/junit5/docs/current/user-guide/>, свободный. (дата обращения: 20.04.2026).

13. Mockito: что это за фреймворк для Java, примеры использования [Электронный ресурс] // Skillfactory. – 2023. – 26 мар. – Режим доступа: <https://blog.skillfactory.ru/glossary/mockito/>, свободный. – (дата обращения: 12.05.2026).

14. Абернети, М. Введение в Swing [Электронный ресурс] // IBM / Майкл Абернети. – Режим доступа: <http://www.interface.ru/home.asp?artId=2686>, свободный. – (дата обращения: 16.05.2026).

15. СанПиН 1.2.3685-21 «Гигиенические нормативы и требования к обеспечению безопасности и (или) безвредности для человека факторов среды обитания» [Электронный ресурс]. – Режим доступа: <https://www.rospotrebnadzor.ru/>, свободный. (дата обращения: 18.05.2026).

Приложение А

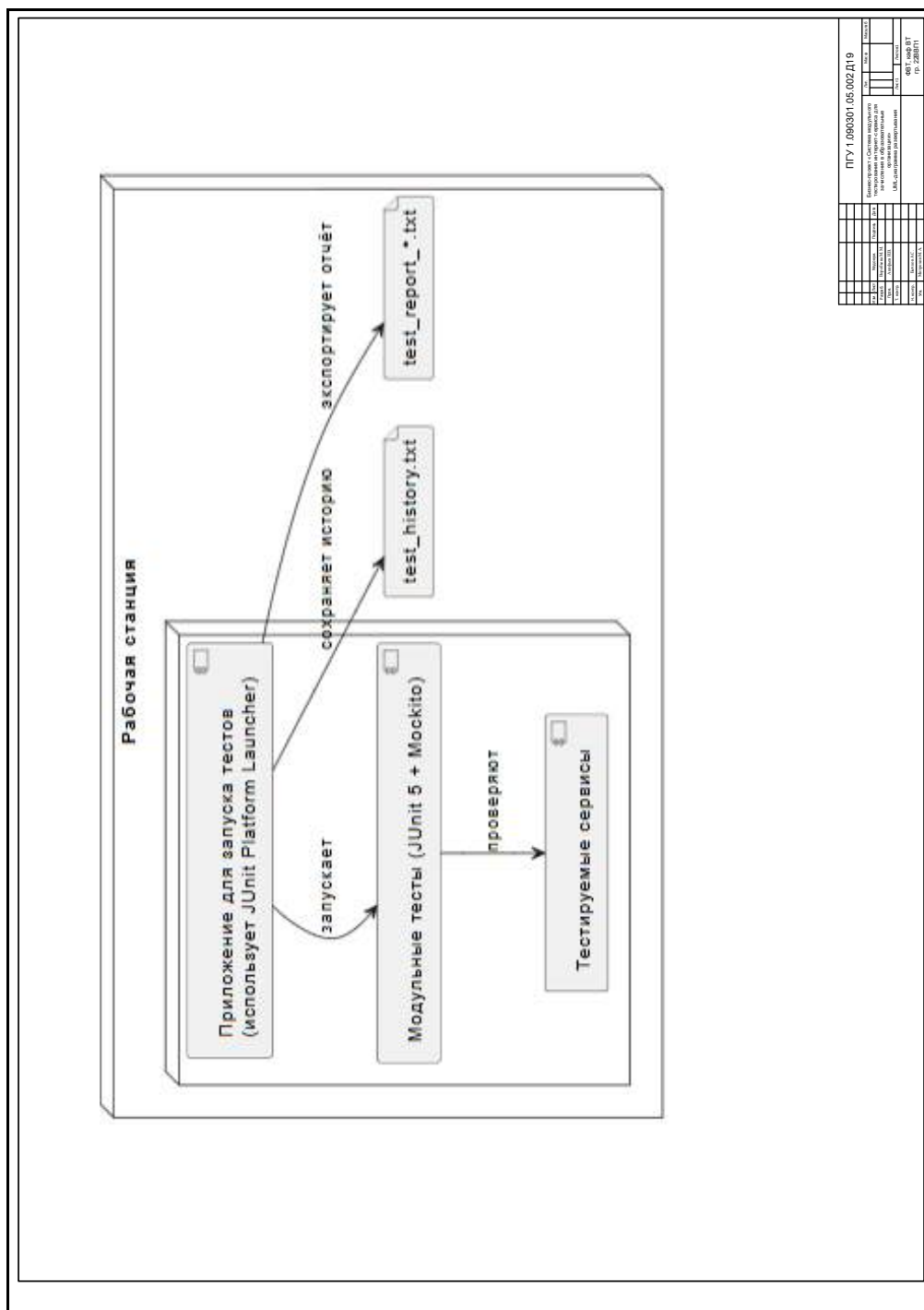
(обязательное)

Бизнес-проект «Система модульного тестирования интернет-сервиса для
зачисления в образовательные организации

UML-диаграммы

					ПГУ 1.090301.05.001 ПЗ	Лист
						84
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1. – Диаграмма развертывания



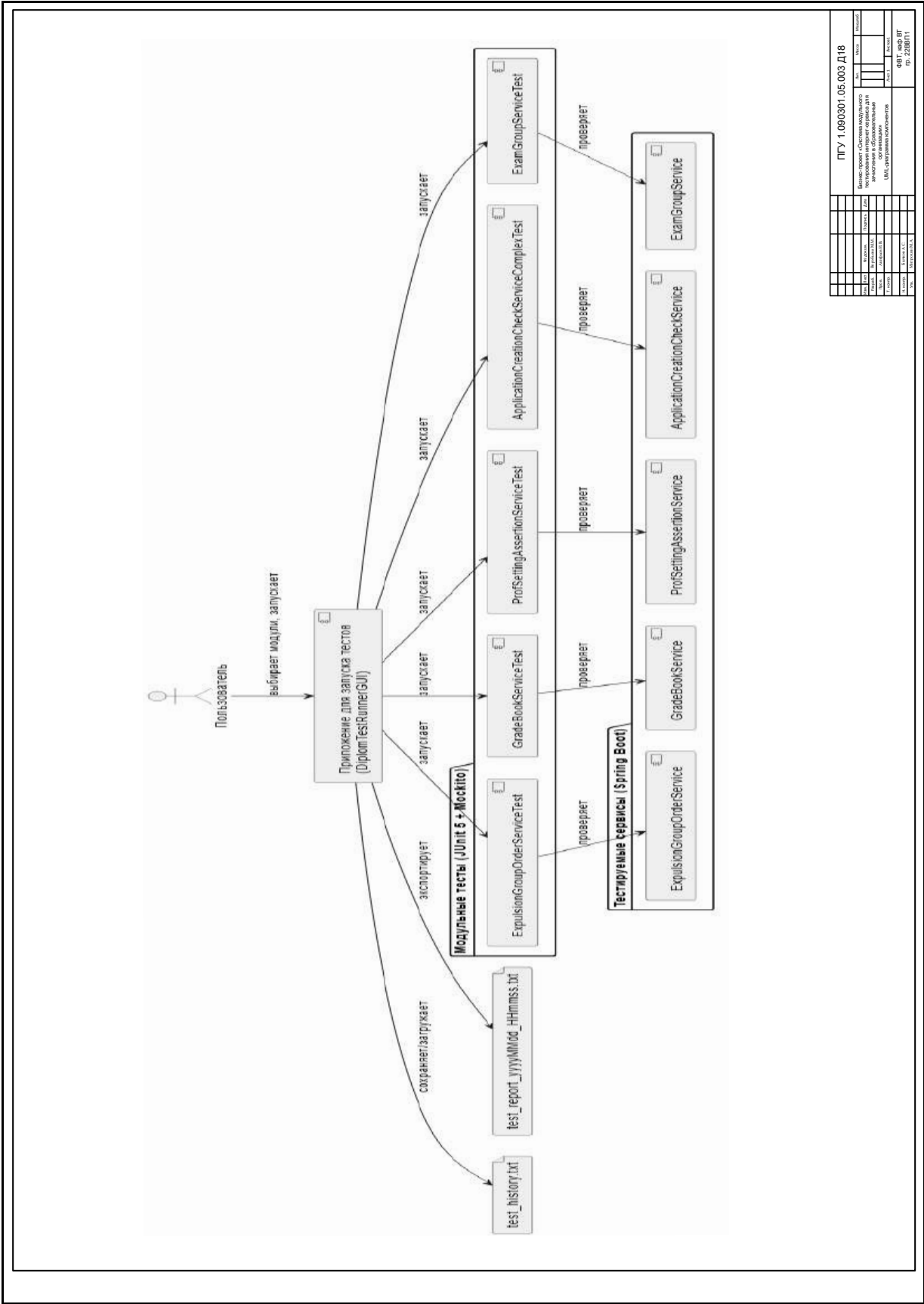
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

Лист

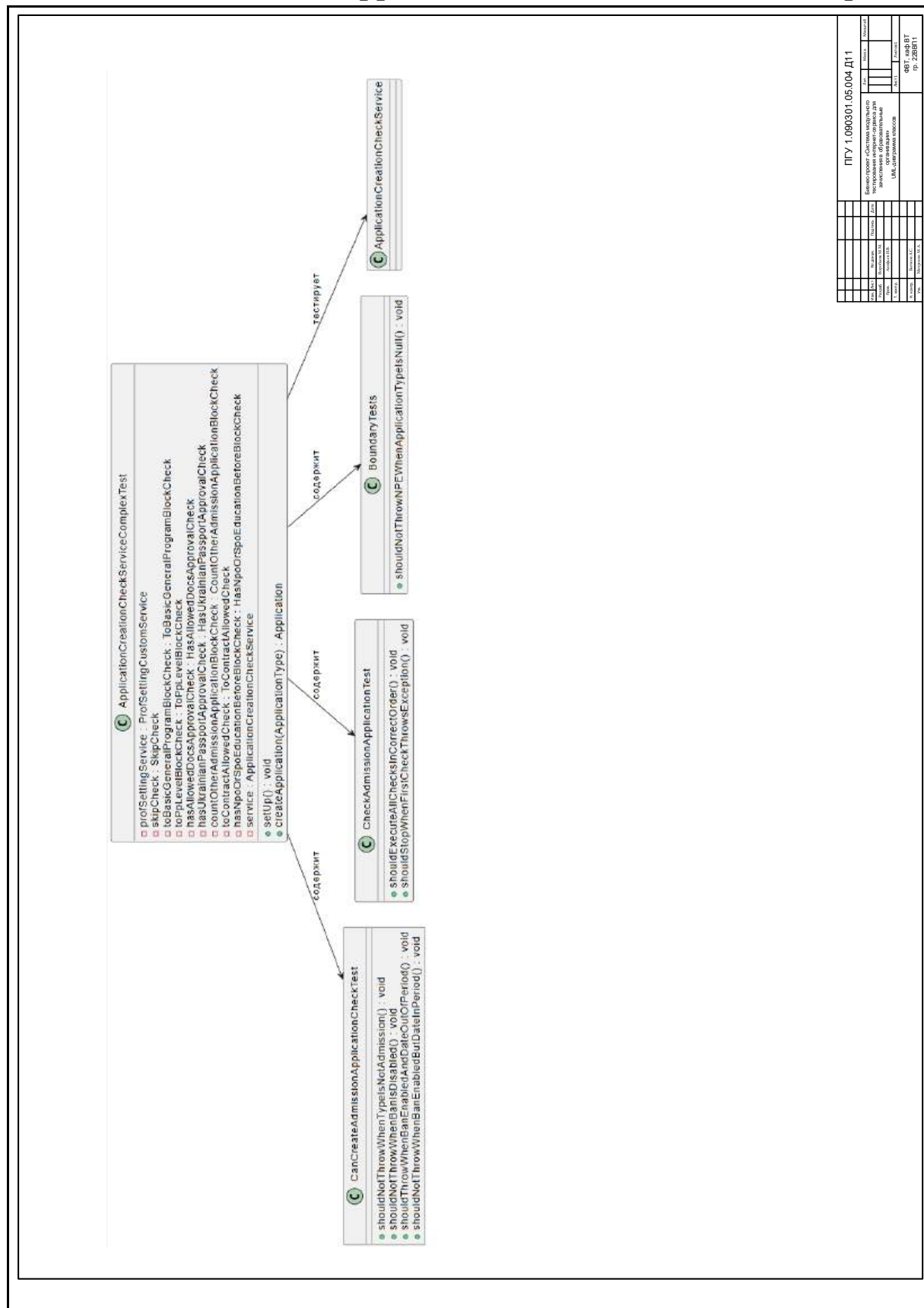
85

Приложение А.2 – Диаграмма компонентов системы модульного тестирования

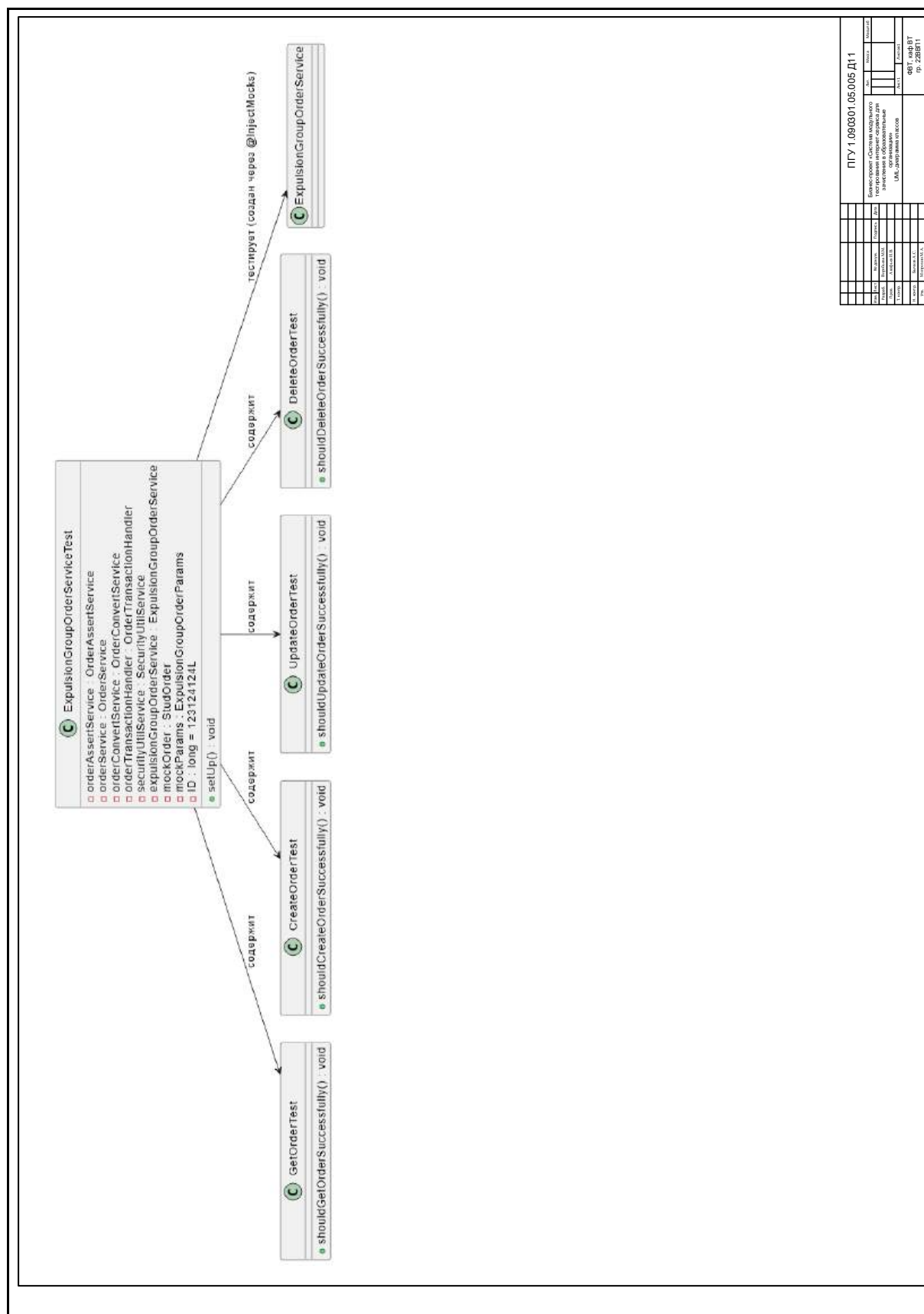


ПГУ 1.090301.05.003 Д18									
И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.
И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.
И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.
И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.
И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.
И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.
И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.
И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.
И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.	И.О.П.

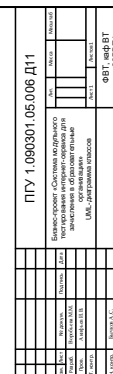
ApplicationCreationCheckServiceComplexTest



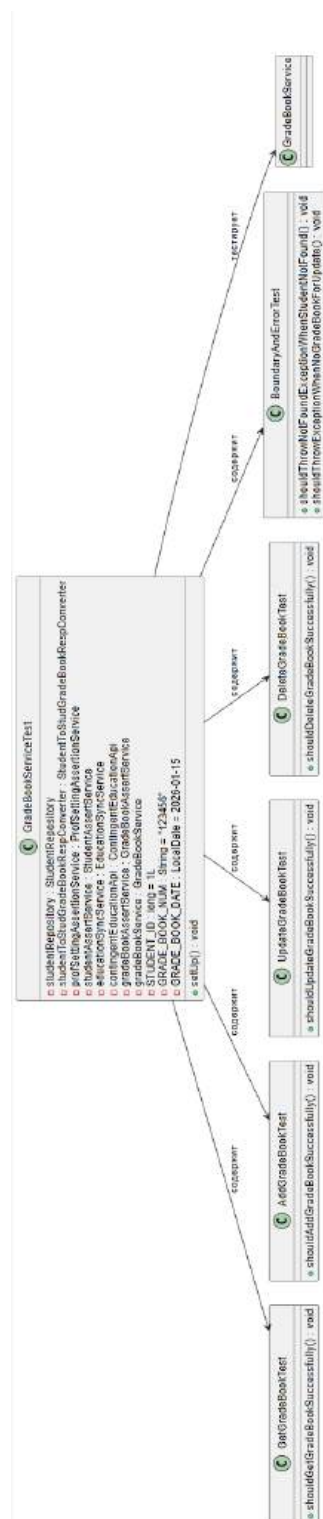
Приложение А.4 – Диаграмма классов для модуля ExpulsionGroupOrderServiceTest



ExamGroupServiceTest

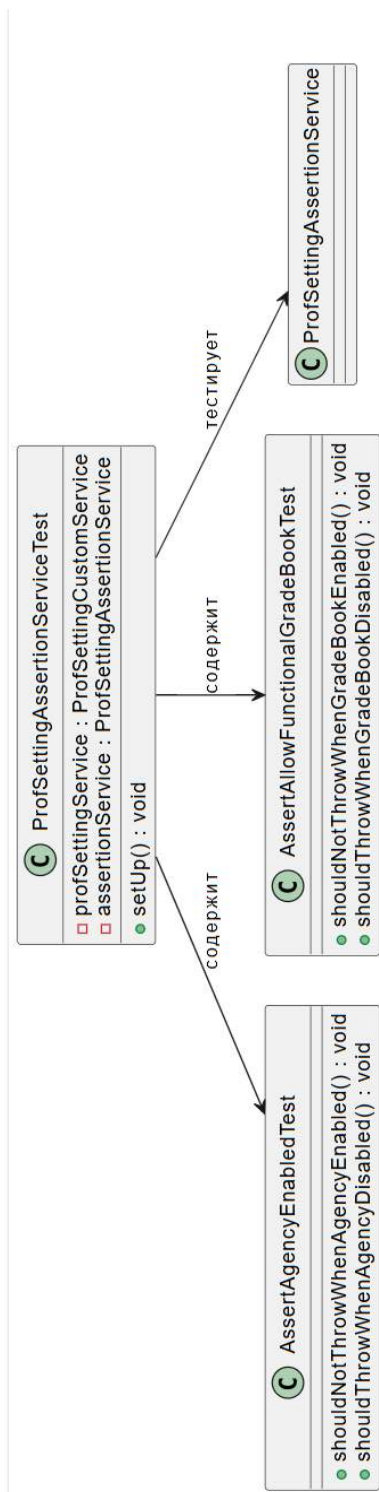


Приложение А.6 – Диаграмма классов тестового модуля GradeBookServiceTest

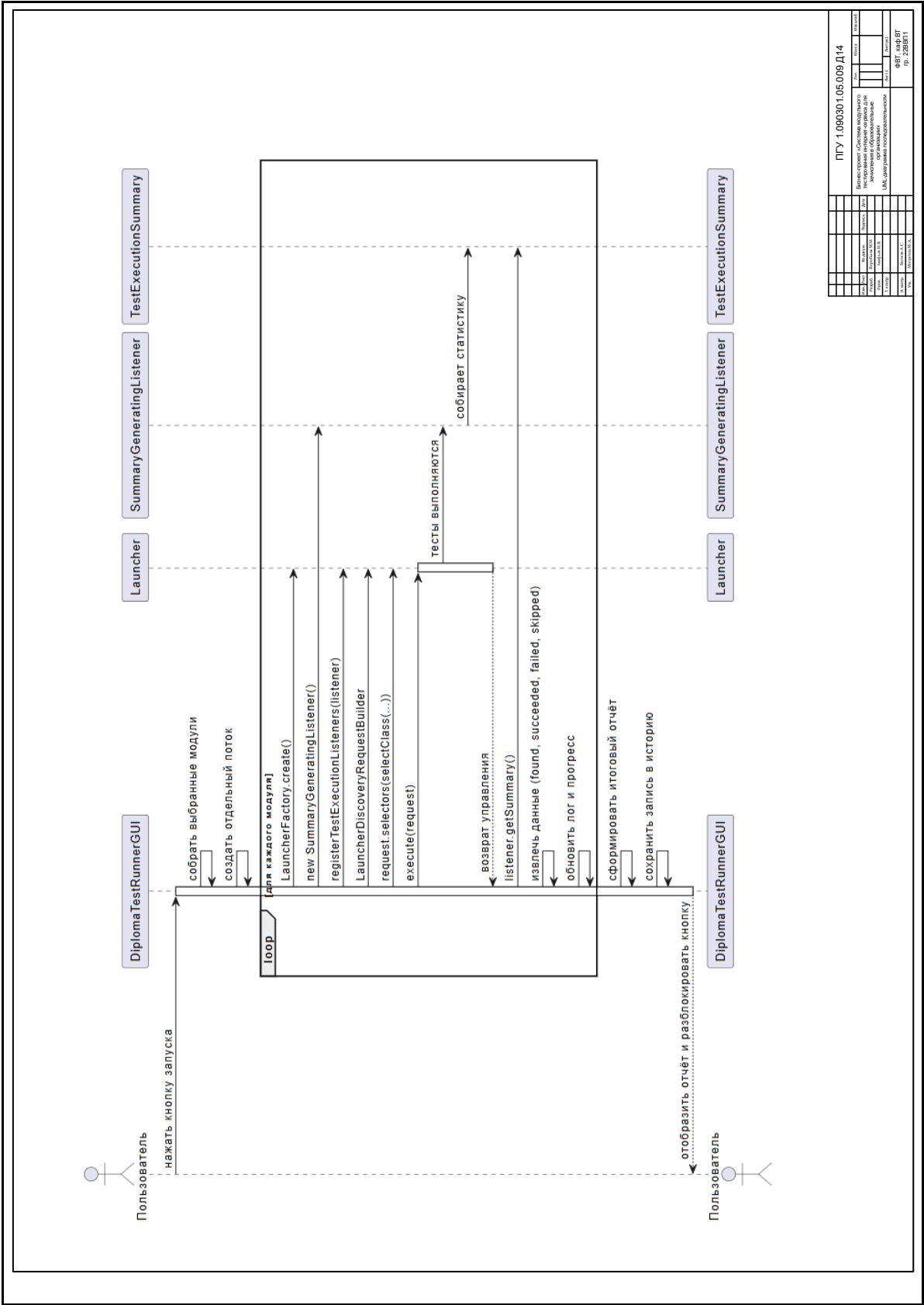


ПГУ 1.090301.05.007 Д11		ПГУ 1.090301.05.007 Д11	
Имя	Фамилия	Имя	Фамилия
Подпись	Подпись	Подпись	Подпись
Дата	Дата	Дата	Дата
ЧЕТ 108/11		ЧЕТ 108/11	
П. 228/11		П. 228/11	

ProfSettingAssertionServiceTest

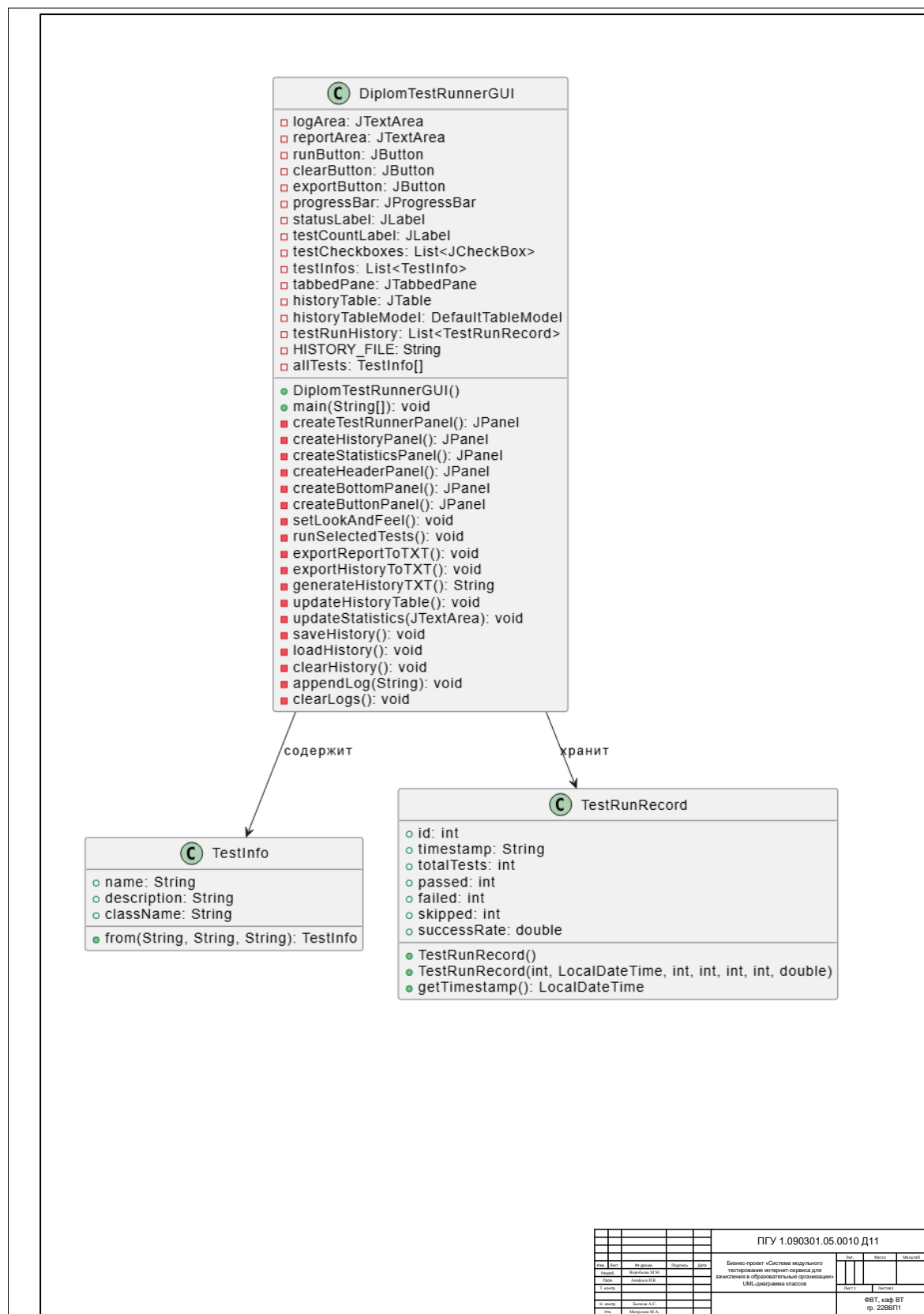
[illegible]

Приложение А.8 – Диаграмма последовательности для запуска тестов

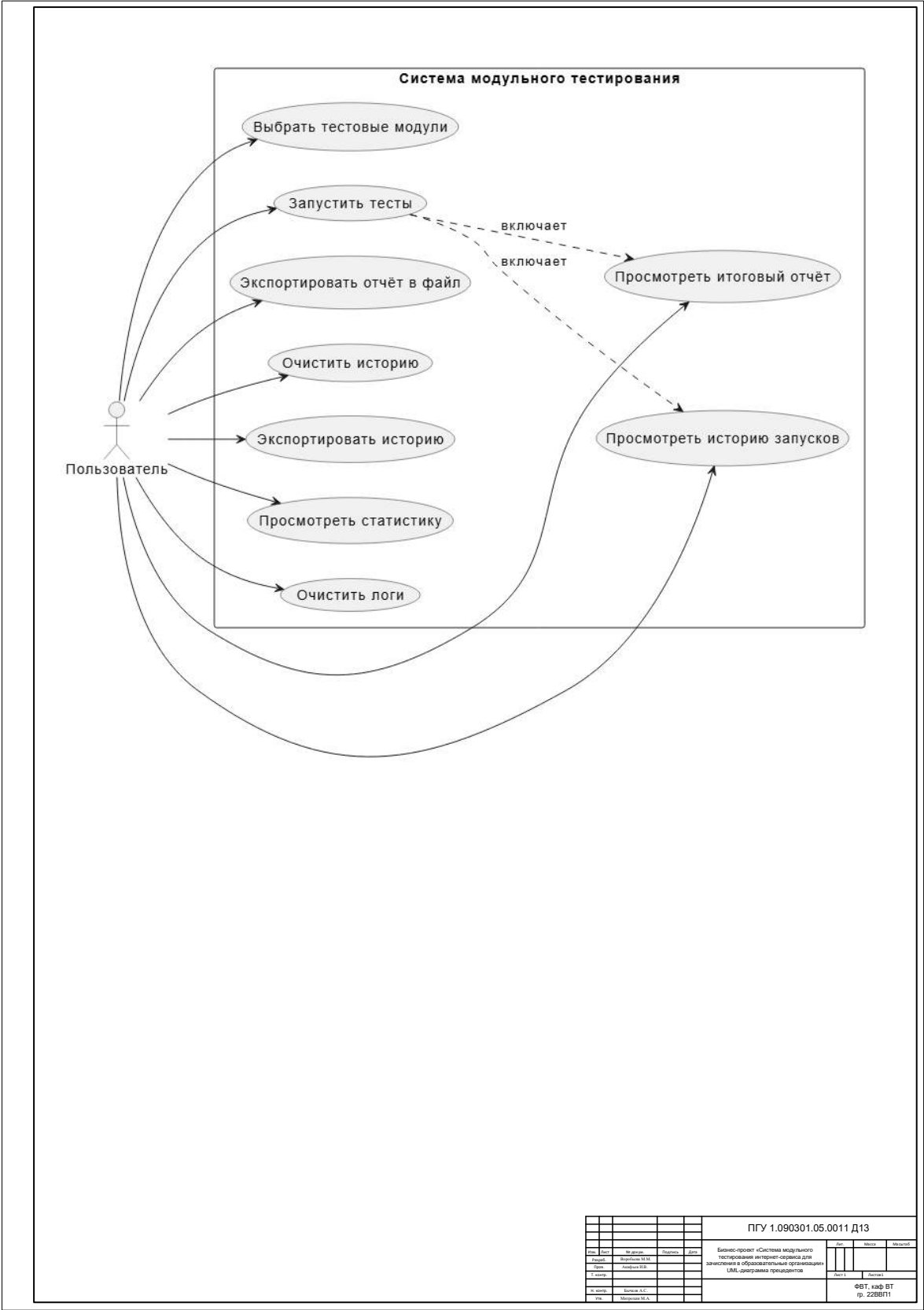


ПГУ 1.090301.05.009 Д14									
Изм.	Лист	№ докум.	Подпись	Дата	Вариант	Итого	Всего	Всего	Всего
1	1	1	1	1	1	1	1	1	1
Вариант-срок: Система модульного тестирования в образовательных учреждениях									
ИМ. разработчик: [подпись]									
ФИТ, 043 ВТ									
пр. 28ВЕТ1									

Приложение А.9 – Диаграмма классов приложения для запуска тестов



Приложение А.10 – Диаграмма вариантов использования системы модульного тестирования



ПГУ 1.090301.05.0011 Д13									
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия
Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия	Имя	Фамилия

Приложение Б
(обязательное)

Бизнес-проект «Система модульного тестирования интернет-сервиса для
зачисления в образовательные организации

Листинг программы

					ПГУ 1.090301.05.001 ПЗ	Лист
						95
Изм.	Лист	№ докум.	Подпись	Дата		

Файл DiplomTestRunnerGUI.java

```
package ru.mos.mes.profteh.backend.diplom;

import org.junit.platform.launcher.Launcher;
import org.junit.platform.launcher.LauncherDiscoveryRequest;
import org.junit.platform.launcher.core.LauncherDiscoveryRequestBuilder;
import org.junit.platform.launcher.core.LauncherFactory;
import org.junit.platform.launcher.listeners.SummaryGeneratingListener;
import org.junit.platform.launcher.listeners.TestExecutionSummary;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import javax.swing.border.TitledBorder;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;

import static
org.junit.platform.engine.discovery.DiscoverySelectors.selectClass;

public class DiplomTestRunnerGUI extends JFrame {

    private JTextArea logArea;
    private JTextArea reportArea;
    private JButton runButton;
    private JButton clearButton;
    private JButton exportButton;
    private JProgressBar progressBar;
    private JLabel statusLabel;
    private JLabel testCountLabel;
    private final List<JCheckBox> testCheckboxes = new ArrayList<>();
    private final List<TestInfo> testInfos = new ArrayList<>();
```

					ПГУ 1.090301.05.001 ПЗ	Лист
						96
Изм.	Лист	№ докум.	Подпись	Дата		

```

private JTabbedPane tabbedPane; //вкладки панель
private JTable historyTable;
private DefaultTableModel historyTableModel;
private List<TestRunRecord> testRunHistory = new ArrayList<>();
private static final String HISTORY_FILE = "test_history.txt";

private final TestInfo[] allTests = {
    TestInfo.from("Групповые приказы об отчислении",
        "Проверка создания, обновления, удаления и выпуска
приказов об отчислении",
"ru.mos.mes.profteh.backend.core.order.ExpulsionGroupOrderServiceTest"),
    TestInfo.from("Управление зачетными книжками",
        "Проверка получения, добавления, обновления и удаления
зачетных книжек студентов",
"ru.mos.mes.profteh.backend.core.student.service.GradeBookServiceTest"),
    TestInfo.from("Настройки функционала",
        "Проверка включения/выключения функционала
межведомственных проверок и зачетных книжек",
"ru.mos.mes.profteh.backend.core.student.service.ProfSettingAssertionServiceTest"),
    TestInfo.from("Заявления",
        "Проверка валидации заявлений: запрет создания, цепочка
проверок, восстановление",
"ru.mos.mes.profteh.backend.core.application.service.ApplicationCreationCheckServiceComplexTest"),
    TestInfo.from("Экзаменационные группы",
        "Проверка создания, удаления, закрытия групп и
выставления оценок",
"ru.mos.mes.profteh.backend.core.exam.service.ExamGroupServiceTest")
};

public DiplomTestRunnerGUI() {

```

```

        setTitle("Бизнес-проект «Система модульного тестирования интернет-
сервиса для зачисления в образовательные организации\n" +
                "\n");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(1100, 800);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout(10, 10));

        loadHistory();

        JPanel headerPanel = createHeaderPanel();
        add(headerPanel, BorderLayout.NORTH);

        tabbedPane = new JTabbedPane();
        tabbedPane.addTab("Запуск тестов", createTestRunnerPanel());
        tabbedPane.addTab("История запусков", createHistoryPanel());
        tabbedPane.addTab("Статистика", createStatisticsPanel());

        add(tabbedPane, BorderLayout.CENTER);

        JPanel buttonPanel = createButtonPanel();
        add(buttonPanel, BorderLayout.SOUTH);

        setLookAndFeel();
        updateHistoryTable();
    }

    private JPanel createTestRunnerPanel() {
        JPanel panel = new JPanel(new BorderLayout(10, 10));
        panel.setBorder(new EmptyBorder(10, 10, 10, 10));

        JPanel testListPanel = new JPanel(new BorderLayout());
        testListPanel.setBorder(new TitledBorder("Выберите модули для
тестирования"));

        JPanel checkBoxPanel = new JPanel(new GridLayout(allTests.length,
1, 5, 5));
        checkBoxPanel.setBorder(new EmptyBorder(10, 10, 10, 10));

```

```

for (TestInfo test : allTests) {
    JCheckBox checkBox = new JCheckBox(test.name);
    checkBox.setToolTipText(test.description);
    checkBox.setSelected(true);
    checkBoxPanel.add(checkBox);
    testCheckboxes.add(checkBox);
    testInfos.add(test);
}

JPanel controlPanel = new JPanel(new FlowLayout(FlowLayout.LEFT,
10, 5));
JButton selectAllButton = new JButton("Выбрать все");
JButton deselectAllButton = new JButton("Снять все");
selectAllButton.addActionListener(e -> testCheckboxes.forEach(cb ->
cb.setSelected(true)));
deselectAllButton.addActionListener(e -> testCheckboxes.forEach(cb
-> cb.setSelected(false)));
controlPanel.add(selectAllButton);
controlPanel.add(deselectAllButton);

JScrollPane checkBoxScroll = new JScrollPane(checkBoxPanel);
checkBoxScroll.setPreferredSize(new Dimension(400, 200));
testListPanel.add(checkBoxScroll, BorderLayout.CENTER);
testListPanel.add(controlPanel, BorderLayout.SOUTH);

JPanel logPanel = new JPanel(new BorderLayout());
logPanel.setBorder(new TitledBorder("Лог выполнения"));

logArea = new JTextArea();
logArea.setEditable(false);
logArea.setFont(new Font("Monospaced", Font.PLAIN, 11));
logArea.setBackground(Color.BLACK);
logArea.setForeground(Color.GREEN);
JScrollPane logScroll = new JScrollPane(logArea);
logPanel.add(logScroll, BorderLayout.CENTER);

JPanel statsMiniPanel = new JPanel(new GridLayout(3, 1, 5, 5));

```

```

statsMiniPanel.setBorder(new TitledBorder("Статус выполнения"));

statusLabel = new JLabel("Статус: Готов к запуску");
statusLabel.setFont(new Font("Arial", Font.PLAIN, 12));
testCountLabel = new JLabel("Выбрано модулей: 0");
testCountLabel.setFont(new Font("Arial", Font.PLAIN, 12));
progressBar = new JProgressBar(0, 100);
progressBar.setStringPainted(true);

statsMiniPanel.add(statusLabel);
statsMiniPanel.add(testCountLabel);
statsMiniPanel.add(progressBar);
logPanel.add(statsMiniPanel, BorderLayout.SOUTH);

JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
splitPane.setLeftComponent(testListPanel);
splitPane.setRightComponent(logPanel);
splitPane.setDividerLocation(400);

panel.add(splitPane, BorderLayout.CENTER);

JPanel reportPanel = createBottomPanel();
panel.add(reportPanel, BorderLayout.SOUTH);

return panel;
}

private JPanel createHistoryPanel() {
    JPanel panel = new JPanel(new BorderLayout(10, 10));
    panel.setBorder(new EmptyBorder(10, 10, 10, 10));

    String[] columns = {"N", "Дата и время", "Всего тестов",
"Пройдено", "Ошибок", "Успешность", "Тренд"};
    historyTableModel = new DefaultTableModel(columns, 0) {
        @Override
        public boolean isCellEditable(int row, int column) {
            return false;
        }
    }
}

```

```

    };
    historyTable = new JTable(historyTableModel);
    historyTable.setFont(new Font("Arial", Font.PLAIN, 12));
    historyTable.getTableHeader().setFont(new Font("Arial", Font.BOLD,
12));
    historyTable.setRowHeight(25);

    JScrollPane scrollPane = new JScrollPane(historyTable);
    scrollPane.setBorder(new TitledBorder("История запусков тестов"));
    panel.add(scrollPane, BorderLayout.CENTER);

    JPanel controlPanel = new JPanel(new FlowLayout(FlowLayout.LEFT,
10, 5));
    JButton clearHistoryButton = new JButton("Очистить историю");
    JButton exportHistoryButton = new JButton("Экспорт истории");

    clearHistoryButton.addActionListener(e -> clearHistory());
    exportHistoryButton.addActionListener(e -> exportHistoryToTXT());

    controlPanel.add(clearHistoryButton);
    controlPanel.add(exportHistoryButton);
    panel.add(controlPanel, BorderLayout.SOUTH);

    return panel;
}

private JPanel createStatisticsPanel() {
    JPanel panel = new JPanel(new BorderLayout(10, 10));
    panel.setBorder(new EmptyBorder(10, 10, 10, 10));

    JTextArea statsArea = new JTextArea();
    statsArea.setEditable(false);
    statsArea.setFont(new Font("Monospaced", Font.PLAIN, 12));
    statsArea.setBackground(new Color(240, 248, 255));

    updateStatistics(statsArea);

    JScrollPane scrollPane = new JScrollPane(statsArea);

```

```

        scrollPane.setBorder(new TitledBorder("Аналитика и статистика"));
        panel.add(scrollPane, BorderLayout.CENTER);

        JButton refreshStatsButton = new JButton("Обновить статистику");
        refreshStatsButton.addActionListener(e ->
updateStatistics(statsArea));

        JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.LEFT));
        buttonPanel.add(refreshStatsButton);
        panel.add(buttonPanel, BorderLayout.SOUTH);

        return panel;
    }

    private JPanel createHeaderPanel() {
        JPanel panel = new JPanel(new BorderLayout());
        panel.setBackground(new Color(33, 37, 41));
        panel.setBorder(new EmptyBorder(15, 15, 15, 15));

        JLabel titleLabel = new JLabel("Система модульного тестирования");
        titleLabel.setFont(new Font("Arial", Font.BOLD, 20));
        titleLabel.setForeground(Color.WHITE);

        JLabel subtitleLabel = new JLabel("Выпускная квалификационная
работа");
        subtitleLabel.setFont(new Font("Arial", Font.PLAIN, 12));
        subtitleLabel.setForeground(Color.LIGHT_GRAY);

        JPanel textPanel = new JPanel(new GridLayout(2, 1));
        textPanel.setOpaque(false);
        textPanel.add(titleLabel);
        textPanel.add(subtitleLabel);

        panel.add(textPanel, BorderLayout.WEST);

        JLabel dateLabel = new
JLabel(LocalDateTime.now().format(DateTimeFormatter.ofPattern("dd.MM.yyyy")
));

```

```

        dateLabel.setFont(new Font("Arial", Font.PLAIN, 12));
        dateLabel.setForeground(Color.LIGHT_GRAY);
        panel.add(dateLabel, BorderLayout.EAST);

        return panel;
    }

    private JPanel createBottomPanel() {
        JPanel panel = new JPanel(new BorderLayout());
        panel.setBorder(new TitledBorder("Итоговый отчёт"));

        reportArea = new JTextArea();
        reportArea.setEditable(false);
        reportArea.setFont(new Font("Monospaced", Font.PLAIN, 12));
        reportArea.setBackground(new Color(240, 248, 255));
        JScrollPane reportScroll = new JScrollPane(reportArea);
        reportScroll.setPreferredSize(new Dimension(800, 200));
        panel.add(reportScroll, BorderLayout.CENTER);

        return panel;
    }

    private JPanel createButtonPanel() {
        JPanel panel = new JPanel(new FlowLayout(FlowLayout.CENTER, 15,
10));
        panel.setBackground(new Color(240, 240, 240));

        runButton = new JButton("Запустить выбранные тесты");
        runButton.setFont(new Font("Arial", Font.BOLD, 14));
        runButton.setBackground(new Color(40, 167, 69));
        runButton.setForeground(Color.WHITE);
        runButton.setFocusPainted(false);
        runButton.setPreferredSize(new Dimension(250, 45));

        clearButton = new JButton("Очистить логи");
        clearButton.setFont(new Font("Arial", Font.PLAIN, 12));
        clearButton.setBackground(new Color(108, 117, 125));
        clearButton.setForeground(Color.WHITE);
    }

```

```

clearButton.setFocusPainted(false);

exportButton = new JButton("Экспорт отчёта");
exportButton.setFont(new Font("Arial", Font.PLAIN, 12));
exportButton.setBackground(new Color(0, 123, 255));
exportButton.setForeground(Color.WHITE);
exportButton.setFocusPainted(false);

JButton exitButton = new JButton("Выход");
exitButton.setFont(new Font("Arial", Font.PLAIN, 12));
exitButton.setBackground(new Color(220, 53, 69));
exitButton.setForeground(Color.WHITE);
exitButton.setFocusPainted(false);

runButton.addActionListener(e -> runSelectedTests());
clearButton.addActionListener(e -> clearLogs());
exportButton.addActionListener(e -> exportReportToTXT());
exitButton.addActionListener(e -> System.exit(0));

panel.add(runButton);
panel.add(clearButton);
panel.add(exportButton);
panel.add(exitButton);

return panel;
}

private void setLookAndFeel() {
    try {

        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    } catch (Exception e) {
        System.err.println("Ошибка установки темы: " + e.getMessage());
    }
}

private void runSelectedTests() {
    List<TestInfo> selectedTests = new ArrayList<>();

```

```

        for (int i = 0; i < testCheckboxes.size(); i++) {
            if (testCheckboxes.get(i).isSelected()) {
                selectedTests.add(testInfos.get(i));
            }
        }

        if (selectedTests.isEmpty()) {
            JOptionPane.showMessageDialog(this,
                "Не выбран ни один тест для запуска!",
                "Предупреждение",
                JOptionPane.WARNING_MESSAGE);
            return;
        }

        runButton.setEnabled(false);
        progressBar.setValue(0);
        progressBar.setString("Запуск...");
        statusLabel.setText("Статус: Выполнение...");
        testCountLabel.setText("Выбрано модулей: " + selectedTests.size());

        logArea.append("\n[" +
            LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")) + "]
=====\\n");

        logArea.append("[[" +
            LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")) + "]
ЗАПУСК ТЕСТОВ\\n");

        logArea.append("[[" +
            LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")) + "]
Выбрано модулей: " + selectedTests.size() + "\\n");

        logArea.append("[[" +
            LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")) + "]
=====\\n\\n");

        new Thread(() -> {
            int totalTests = 0;
            int totalPassed = 0;
            int totalFailed = 0;
            int totalSkipped = 0;

```

```

        boolean hasErrors = false;

        StringBuilder fullReport = new StringBuilder();

        fullReport.append("=====\n");
        fullReport.append("ИТОГОВЫЙ ОТЧЁТ О ВЫПОЛНЕНИИ ТЕСТОВ\n");

        fullReport.append("=====\n");
        fullReport.append("Дата и время:
") .append(LocalDate.now().format(DateTimeFormatter.ofPattern("dd.MM.yyy
у HH:mm:ss"))) .append("\n\n");

        for (int i = 0; i < selectedTests.size(); i++) {
            TestInfo info = selectedTests.get(i);
            appendLog("\n[" +
LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")) + "]"
Запуск: " + info.name);
            appendLog("    Описание: " + info.description);

            try {
                Launcher launcher = LauncherFactory.create();
                SummaryGeneratingListener listener = new
SummaryGeneratingListener();
                launcher.registerTestExecutionListeners(listener);

                LauncherDiscoveryRequest request =
LauncherDiscoveryRequestBuilder.request()

.selectors(selectClass(Class.forName(info.className)))
                .build();

                launcher.execute(request);
                TestExecutionSummary summary = listener.getSummary();

                totalTests += (int) summary.getTestsFoundCount();
                totalPassed += (int) summary.getTestsSucceededCount();
                totalFailed += (int) summary.getTestsFailedCount();
                totalSkipped += (int) summary.getTestsSkippedCount();
            }
        }
    }
}

```

```

        if (summary.getTestsFailedCount() > 0) {
            hasErrors = true;
        }

        appendLog("    Результат: " +
summary.getTestsSucceededCount() + "/" + summary.getTestsFoundCount() + "
пройдено");

        if (summary.getTestsFailedCount() > 0) {
            appendLog("    Ошибок: " +
summary.getTestsFailedCount());
            fullReport.append("X
").append(info.name).append("\n");
            fullReport.append("    Ошибок:
").append(summary.getTestsFailedCount()).append("\n");

            summary.getFailures().forEach(failure -> {
                fullReport.append("    -
").append(failure.getTestIdentifier().getDisplayName()).append("\n");
                fullReport.append("
").append(failure.getException().getMessage()).append("\n");
            });
        } else {
            fullReport.append("OK ").append(info.name).append("
- УСПЕШНО\n");
        }

    } catch (ClassNotFoundException e) {
        appendLog("    ОШИБКА: Класс не найден - " +
info.className);

        totalFailed++;
        hasErrors = true;
        fullReport.append("X ").append(info.name).append("
КЛАСС НЕ НАЙДЕН\n");
    } catch (Exception e) {
        appendLog("    ОШИБКА: " + e.getMessage());
        totalFailed++;
    }

```

```

        hasErrors = true;
        fullReport.append("X ").append(info.name).append(" -
ОШИБКА ВЫПОЛНЕНИЯ\n");
    }

    int progress = (int) (((double) (i + 1) /
selectedTests.size()) * 100);
    SwingUtilities.invokeLater(() -> {
        progressBar.setValue(progress);
        progressBar.setString(progress + "%");
    });
}

appendLog("\n[" +
LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")) + "]"
===== \n");

appendLog "[" +
LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")) + "]"
ВЫПОЛНЕНИЕ ЗАВЕРШЕНО\n");

appendLog "[" +
LocalDateTime.now().format(DateTimeFormatter.ofPattern("HH:mm:ss")) + "]"
===== \n");

fullReport.append("\n");

fullReport.append("===== \n");
fullReport.append("СТАТИСТИКА\n");

fullReport.append("===== \n");
fullReport.append("Всего тестов:
").append(totalTests).append("\n");
fullReport.append("Пройдено:
").append(totalPassed).append("\n");
fullReport.append("Ошибок: ").append(totalFailed).append("\n");
fullReport.append("Пропущено:
").append(totalSkipped).append("\n");
fullReport.append("Успешность: ").append(totalTests > 0 ?
(totalPassed * 100 / totalTests) : 0).append("%\n");

```

```

fullReport.append("=====\n");

int finalTotalTests = totalTests;
int finalTotalPassed = totalPassed;
int finalTotalFailed = totalFailed;
int finalTotalSkipped = totalSkipped;
String finalReport = fullReport.toString();
LocalDateTime runTime = LocalDateTime.now();

TestRunRecord record = new TestRunRecord(
    testRunHistory.size() + 1,
    runTime,
    finalTotalTests,
    finalTotalPassed,
    finalTotalFailed,
    finalTotalSkipped,
    finalTotalTests > 0 ? (finalTotalPassed * 100 /
finalTotalTests) : 0
);
testRunHistory.add(record);
saveHistory();
updateHistoryTable();
updateStatistics(null);

final boolean hasErrorsFinal = hasErrors;
SwingUtilities.invokeLater(() -> {
    reportArea.setText(finalReport);

    if (!hasErrorsFinal && finalTotalTests > 0) {
        statusLabel.setText("Статус: ВЫПОЛНЕНО УСПЕШНО");
        statusLabel.setForeground(new Color(40, 167, 69));
    } else if (finalTotalTests == 0) {
        statusLabel.setText("Статус: ТЕСТЫ НЕ ЗАПУЩЕНЫ");
        statusLabel.setForeground(new Color(255, 193, 7));
    } else {
        statusLabel.setText("Статус: ОШИБКА ВЫПОЛНЕНИЯ");
        statusLabel.setForeground(new Color(220, 53, 69));
    }
});

```

```

        }

        runButton.setEnabled(true);
        progressBar.setString("Готово");
    });

    }).start();
}

private void exportReportToTXT() {
    String reportText = reportArea.getText();
    if (reportText.isEmpty()) {
        JOptionPane.showMessageDialog(this,
            "Нет данных для экспорта. Сначала запустите тесты.",
            "Предупреждение",
            JOptionPane.WARNING_MESSAGE);
        return;
    }

    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setSelectedFile(new File("test_report_" +

LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyyMMdd_HHmmss"))
+ ".txt"));

    int result = fileChooser.showSaveDialog(this);
    if (result == JFileChooser.APPROVE_OPTION) {
        try (FileWriter fw = new
FileWriter(fileChooser.getSelectedFile())) {
            fw.write(reportText);
            JOptionPane.showMessageDialog(this,
                "Отчёт успешно сохранён!",
                "Успех",
                JOptionPane.INFORMATION_MESSAGE);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this,
                "Ошибка при сохранении: " + e.getMessage(),
                "Ошибка",

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						110
Изм.	Лист	№ докум.	Подпись	Дата		

```

        JOptionPane.ERROR_MESSAGE);
    }
}

private void exportHistoryToTXT() {
    if (testRunHistory.isEmpty()) {
        JOptionPane.showMessageDialog(this,
            "Нет данных для экспорта. История пуста.",
            "Предупреждение",
            JOptionPane.WARNING_MESSAGE);
        return;
    }

    JFileChooser fileChooser = new JFileChooser();
    fileChooser.setSelectedFile(new File("test_history_" +
        LocalDateTime.now().format(DateTimeFormatter.ofPattern("yyyyMMdd_HH:mm:ss"))
        + ".txt"));

    int result = fileChooser.showSaveDialog(this);
    if (result == JFileChooser.APPROVE_OPTION) {
        try (FileWriter fw = new
        FileWriter(fileChooser.getSelectedFile())) {
            fw.write(generateHistoryTXT());
            JOptionPane.showMessageDialog(this,
                "История успешно сохранена!",
                "Успех",
                JOptionPane.INFORMATION_MESSAGE);
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this,
                "Ошибка при сохранении: " + e.getMessage(),
                "Ошибка",
                JOptionPane.ERROR_MESSAGE);
        }
    }
}

```

```

private String generateHistoryTXT() {
    StringBuilder sb = new StringBuilder();
    sb.append("ИСТОРИЯ ЗАПУСКОВ ТЕСТОВ\n");

    for (int i = testRunHistory.size() - 1; i >= 0; i--) {
        TestRunRecord r = testRunHistory.get(i);
        sb.append("Запуск N").append(r.id).append("\n");
        sb.append("    Дата и время:
").append(r.getTimestamp().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss"))).append("\n");
        sb.append("    Всего тестов:
").append(r.totalTests).append("\n");
        sb.append("    Пройдено: ").append(r.passed).append("\n");
        sb.append("    Ошибок: ").append(r.failed).append("\n");
        sb.append("    Пропущено: ").append(r.skipped).append("\n");
        sb.append("    Успешность:
").append(r.successRate).append("%\n");
        sb.append("    Тренд: ");

        if (i > 0) {
            TestRunRecord prev = testRunHistory.get(i - 1);
            if (r.successRate > prev.successRate) {
                sb.append("Улучшение\n");
            } else if (r.successRate < prev.successRate) {
                sb.append("Ухудшение\n");
            } else {
                sb.append("Стабильно\n");
            }
        } else {
            sb.append("-\n");
        }
        sb.append("\n");
    }

    sb.append("Всего запусков:
").append(testRunHistory.size()).append("\n");

```

```

        return sb.toString();
    }

    private void updateHistoryTable() {
        SwingUtilities.invokeLater(() -> {
            historyTableModel.setRowCount(0);
            for (int i = testRunHistory.size() - 1; i >= 0; i--) {
                TestRunRecord r = testRunHistory.get(i);
                String trend = "";
                if (i > 0) {
                    TestRunRecord prev = testRunHistory.get(i - 1);
                    if (r.successRate > prev.successRate) trend = "↑";
                    else if (r.successRate < prev.successRate) trend = "↓";
                    else trend = "→";
                } else {
                    trend = "-";
                }
                historyTableModel.addRow(new Object[]{
                    r.id,

r.getTimestamp().format(DateTimeFormatter.ofPattern("dd.MM.yyyy
HH:mm:ss")),

                    r.totalTests,
                    r.passed,
                    r.failed,
                    r.successRate + "%",
                    trend

                });
            }
        });
    }

    private void updateStatistics(JTextArea statsArea) {
        if (testRunHistory.isEmpty()) return;

        double avgSuccessRate = testRunHistory.stream().mapToDouble(r ->
r.successRate).average().orElse(0);

```

```

        TestRunRecord best = testRunHistory.stream().max((a, b) ->
Double.compare(a.successRate, b.successRate)).orElse(null);

        TestRunRecord worst = testRunHistory.stream().min((a, b) ->
Double.compare(a.successRate, b.successRate)).orElse(null);

        int totalTestsRun = testRunHistory.stream().mapToInt(r ->
r.totalTests).sum();

        int totalPassed = testRunHistory.stream().mapToInt(r ->
r.passed).sum();

        String stats =

                "
                СТАТИСТИКА ТЕСТИРОВАНИЯ

\n" +

                "
                Всего запусков:
                " +
testRunHistory.size() + "\n" +

                "
                Всего выполнено тестов:
                " +
totalTestsRun + "\n" +

                "
                Всего пройдено:
                " +
totalPassed + "\n" +

                "
                Средняя успешность:
                " +
String.format("%.1f", avgSuccessRate) + "%\n" +

                "
                Лучший результат:
                " +
String.format("%.1f", best != null ? best.successRate : 0) + "%\n" +

                "
                Худший результат:
                " +
String.format("%.1f", worst != null ? worst.successRate : 0) + "%\n" ;

        if (statsArea != null) {
            SwingUtilities.invokeLater(() -> statsArea.setText(stats));
        }
    }

    private void saveHistory() {
        try (FileWriter fw = new FileWriter(HISTORY_FILE)) {
            for (TestRunRecord record : testRunHistory) {
                fw.write(record.id + "|" +
                    record.timestamp + "|" +

```

```

        record.totalTests + "|" +
        record.passed + "|" +
        record.failed + "|" +
        record.skipped + "|" +
        record.successRate + "\n");
    }
} catch (IOException e) {
    System.err.println("Ошибка сохранения истории: " +
e.getMessage());
}
}

private void loadHistory() {
    testRunHistory = new ArrayList<>();
    try {
        File file = new File(HISTORY_FILE);
        if (file.exists()) {
            List<String> lines =
Files.readAllLines(Paths.get(HISTORY_FILE));
            for (String line : lines) {
                String[] parts = line.split("\\|");
                if (parts.length == 7) {
                    TestRunRecord record = new TestRunRecord();
                    record.id = Integer.parseInt(parts[0]);
                    record.timestamp = parts[1];
                    record.totalTests = Integer.parseInt(parts[2]);
                    record.passed = Integer.parseInt(parts[3]);
                    record.failed = Integer.parseInt(parts[4]);
                    record.skipped = Integer.parseInt(parts[5]);
                    record.successRate = Double.parseDouble(parts[6]);
                    testRunHistory.add(record);
                }
            }
        }
    } catch (IOException e) {
        System.err.println("Ошибка загрузки истории: " +
e.getMessage());
        testRunHistory = new ArrayList<>();
    }
}

```

```

    }

}

private void clearHistory() {
    int confirm = JOptionPane.showConfirmDialog(this,
        "Вы уверены, что хотите очистить всю историю?",
        "Подтверждение",
        JOptionPane.YES_NO_OPTION);
    if (confirm == JOptionPane.YES_OPTION) {
        testRunHistory.clear();
        saveHistory();
        updateHistoryTable();
        updateStatistics(null);
        appendLog("История запусков очищена.");
    }
}

private void appendLog(String message) {
    SwingUtilities.invokeLater(() -> {
        logArea.append(message + "\n");
        logArea.setCaretPosition(logArea.getDocument().getLength());
    });
}

private void clearLogs() {
    logArea.setText("");
    reportArea.setText("");
    statusLabel.setText("Статус: Готов к запуску");
    statusLabel.setForeground(Color.BLACK);
    testCountLabel.setText("Выбрано модулей: 0");
    progressBar.setValue(0);
    progressBar.setString("0%");
    appendLog("Логи очищены. Готов к запуску.");
}

private static class TestInfo {
    String name;
    String description;
}

```

```

        String className;

        static TestInfo from(String name, String description, String
className) {
            TestInfo info = new TestInfo();
            info.name = name;
            info.description = description;
            info.className = className;
            return info;
        }
    }

    private static class TestRunRecord {
        int id;
        String timestamp;
        int totalTests;
        int passed;
        int failed;
        int skipped;
        double successRate;

        TestRunRecord() {}

        TestRunRecord(int id, LocalDateTime timestamp, int totalTests, int
passed, int failed, int skipped, double successRate) {
            this.id = id;
            this.timestamp = timestamp.toString();
            this.totalTests = totalTests;
            this.passed = passed;
            this.failed = failed;
            this.skipped = skipped;
            this.successRate = successRate;
        }

        LocalDateTime getTimestamp() {
            return LocalDateTime.parse(timestamp);
        }
    }
}

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						117
Изм.	Лист	№ докум.	Подпись	Дата		

```

        public static void main(String[] args) {
            SwingUtilities.invokeLater(() -> {
                DiplomTestRunnerGUI gui = new DiplomTestRunnerGUI();
                gui.setVisible(true);
            });
        }
    }
}

```

Файл ApplicationCreationCheckServiceComplexTest.java

```

package ru.mos.mes.profteh.backend.core.application.service;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InOrder;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import ru.mos.mes.profteh.backend.core.application.service.check.*;
import
ru.mos.mes.profteh.backend.core.profsetting.ProfSettingCustomService;
import ru.mos.mes.profteh.core.application.domain.Application;
import ru.mos.mes.profteh.core.application.domain.ApplicationType;
import ru.mos.mes.profteh.core.exception.ApplicationException;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.*;
import static ru.mos.mes.profteh.core.application.domain.ApplicationType.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("ApplicationCreationCheckService - unit-тесты")
class ApplicationCreationCheckServiceComplexTest {

    @Mock
    private ProfSettingCustomService profSettingService;

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						118
Изм.	Лист	№ докум.	Подпись	Дата		

```

    @Mock
    private SkipCheck skipCheck;

    @Mock
    private ToBasicGeneralProgramBlockCheck
toBasicGeneralProgramBlockCheck;

    @Mock
    private ToPpLevelBlockCheck toPpLevelBlockCheck;

    @Mock
    private HasAllowedDocsApprovalCheck hasAllowedDocsApprovalCheck;

    @Mock
    private HasUkrainianPassportApprovalCheck
hasUkrainianPassportApprovalCheck;

    @Mock
    private CountOtherAdmissionApplicationBlockCheck
countOtherAdmissionApplicationBlockCheck;

    @Mock
    private ToContractAllowedCheck toContractAllowedCheck;

    @Mock
    private HasNpoOrSpoEducationBeforeBlockCheck
hasNpoOrSpoEducationBeforeBlockCheck;

    private ApplicationCreationCheckService service;

    @BeforeEach
    void setUp() {
        service = new ApplicationCreationCheckService(
            profSettingService,
            skipCheck,
            toBasicGeneralProgramBlockCheck,
            toPpLevelBlockCheck,
            hasAllowedDocsApprovalCheck,

```

```

        hasUkrainianPassportApprovalCheck,
        countOtherAdmissionApplicationBlockCheck,
        toContractAllowedCheck,
        hasNpoOrSpoEducationBeforeBlockCheck
    );
}

@Nested
@DisplayName("canCreateAdmissionApplicationCheck - проверка создания
заявления")
class CanCreateAdmissionApplicationCheckTest {

    @Test
    @DisplayName("Не бросает исключение, если тип заявления не
ADMISSION")
    void shouldNotThrowWhenTypeIsNotAdmission() {
        Application application = createApplication(FINANCING_CHANGE);

        assertDoesNotThrow(() ->
service.canCreateAdmissionApplicationCheck(application));
        verify(profSettingService, never()).isBanAppAdmission();
    }

    @Test
    @DisplayName("Не бросает исключение, если бан отключен")
    void shouldNotThrowWhenBanIsDisabled() {
        Application application = createApplication(ADMISSION);
        when(profSettingService.isBanAppAdmission()).thenReturn(false);

        assertDoesNotThrow(() ->
service.canCreateAdmissionApplicationCheck(application));
    }

    @Test
    @DisplayName("Бросает исключение, если бан включен И дата вне
периода")
    void shouldThrowWhenBanEnabledAndDateOutOfPeriod() {
        Application application = createApplication(ADMISSION);

```

```

        when (profSettingService.isBanAppAdmission()) .thenReturn(true);

when (profSettingService.isCurrentDateOutOfAppAdmissionDates()) .thenReturn(true);

        assertThrows(ApplicationException.class,
                        () ->
service.canCreateAdmissionApplicationCheck(application));
    }

@Test
    @DisplayName("Не бросает исключение, если бан включен, но дата в пределах периода")
    void shouldNotThrowWhenBanEnabledButDateInPeriod() {
        Application application = createApplication(ADMISSION);
        when (profSettingService.isBanAppAdmission()) .thenReturn(true);

when (profSettingService.isCurrentDateOutOfAppAdmissionDates()) .thenReturn(false);

        assertDoesNotThrow(() ->
service.canCreateAdmissionApplicationCheck(application));
    }
}

@Nested
    @DisplayName("checkAdmissionApplication - цепочка проверок")
    class CheckAdmissionApplicationTest {

        @Test
        @DisplayName("Вызывает все проверки в правильном порядке")
        void shouldExecuteAllChecksInCorrectOrder() {
            Application application = createApplication(ADMISSION);

            service.checkAdmissionApplication(application);

            InOrder inOrder = inOrder(
                skipCheck,

```

```

        toBasicGeneralProgramBlockCheck,
        toPpLevelBlockCheck,
        hasAllowedDocsApprovalCheck,
        hasUkrainianPassportApprovalCheck,
        countOtherAdmissionApplicationBlockCheck,
        toContractAllowedCheck,
        hasNpoOrSpoEducationBeforeBlockCheck
    );

    inOrder.verify(skipCheck).check(application);

inOrder.verify(toBasicGeneralProgramBlockCheck).check(application);
    inOrder.verify(toPpLevelBlockCheck).check(application);
    inOrder.verify(hasAllowedDocsApprovalCheck).check(application);

inOrder.verify(hasUkrainianPassportApprovalCheck).check(application);

inOrder.verify(countOtherAdmissionApplicationBlockCheck).check(application)
;

    inOrder.verify(toContractAllowedCheck).check(application);

inOrder.verify(hasNpoOrSpoEducationBeforeBlockCheck).check(application);
}

@Test
@DisplayName("При исключении в проверке остальные не вызываются")
void shouldStopWhenFirstCheckThrowsException() {
    Application application = createApplication(ADMISSION);
    doThrow(new ApplicationException("Ошибка"))
        .when(toPpLevelBlockCheck).check(application);

    assertThrows(ApplicationException.class,
        () -> service.checkAdmissionApplication(application));

    verify(skipCheck).check(application);
    verify(toBasicGeneralProgramBlockCheck).check(application);
    verify(toPpLevelBlockCheck).check(application);
    verify(hasAllowedDocsApprovalCheck, never()).check(any());

```

```

    }

}

private Application createApplication(ApplicationType type) {
    Application application = new Application();
    application.setType(type);
    return application;
}

@Nested
@DisplayName("Граничные случаи")
class BoundaryTests {

    @Test
    @DisplayName("Заявление без типа не должно вызывать NPE в canCreateAdmissionApplicationCheck")
    void shouldNotThrowNPEWhenApplicationTypeIsNull() {
        Application application = new Application();

        assertDoesNotThrow(() ->
service.canCreateAdmissionApplicationCheck(application));
    }

}
}

```

Файл ExpulsionGroupOrderServiceTest.java

```

package ru.mos.mes.profteh.backend.core.order;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.InjectMocks;

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						123
Изм.	Лист	№ докум.	Подпись	Дата		

```

import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import
ru.mos.mes.profteh.backend.core.order.model.expulsion.ExpulsionGroupOrderCo
ntentResponse;
import
ru.mos.mes.profteh.backend.core.order.model.expulsion.ExpulsionGroupOrderPa
rams;
import
ru.mos.mes.profteh.backend.core.order.model.expulsion.ExpulsionGroupOrderRe
sult;
import ru.mos.mes.profteh.backend.core.order.service.*;
import ru.mos.mes.profteh.backend.core.system.util.SecurityUtilService;
import ru.mos.mes.profteh.core.order.domain.OrderType;
import ru.mos.mes.profteh.core.order.domain.StudOrder;

import java.util.ArrayList;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("ExpulsionGroupOrderService - unit-тесты")
public class ExpulsionGroupOrderServiceTest {

    private static final long ID = 123124124L;

    @Mock
    private OrderAssertService orderAssertService;

    @Mock
    private OrderService orderService;

    @Mock
    private OrderConvertService orderConvertService;

    @Mock
    private OrderTransactionHandler orderTransactionHandler;

    @Mock
    private SecurityUtilService securityUtilService;

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						124
Изм.	Лист	№ докум.	Подпись	Дата		

```

@InjectMocks
private ExpulsionGroupOrderService expulsionGroupOrderService;

private StudOrder mockOrder;
private ExpulsionGroupOrderParams mockParams;

@BeforeEach
void setUp() {
    mockOrder = mock(StudOrder.class);
    mockParams = mock(ExpulsionGroupOrderParams.class);
}

@Nested
@DisplayName("getOrder - получение приказа об отчислении")
class GetOrderTest {

    @Test
    @DisplayName("Успешно получает приказ и вызывает конвертацию")
    void shouldGetOrderSuccessfully() {
        when(mockOrder.getOrderItems()).thenReturn(new ArrayList<>());
        ExpulsionGroupOrderContentResponse expectedResponse =
mock(ExpulsionGroupOrderContentResponse.class);

        when(orderService.findOrderByIdAndType(ID,
OrderType.EXPULSION_GROUP)).thenReturn(mockOrder);

when(orderConvertService.toExpulsionGroupOrderContentResponse(mockOrder,
new ArrayList<>()))
        .thenReturn(expectedResponse);

doNothing().when(securityUtilService).securityCheckOrder(mockOrder);

        ExpulsionGroupOrderContentResponse actual =
expulsionGroupOrderService.getOrder(ID);

        assertNotNull(actual);
        assertEquals(expectedResponse, actual);
    }
}

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						125
Изм.	Лист	№ докум.	Подпись	Дата		

```

verify(orderConvertService).toExpulsionGroupOrderContentResponse(mockOrder,
new ArrayList<>());
    }
}

@Nested
@DisplayName("createOrder - создание приказа об отчислении")
class CreateOrderTest {

    @Test
    @DisplayName("Успешно создаёт приказ через транзакционный
обработчик")
    void shouldCreateOrderSuccessfully() {
        ExpulsionGroupOrderResult expectedResponse =
mock(ExpulsionGroupOrderResult.class);

when(orderTransactionHandler.runInTransaction(any())) .thenReturn(expectedRe
sponse);

        ExpulsionGroupOrderResult actual =
expulsionGroupOrderService.createOrder(mockParams);

        assertNotNull(actual);
        assertEquals(expectedResponse, actual);
        verify(orderTransactionHandler).runInTransaction(any());
    }
}

@Nested
@DisplayName("updateOrder - обновление приказа об отчислении")
class UpdateOrderTest {

    @Test
    @DisplayName("Успешно обновляет приказ, сначала проверив статус,
через транзакцию")
    void shouldUpdateOrderSuccessfully() {

```

```

        ExpulsionGroupOrderResult expectedResponse =
mock(ExpulsionGroupOrderResult.class);

        when(orderService.findOrderByIdAndType(ID,
OrderType.EXPULSION_GROUP)).thenReturn(mockOrder);

when(orderTransactionHandler.runInTransaction(any())) .thenReturn(expectedRe
sponse);

        ExpulsionGroupOrderResult actual =
expulsionGroupOrderService.updateOrder(ID, mockParams);

        assertNotNull(actual);
        assertEquals(expectedResponse, actual);

verify(orderAssertService).assertOrderStatusForUpdate(mockOrder);
        verify(orderTransactionHandler).runInTransaction(any());
    }
}

@Nested
@DisplayName("deleteOrder - удаление приказа об отчислении")
class DeleteOrderTest {

    @Test
    @DisplayName("Успешно удаляет приказ, проверив статус, через
транзакцию")
    void shouldDeleteOrderSuccessfully() {
        when(orderService.findOrderByIdAndType(ID,
OrderType.EXPULSION_GROUP)).thenReturn(mockOrder);

        assertDoesNotThrow(() ->
expulsionGroupOrderService.deleteOrder(ID));

verify(orderAssertService).assertOrderStatusForDelete(mockOrder);
        verify(orderTransactionHandler).runInTransaction(any());
    }
}

```

```

    }
}

```

Файл ExamGroupServiceTest.java

```

package ru.mos.mes.profteh.backend.core.exam.service;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import org.springframework.data.jpa.domain.Specification;
import
ru.mos.mes.profteh.backend.core.application.domain.ApplicationToExamGroup;
import
ru.mos.mes.profteh.backend.core.application.repo.ApplicationCustomRepository;
import
ru.mos.mes.profteh.backend.core.application.repo.ApplicationToExamGroupRepository;
import
ru.mos.mes.profteh.backend.core.classifier.service.ClassifierService;
import ru.mos.mes.profteh.backend.core.exam.converter.*;
import ru.mos.mes.profteh.backend.core.exam.domain.ExamGroup;
import ru.mos.mes.profteh.backend.core.exam.domain.ExamGroupStatus;
import ru.mos.mes.profteh.backend.core.exam.model.*;
import
ru.mos.mes.profteh.backend.core.exam.repo.ApplicationExamMarkViewRepo;
import ru.mos.mes.profteh.backend.core.exam.repo.ExamGroupRegistryViewRepo;
import ru.mos.mes.profteh.backend.core.exam.repo.ExamGroupRepo;
import ru.mos.mes.profteh.backend.error.ExamGroupNotFoundException;
import
ru.mos.mes.profteh.backend.sync.etpmv.service.StatusNotificationService;
import ru.mos.mes.profteh.core.application.domain.Application;
import ru.mos.mes.profteh.core.exception.ApplicationException;

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						128
Изм.	Лист	№ докум.	Подпись	Дата		

```

import java.time.LocalDateTime;
import java.util.Collections;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("ExamGroupService - unit-тесты")
class ExamGroupServiceTest {

    private static final long EXAM_GROUP_ID = 1L;
    private static final String GROUP_NAME = "Экзаменационная группа 1";
    private static final LocalDateTime EXAM_DATE =
LocalDateTime.now().plusDays(7);

    @Mock
    private ExamGroupRegistryViewRepo examGroupRegistryViewRepo;

    @Mock
    private ExamGroupRegistryViewToSearchRespConverter
examGroupRegistryViewToSearchRespConverter;

    @Mock
    private ExamGroupRepo examGroupRepo;

    @Mock
    private ExamGroupToRespConverter examGroupToRespConverter;

    @Mock
    private ExamGroupReqToEntityConverter examGroupReqToEntityConverter;

    @Mock
    private ApplicationToExamGroupRepo applicationToExamGroupRepo;

    @Mock
    private ApplicationExamMarkViewRepo applicationExamMarkViewRepo;

```

```

    @Mock
    private ApplicationExamMarkViewToRespConverter
applicationExamMarkViewToRespConverter;

    @Mock
    private ApplicationArchiveExamMarkViewToRespConverter
applicationArchiveExamMarkViewToRespConverter;

    @Mock
    private StatusNotificationService statusNotificationService;

    @Mock
    private ApplicationCustomRepository applicationCustomRepository;

    @Mock
    private ClassifierService classifierService;

    private ExamGroupService examGroupService;

    @BeforeEach
    void setUp() {
        examGroupService = new ExamGroupService(
            examGroupRegistryViewRepo,
            examGroupRegistryViewToSearchRespConverter,
            examGroupRepo,
            examGroupToRespConverter,
            examGroupReqToEntityConverter,
            applicationToExamGroupRepo,
            applicationExamMarkViewRepo,
            applicationExamMarkViewToRespConverter,
            applicationArchiveExamMarkViewToRespConverter,
            statusNotificationService,
            applicationCustomRepository,
            classifierService
        );
    }

```

```

@Nested
@DisplayName("getById - получение экзаменационной группы по ID")
class GetByIdTest {

    @Test
    @DisplayName("Успешное получение группы через API")
    void shouldGetExamGroupSuccessfully() {

        ExamGroup examGroup = mock(ExamGroup.class);
        ExamGroupResp expectedResponse = mock(ExamGroupResp.class);

        when(examGroupRepo.findById(EXAM_GROUP_ID)).thenReturn(Optional.of(examGroup));

        when(examGroupToRespConverter.convert(examGroup)).thenReturn(expectedResponse);

        ExamGroupResp actual =
examGroupService.getExamGroup(EXAM_GROUP_ID);

        assertNotNull(actual);
        assertEquals(expectedResponse, actual);
        verify(examGroupToRespConverter).convert(examGroup);
    }

    @Test
    @DisplayName("Бросает исключение, когда группа не найдена")
    void shouldThrowExceptionWhenNotFound() {

        when(examGroupRepo.findById(EXAM_GROUP_ID)).thenReturn(Optional.empty());

        assertThrows(ExamGroupNotFoundException.class,
            () -> examGroupService.getExamGroup(EXAM_GROUP_ID));
    }

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						131
Изм.	Лист	№ докум.	Подпись	Дата		

```

    }

}

@Nested
@DisplayName("deleteExamGroup - удаление экзаменационной группы")
class DeleteExamGroupTest {

    @Test
    @DisplayName("Успешное удаление группы")
    void shouldDeleteExamGroupSuccessfully() {

        ExamGroup examGroup = mock(ExamGroup.class);

        when(examGroup.getStatus()).thenReturn(ExamGroupStatus.INITIAL);

        when(examGroupRepo.findById(EXAM_GROUP_ID)).thenReturn(Optional.of(examGroup));

        when(applicationToExamGroupRepo.countAllByExamGroup(examGroup)).thenReturn(0L);

        doNothing().when(examGroupRepo).delete(any(ExamGroup.class));

        assertDoesNotThrow(() ->
examGroupService.deleteExamGroup(EXAM_GROUP_ID));
        verify(examGroupRepo, times(1)).delete(any(ExamGroup.class));
    }

    @Test
    @DisplayName("Бросает исключение, если группа не в статусе INITIAL")
    void shouldThrowExceptionWhenGroupNotInInitialStatus() {

        ExamGroup examGroup = mock(ExamGroup.class);

        when(examGroup.getStatus()).thenReturn(ExamGroupStatus.COMPLETE);

```

```

when(examGroupRepo.findById(EXAM_GROUP_ID)).thenReturn(Optional.of(examGroup));

        assertThrows(ApplicationException.class,
            () -> examGroupService.deleteExamGroup(EXAM_GROUP_ID));
        verify(examGroupRepo, never()).delete(any(ExamGroup.class));
    }

    @Test
    @DisplayName("Бросает исключение, если в группе есть студенты")
    void shouldThrowExceptionWhenGroupHasStudents() {

        ExamGroup examGroup = mock(ExamGroup.class);

        when(examGroup.getStatus()).thenReturn(ExamGroupStatus.INITIAL);

        when(examGroupRepo.findById(EXAM_GROUP_ID)).thenReturn(Optional.of(examGroup));

        when(applicationToExamGroupRepo.countAllByExamGroup(examGroup)).thenReturn(
            5L);

        assertThrows(ApplicationException.class,
            () -> examGroupService.deleteExamGroup(EXAM_GROUP_ID));
        verify(examGroupRepo, never()).delete(any(ExamGroup.class));
    }

    @Nested
    @DisplayName("canCloseExamGroup - проверка возможности закрытия группы")
    class CanCloseExamGroupTest {

        @Test
        @DisplayName("Возвращает true, если группу можно закрыть")
        void shouldReturnTrueWhenCanClose() {

```

```

        ExamGroup examGroup = mock(ExamGroup.class);

        when(examGroup.getStatus()).thenReturn(ExamGroupStatus.INITIAL);

        when(examGroup.getExamDate()).thenReturn(LocalDateTime.now().minusDays(1));

        when(examGroupRepo.findById(EXAM_GROUP_ID)).thenReturn(Optional.of(examGroup));

        when(applicationToExamGroupRepo.countAllByExamGroupAndMarkIsNull(examGroup))
            .thenReturn(0L);

        CanDoResp result =
            examGroupService.canCloseExamGroup(EXAM_GROUP_ID);

        assertNotNull(result);
        assertTrue(result.isAllowed());
    }

    @Test
    @DisplayName("Возвращает false, если дата экзамена еще не наступила")
    void shouldReturnFalseWhenExamDateInFuture() {
        ExamGroup examGroup = mock(ExamGroup.class);

        when(examGroup.getStatus()).thenReturn(ExamGroupStatus.INITIAL);

        when(examGroup.getExamDate()).thenReturn(LocalDateTime.now().plusDays(1));

        when(examGroupRepo.findById(EXAM_GROUP_ID)).thenReturn(Optional.of(examGroup));

        CanDoResp result =
            examGroupService.canCloseExamGroup(EXAM_GROUP_ID);
    }

```

```

        assertNotNull(result);
        assertFalse(result.isAllowed());
    }
}

@Nested
@DisplayName("setExamGroupApplicationMark - выставление оценки")
class SetExamGroupApplicationMarkTest {

    @Test
    @DisplayName("Успешное выставление оценки")
    void shouldSetMarkSuccessfully() {

        ApplicationExamMarkReq request = new ApplicationExamMarkReq();
        request.setApplicationToExamGroupId(10L);
        request.setMark(1);

        ApplicationToExamGroup applicationToExamGroup =
mock(ApplicationToExamGroup.class);

when(applicationToExamGroupRepo.findById(10L)).thenReturn(Optional.of(applicationToExamGroup));

when(applicationToExamGroupRepo.save(applicationToExamGroup)).thenReturn(applicationToExamGroup);

        assertDoesNotThrow(() ->
examGroupService.setExamGroupApplicationMark(request));

        verify(applicationToExamGroup).setMark(1);
        verify(applicationToExamGroup).setMarkedByCurrentUser();

        verify(applicationToExamGroup).setMarkedAt(any(LocalDateTime.class));
    }
}

```

```

verify(applicationToExamGroupRepo).save(applicationToExamGroup);
    }

    @Test
    @DisplayName("Бросает исключение, если запись не найдена")
    void shouldThrowExceptionWhenNotFound() {

        ApplicationExamMarkReq request = new ApplicationExamMarkReq();
        request.setApplicationToExamGroupId(10L);
        request.setMark(1);

        when(applicationToExamGroupRepo.findById(10L)).thenReturn(Optional.empty());
        ;

        assertThrows(ApplicationException.class,
            () ->
            examGroupService.setExamGroupApplicationMark(request));
        verify(applicationToExamGroupRepo, never()).save(any());
    }
}
}

```

Файл GradeBookServiceTest.java

```

package ru.mos.mes.profteh.backend.core.student.service;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;

```

```

import ru.mos.mes.profteh.api.contingent.client.model.PersonEducation;
import
ru.mos.mes.profteh.backend.core.profsetting.ProfSettingAssertionService;
import
ru.mos.mes.profteh.backend.core.student.converter.StudentToStudGradeBookRes
pConverter;
import ru.mos.mes.profteh.backend.core.student.model.StudGradeBookReq;
import ru.mos.mes.profteh.backend.core.student.model.StudGradeBookResp;
import ru.mos.mes.profteh.backend.error.NotFoundException;
import
ru.mos.mes.profteh.backend.sync.contingent.api.ContingentEducationApi;
import
ru.mos.mes.profteh.backend.sync.contingent.sync.EducationSyncService;
import ru.mos.mes.profteh.core.student.domain.Student;
import ru.mos.mes.profteh.core.student.domain.StudentEducation;
import ru.mos.mes.profteh.core.student.repository.StudentRepository;

import java.time.LocalDate;
import java.util.Collections;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("GradeBookService - unit-тесты")
class GradeBookServiceTest {

    private static final long STUDENT_ID = 1L;
    private static final String GRADE_BOOK_NUM = "123456";
    private static final LocalDate GRADE_BOOK_DATE = LocalDate.of(2026, 1,
15);

    @Mock
    private StudentRepository studentRepository;

    @Mock
    private StudentToStudGradeBookRespConverter

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						137
Изм.	Лист	№ докум.	Подпись	Дата		

```

studentToStudGradeBookRespConverter;

@Mock
private ProfSettingAssertionService profSettingAssertionService;

@Mock
private StudentAssertService studentAssertService;

@Mock
private EducationSyncService educationSyncService;

@Mock
private ContingentEducationApi contingentEducationApi;

@Mock
private GradeBookAssertService gradeBookAssertService;

private GradeBookService gradeBookService;

@BeforeEach
void setUp() {
    gradeBookService = new GradeBookService(
        studentRepository,
        studentToStudGradeBookRespConverter,
        profSettingAssertionService,
        studentAssertService,
        educationSyncService,
        contingentEducationApi,
        gradeBookAssertService
    );
}

@Nested
@DisplayName("getGradeBook - получение зачетной книжки")
class GetGradeBookTest {

    @Test
    @DisplayName("Успешное получение зачетной книжки")

```

```

void shouldGetGradeBookSuccessfully() {

    Student student = mock(Student.class);
    StudGradeBookResp expectedResponse =
mock(StudGradeBookResp.class);

    when(studentRepository.getStudentById(STUDENT_ID)).thenReturn(student);

    when(studentToStudGradeBookRespConverter.convert(student)).thenReturn(expectedResponse);

    doNothing().when(profSettingAssertionService).assertAllowFunctionalGradeBook();

    StudGradeBookResp actual =
gradeBookService.getGradeBook(STUDENT_ID);

    assertNotNull(actual);
    assertEquals(expectedResponse, actual);

    verify(profSettingAssertionService).assertAllowFunctionalGradeBook();
    verify(studentRepository).getStudentById(STUDENT_ID);
    verify(studentToStudGradeBookRespConverter).convert(student);
}

}

@Nested
@DisplayName("addGradeBook - добавление зачетной книжки")
class AddGradeBookTest {

    @Test
    @DisplayName("Успешное добавление зачетной книжки")
    void shouldAddGradeBookSuccessfully() {

        Student student = mock(Student.class);

```

```

        StudentEducation education = mock(StudentEducation.class);
        when(student.getEducation()).thenReturn(education);

        StudGradeBookReq request = new StudGradeBookReq(GRADE_BOOK_NUM,
        GRADE_BOOK_DATE);

        StudGradeBookResp expectedResponse =
        mock(StudGradeBookResp.class);

        when(studentRepository.getStudentById(STUDENT_ID)).thenReturn(student);

        when(studentToStudGradeBookRespConverter.convert(student)).thenReturn(expec
        tedResponse);

        doNothing().when(profSettingAssertionService).assertAllowFunctionalGradeBoo
        k();

        doNothing().when(gradeBookAssertService).assertNoGradeBook(student);
        when(studentRepository.save(student)).thenReturn(student);

        doNothing().when(studentAssertService).assertHasMeshId(student);

        PersonEducation personEducation = mock(PersonEducation.class);
        when(educationSyncService.getCurrentPersonEducationSync(any(),
        eq(student))).thenReturn(personEducation);

        when(contingentEducationApi.getPersonEducations(student)).thenReturn(Collec
        tions.emptyList());

        doNothing().when(contingentEducationApi).updatePersonEducation(any(),
        any(), any(), any());

        StudGradeBookResp actual =
        gradeBookService.addGradeBook(STUDENT_ID, request);

        assertNotNull(actual);
        assertEquals(expectedResponse, actual);

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						140
Изм.	Лист	№ докум.	Подпись	Дата		

```

        verify(gradeBookAssertService).assertNoGradeBook(student);
        verify(studentRepository, atLeastOnce()).save(student);
    }
}

@Nested
@DisplayName("updateGradeBook - обновление зачетной книжки")
class UpdateGradeBookTest {

    @Test
    @DisplayName("Успешное обновление зачетной книжки")
    void shouldUpdateGradeBookSuccessfully() {

        Student student = mock(Student.class);
        StudentEducation education = mock(StudentEducation.class);
        when(student.getEducation()).thenReturn(education);

        StudGradeBookReq request = new StudGradeBookReq(GRADE_BOOK_NUM,
        GRADE_BOOK_DATE);
        StudGradeBookResp expectedResponse =
        mock(StudGradeBookResp.class);

        when(studentRepository.getStudentById(STUDENT_ID)).thenReturn(student);

        when(studentToStudGradeBookRespConverter.convert(student)).thenReturn(expec
        tedResponse);

        doNothing().when(profSettingAssertionService).assertAllowFunctionalGradeBoo
        k();

        doNothing().when(gradeBookAssertService).assertHasGradeBook(student);
        when(studentRepository.save(student)).thenReturn(student);

        doNothing().when(studentAssertService).assertHasMeshId(student);

        PersonEducation personEducation = mock(PersonEducation.class);
        when(educationSyncService.getCurrentPersonEducationSync(any(),

```

```

eq(student))).thenReturn(personEducation);

when(contingentEducationApi.getPersonEducations(student)).thenReturn(Collections.emptyList());

doNothing().when(contingentEducationApi).updatePersonEducation(any(), any(), any(), any());

    StudGradeBookResp actual =
gradeBookService.updateGradeBook(STUDENT_ID, request);

    assertNotNull(actual);
    assertEquals(expectedResponse, actual);
    verify(gradeBookAssertService).assertHasGradeBook(student);
}
}

@Nested
@DisplayName("deleteGradeBook - удаление зачетной книжки")
class DeleteGradeBookTest {

    @Test
    @DisplayName("Успешное удаление зачетной книжки")
    void shouldDeleteGradeBookSuccessfully() {

        Student student = mock(Student.class);
        StudentEducation education = mock(StudentEducation.class);
        when(student.getEducation()).thenReturn(education);

        when(studentRepository.getById(STUDENT_ID)).thenReturn(student);

        doNothing().when(profSettingAssertService).assertAllowFunctionalGradeBook();

        when(studentRepository.save(student)).thenReturn(student);

        doNothing().when(studentAssertService).assertHasMeshId(student);

```

```

        PersonEducation personEducation = mock(PersonEducation.class);
        when(educationSyncService.getCurrentPersonEducationSync(any(),
eq(student))).thenReturn(personEducation);

when(contingentEducationApi.getPersonEducations(student)).thenReturn(Collec
tions.emptyList());

doNothing().when(contingentEducationApi).updatePersonEducation(any(),
any(), any(), any());

        assertDoesNotThrow(() ->
gradeBookService.deleteGradeBook(STUDENT_ID));

        verify(studentRepository).save(student);
    }
}

@Nested
@DisplayName("Граничные случаи и ошибки")
class BoundaryAndErrorTest {

    @Test
    @DisplayName("При отсутствии студента выбрасывается
NotFoundException")
    void shouldThrowNotFoundExceptionWhenStudentNotFound() {

        StudGradeBookReq request = new StudGradeBookReq(GRADE_BOOK_NUM,
GRADE_BOOK_DATE);

when(studentRepository.getStudentById(STUDENT_ID)).thenThrow(new
NotFoundException("Студент не найден"));

        assertThrows(NotFoundException.class,
            () -> gradeBookService.addGradeBook(STUDENT_ID,
request));
    }
}

```

```

        @Test
        @DisplayName("При отсутствии зачетной книжки при обновлении
выбрасывается исключение")
        void shouldThrowExceptionWhenNoGradeBookForUpdate() {

            Student student = mock(Student.class);
            StudGradeBookReq request = new StudGradeBookReq(GRADE_BOOK_NUM,
GRADE_BOOK_DATE);

            when(studentRepository.getStudentById(STUDENT_ID)).thenReturn(student);

            doNothing().when(profSettingAssertionService).assertAllowFunctionalGradeBoo
k();

            doThrow(new RuntimeException("Зачетная книжка не найдена"))

            .when(gradeBookAssertService).assertHasGradeBook(student);

            assertThrows(RuntimeException.class,
                () -> gradeBookService.updateGradeBook(STUDENT_ID,
request));
        }
    }
}

```

Файл ProfSettingAssertionServiceTest.java

```

package ru.mos.mes.profteh.backend.core.student.service;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.mockito.Mock;
import org.mockito.junit.jupiter.MockitoExtension;
import
ru.mos.mes.profteh.backend.core.profsetting.ProfSettingAssertionService;
import

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						144
Изм.	Лист	№ докум.	Подпись	Дата		

```

ru.mos.mes.profteh.backend.core.profsetting.ProfSettingCustomService;
import ru.mos.mes.profteh.core.exception.ApplicationException;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.Mockito.*;

@ExtendWith(MockitoExtension.class)
@DisplayName("ProfSettingAssertionService - unit-тесты")
class ProfSettingAssertionServiceTest {

    @Mock
    private ProfSettingCustomService profSettingService;

    private ProfSettingAssertionService assertionService;

    @BeforeEach
    void setUp() {
        assertionService = new
ProfSettingAssertionService(profSettingService);
    }

    @Nested
    @DisplayName("assertAgencyEnabled - проверка межведомственных
проверок")
    class AssertAgencyEnabledTest {

        @Test
        @DisplayName("Не бросает исключение, когда функционал включен")
        void shouldNotThrowWhenAgencyEnabled() {

when(profSettingService.isSettingFalse(ProfSettingCustomService.AGENCY_ENAB
LED)).thenReturn(false);

        assertDoesNotThrow(() ->
assertionService.assertAgencyEnabled());

verify(profSettingService).isSettingFalse(ProfSettingCustomService.AGENCY_E

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						145
Изм.	Лист	№ докум.	Подпись	Дата		

```

NABLED) ;

    }

    @Test
    @DisplayName("Бросает исключение, когда функционал выключен")
    void shouldThrowWhenAgencyDisabled() {

when(profSettingService.isSettingFalse(ProfSettingCustomService.AGENCY_ENAB
LED)).thenReturn(true);

        ApplicationException exception =
assertThrows(ApplicationException.class,
                () -> assertionService.assertAgencyEnabled());

        assertEquals("Функционал запуска межведомственных проверок
выключен", exception.getMessage());

verify(profSettingService).isSettingFalse(ProfSettingCustomService.AGENCY_E
NABLED);
    }
}

    @Nested
    @DisplayName("assertAllowFunctionalGradeBook - проверка зачетной
книжки")
    class AssertAllowFunctionalGradeBookTest {

        @Test
        @DisplayName("Не бросает исключение, когда зачетная книжка
включена")
        void shouldNotThrowWhenGradeBookEnabled() {

when(profSettingService.isSettingFalse(ProfSettingCustomService.ALLOW_FUNCT
IONAL_GRADE_BOOK)).thenReturn(false);

                assertDoesNotThrow(() ->

```

```

assertionService.assertAllowFunctionalGradeBook());

verify(profSettingService).isSettingFalse(ProfSettingCustomService.ALLOW_FUNC
NCTIONAL_GRADE_BOOK);
    }

    @Test
    @DisplayName("Бросает исключение, когда зачетная книжка выключена")
    void shouldThrowWhenGradeBookDisabled() {

when(profSettingService.isSettingFalse(ProfSettingCustomService.ALLOW_FUNC
TIONAL_GRADE_BOOK)).thenReturn(true);

        ApplicationException exception =
assertThrows(ApplicationException.class,
                () ->
assertionService.assertAllowFunctionalGradeBook());

        assertEquals("Функционал зачетной книжки выключен",
exception.getMessage());

verify(profSettingService).isSettingFalse(ProfSettingCustomService.ALLOW_FUNC
NCTIONAL_GRADE_BOOK);
    }
}

```

					ПГУ 1.090301.05.001 ПЗ	Лист
						147
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(обязательное)

Бизнес-проект «Система модульного тестирования интернет-сервиса для
зачисления в образовательные организации

Презентация

					ПГУ 1.090301.05.001 ПЗ	Лист
						148
Изм.	Лист	№ докум.	Подпись	Дата		

Бизнес-проект «Система модульного тестирования интернет-сервиса для зачисления в образовательные организации»

Выпускная квалификационная работа

Автор: Воробьева М. М., студентка гр. 22ВВГП1

Руководитель: к.э.н., доцент Акифьев И. В.

Пенза,
2026

1

АКТУАЛЬНОСТЬ ТЕМЫ

ВЫСОКИЕ ТРЕБОВАНИЯ К КАЧЕСТВУ

Система должна работать безотказно при пиковых нагрузках и частых обновлениях функционала.

РИСКИ

Любая ошибка в коде может сорвать критически важную приёмную кампанию.

ЦИФРОВАЯ ТРАНСФОРМАЦИЯ ОБРАЗОВАНИЯ

Онлайн-зачисление становится стандартом, сервисы постоянно развиваются и усложняются.

РЕШЕНИЕ: МОДУЛЬНОЕ ТЕСТИРОВАНИЕ

2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

Лист

149

ПРЕИМУЩЕСТВА МОДУЛЬНОГО ТЕСТИРОВАНИЯ

ЭКОНОМИЯ ВРЕМЕНИ

Значительное сокращение времени на поиск и локализацию ошибок в коде.

БЕЗОПАСНЫЙ РЕФАКТОРИНГ

Позволяет изменять внутреннюю структуру кода без риска сломать логику.

ЖИВАЯ ДОКУМЕНТАЦИЯ

Тесты наглядно описывают, как именно должен работать каждый модуль системы.

ЭКОНОМИЧЕСКАЯ ЭФФЕКТИВНОСТЬ

Исправление ошибки на этапе модульного тестирования обходится в десятки раз дешевле, чем после релиза.

3

ЦЕЛЬ И ЗАДАЧИ РАБОТЫ

Цель: разработать систему модульного тестирования для интернет-сервиса зачисления.

✔ Анализ существующих аналогов и решений

✔ Разработка приложения для запуска

✔ Выбор стека технологий

✔ Тестирование и отладка

✔ Реализация тестов для 5 ключевых сервисов

✔ Описание бизнес-модели

4

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

Лист

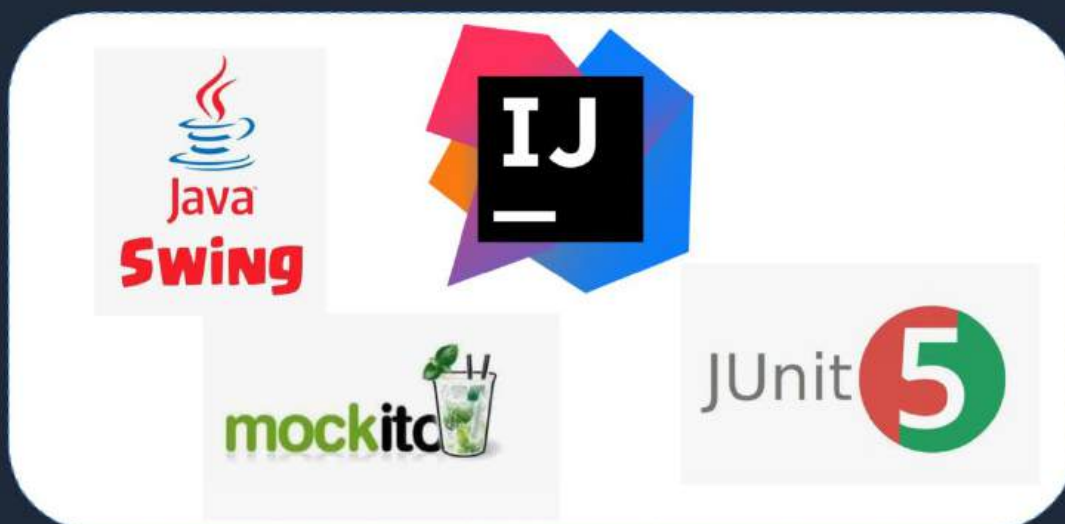
150

Аналоги системы модульного тестирования

Критерий	Maven Surefire Plugin	GitLab CI	TeamCity	Разработанная система
Графический интерфейс	-	+	+	+
Выборочный запуск тестов	-	-	-	+
История запусков	-	+	+	+
Сложность настройки	Высокая	Средняя	Высокая	Низкая

5

Используемый стек технологий



6

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

Лист

151

АРХИТЕКТУРА СИСТЕМЫ

ПРИЛОЖЕНИЕ ДЛЯ ЗАПУСКА

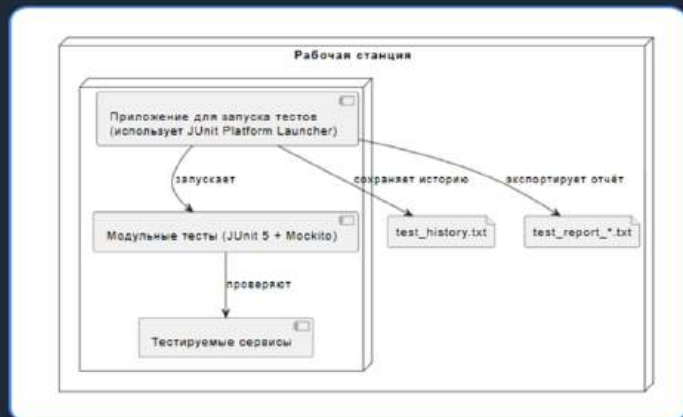
Интерфейс на Java Swing для управления и визуализации.

МОДУЛЬНЫЕ ТЕСТЫ

Изолированная проверка логики с использованием JUnit 5 и Mockito.

ТЕСТИРУЕМЫЕ СЕРВИСЫ

Целевой интернет-сервис зачисления.



7

РЕАЛИЗАЦИЯ МОДУЛЬНЫХ ТЕСТОВ



Каждый тестовый класс содержит реальный экземпляр тестируемого сервиса (создаётся в setUp()) и моки его зависимостей (аннотация @Mock).

Тесты сгруппированы с помощью @Nested по проверяемым методам или сценариям.

Полная изоляция: зависимости сервисов заменены моками, тесты не зависят от реальной БД и внешних зависимостей.

Тесты быстрые, воспроизводимые, не зависят от внешней среды.

8

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

Лист

152

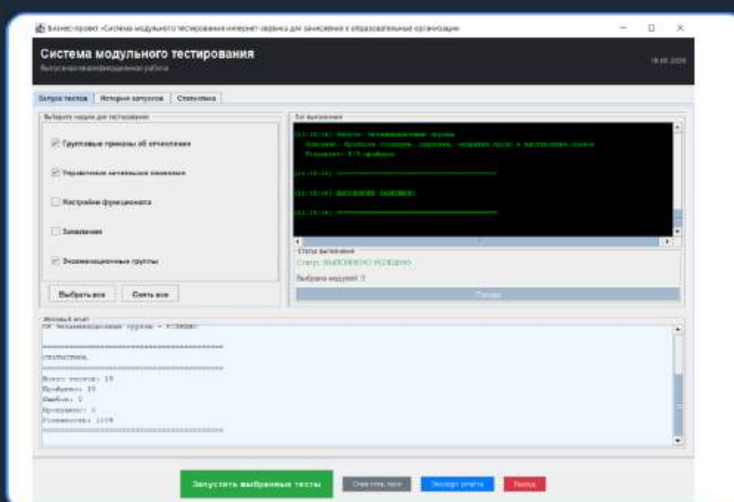
РЕАЛИЗАЦИЯ МОДУЛЬНЫХ ТЕСТОВ

Всего реализовано:

Тестовый класс	Проверяемый сервис	Кол-во тестов
ApplicationCreationCheckServiceComplexTest	Заявления на поступление	7
ExamGroupServiceTest	Экзаменационные группы	9
GradeBookServiceTest	Зачётные книжки	6
ProfSettingAssertionServiceTest	Настройки функционала	4
ExpulsionGroupOrdersServiceTest	Групповые приказы	4

9

ПРИЛОЖЕНИЕ ДЛЯ ЗАПУСКА ТЕСТОВ



Особенности

- ✓ Простота – выбор модулей, запуск одной кнопкой
- ✓ Наглядность – лог + итоговый отчет в одном окне
- ✓ История и статистика – таблица и показатели
- ✓ Экспорт – сохранение отчёта в файл

10

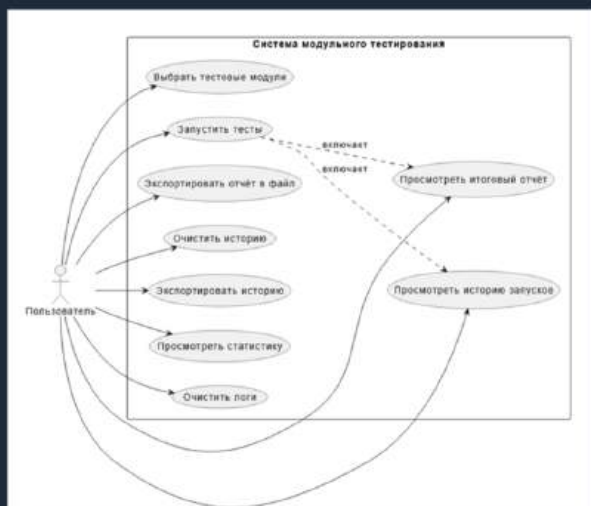
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

Лист

153

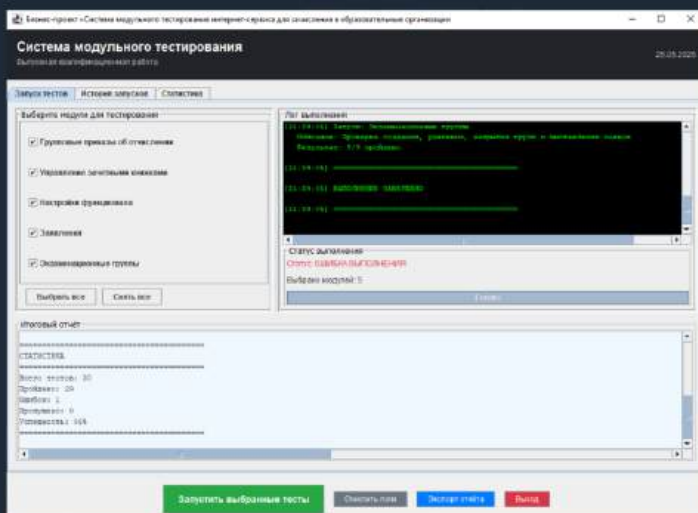
Варианты использования системы модульного тестирования



1. Выбор модулей для запуска
2. Логирование выполнения
3. Индикатор прогресса
4. Экспорт отчётов
5. Просмотр истории и статистики

1
1

ТЕСТИРОВАНИЕ



План тестирования состоял из 5 основных частей:

- ✓ проверка модульных тестов;
- ✓ проверка приложения-раннера;
- ✓ комплексная проверка;
- ✓ анализ результатов прохождения тест-кейсов и исправление возможных ошибок;
- ✓ повторное тестирование всех компонентов программы.

Система работает стабильно

1
2

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.05.001 ПЗ

Лист

154

БИЗНЕС-МОДЕЛЬ ПРОЕКТА

- **Ценностное предложение:** Разработчик экономит время на написание тестов, получая готовые тесты для образовательного сервиса и приложение с выборочным запуском, историей и трендами успешности.
- **Потоки доходов:** Прямая продажа лицензии конечному пользователю (IT-компания), техническая поддержка, выпуск обновлений
- **Ключевые ресурсы:** исходный код, 30 готовых модульных тестов, исходный код графического приложения.
- **Ключевая деятельность:** Разработка и поддержка кода, исправление ошибок и выпуск обновлений, добавление новых модулей тестов по требованию, техническая поддержка клиентов.
- **Структура издержек:** Разработка и поддержка кода (60%), маркетинг и продвижение (20%), техническая поддержка (20%).

1
3

ИТОГИ РАБОТЫ



Разработана архитектура системы и реализовано 30 модульных тестов.



Создано приложение для запуска тестов с функциями отчётности и хранения истории.



Проведено комплексное тестирование, выявленные ошибки исправлены.



Построена бизнес-модель, подтверждающая готовность к внедрению.

1
4

					ПГУ 1.090301.05.001 ПЗ	Лист
						155
Изм.	Лист	№ докум.	Подпись	Дата		

СПАСИБО ЗА ВНИМАНИЕ!

1
5

					ПГУ 1.090301.05.001 ПЗ	Лист
						156
Изм.	Лист	№ докум.	Подпись	Дата		