МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

09.03.03 Прикладная информатика
(код и наименование направления подготовки, специальности)

Разработка программного обеспечения
(направленность (профиль)/специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Разработка Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах

Обучающийся И.Б. Волынчиков _____
(Инициалы Фамилия) (личная подпись)

Руководитель Н.Н. Казаченок
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2024

Аннотация

В связи с происходящими мировыми событиями, особенно важным становится вопрос обеспечения технологического суверенитета. В то же время, множество программных продуктов, особенно узкоспециализированных, не имеет широко распространенных российских аналогов.

Одним из таких продуктов являются сервисы для создания двойных тестовых объектов в автоматизированных модульных тестах. В то же время, отказаться от использования таких сервисов в индустрии разработки и тестирования программного обеспечения, предусматривающего широкое использование API-взаимодействий, практически невозможно.

Настоящая работа посвящена разработке Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах. Функционала языка программирования Java, а также таких фреймворков как Spring и Thymeleaf, вполне достаточно для создания работоспособного приложения, пригодного для использования в командах разработки и тестирования для имитации ответов API сторонних сервисов, имеющего в том числе удобный веб-интерфейс.

Оглавление

Введение

В современной практике программирования особое внимание уделяется автоматизации тестирования, которая играет ключевую роль в обеспечении качества программных продуктов. Одним из фундаментальных компонентов эффективной стратегии тестирования является применение тестовых двойников (test doubles). Эти инструменты позволяют осуществить изоляцию тестируемого компонента от его зависимостей, что существенно повышает точность и надежность тестирования.

Актуальность исследования подтверждается тем, что ведущие решения в области создания тестовых двойников, такие как WireMock и MockServer, представляют собой продукты иностранного происхождения, что может привести к проблемам с зависимостью от зарубежных разработок и потенциальными сложностями в их интеграции и поддержке.

Объектом исследования является процесс разработки и тестирования программного обеспечения в контексте интеграции и использования веб-сервисов.

Предметом исследования выступают методы и технологии автоматизации создания макетов ответов API для ускорения и оптимизации процессов разработки и тестирования программных решений.

Целью выпускной квалификационной работы является разработка и внедрение системы управления макетами ответов API (MockManager), направленной на повышение эффективности и качества процессов разработки и тестирования программного обеспечения путем автоматизации подготовки и управления тестовыми данными.

В рамках данной работы предлагается разработка RESTful сервиса на базе платформы Spring Boot, который будет функционировать как управляющий центр для динамического создания и управления тестовыми двойниками, используемыми в автоматизированных модульных тестах. Проект направлен не только на разработку и внедрение данного сервиса, но и

на демонстрацию его практической применимости через серию конкретных примеров. Это даст возможность не только глубже понять механизмы работы тестовых двойников, но и оценить их вклад в повышение эффективности разработки программных решений.

Задачи, решаемые в работе:

- изучение существующих подходов и инструментов для автоматизации создания макетов ответов API;
- проектирование архитектуры и выбор технологического стека для разработки системы MockManager;
- реализация веб-интерфейса и функционала управления макетами ответов API;
- тестирование системы на реальных данных и анализ полученных результатов.

Среди отечественных разработок в данной области наблюдается значительное разнообразие подходов и инструментов, которые часто разрабатываются в рамках отдельных предприятий «с нуля». Такой подход препятствует стандартизации и обмену опытом между организациями, что в свою очередь затрудняет масштабирование решений и снижает эффективность их использования. Возникающая потребность в унификации подходов к созданию и поддержке тестовых двойников ставит перед научным и профессиональным сообществами задачу разработки стандартизированных и универсально применимых инструментов.

Таким образом, разработка унифицированного RESTful сервиса для управления тестовыми двойниками на основе Spring Boot представляет собой ответ на актуальную потребность рынка в более гибких, доступных и эффективных инструментах для автоматизации тестирования. Это не только улучшит процессы разработки во многих компаниях, но и позволит более свободно обмениваться опытом и технологиями между различными игроками на рынке.

Ожидаемые результаты:

- уменьшение времени на подготовку и проведение тестирования ПО за счет использования автоматизированного инструмента для создания и управления макетами ответов API;
- повышение качества программного продукта благодаря более тщательному и системному подходу к тестированию;
- экономическая выгода от сокращения затрат на трудозатраты программистов и тестировщиков.

Настоящая выпускная квалификационная бакалаврская работа состоит из введения, трех глав, заключения, списка используемой литературы и используемых источников.

Первая глава посвящена анализу предметной области.

Вторая глава освещает вопросы архитектуры и особенностей проекта.

Третья глава описывает процесс тестирования Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах – юнит-тесты, модульные и интеграционные тесты, применяемые при разработке.

В заключении подводятся итоги выполненного программного проекта и описываются результаты выполнения выпускной квалификационной работы.

Бакалаврская работа состоит из 51 страницы текста, 14 рисунков, 1 таблицы и 31 источника.

Глава 1 Постановка задачи на разработку программного обеспечения Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах

1.1 Подходы и инструменты для автоматизации создания макетов ответов API

Для всестороннего понимания процесса тестирования критически важно освоить концепцию использования тестовых двойников. Эти элементы автоматизированных тестов имитируют поведение реальных компонентов системы, что позволяет проводить тестирование отдельных частей программы в условиях строгой изоляции. Использование тестовых двойников не только ускоряет процесс тестирования, но и значительно повышает его точность, особенно при работе с сложными системами.

Тестовые двойники классифицируются на несколько основных типов: моки (mocks), стабы (stubs), фейки (fakes) и шпионы (spies). [8] Каждый из этих типов имеет свою специфическую функцию:

- моки применяются для эмуляции поведения объектов, позволяя проверять взаимодействия внутри системы через подсчет вызовов и проверку передаваемых данных;
- стабы используются для предоставления предопределенных ответов на вызовы методов, что особенно полезно в условиях стабильных, но сложно конфигурируемых зависимостей;
- фейки представляют собой рабочие реализации, которые, однако, выполняют упрощенные функции по сравнению с настоящими компонентами, часто используются для тестирования функциональности;
- шпионы отслеживают использование объектов, записывая вызовы методов, что позволяет анализировать поведение системы после выполнения тестов.

Эти инструменты оказываются незаменимыми в ситуациях, когда прямое взаимодействие с подсистемами или внешними сервисами невозможно или нежелательно по причинам, таким как отсутствие настройки интеграции, некорректные или устаревшие данные от внешних сервисов, или когда требуется изолировать тестирование от возможных побочных эффектов.

Применение тестовых двойников значительно повышает стабильность и надежность автоматизированных тестов. Например, стаб может быть использован для имитации ответов от веб-сервиса Федеральной налоговой службы в банковской программе, создавая условия, при которых программа будет воспринимать наличие налоговой задолженности у клиента. Такой подход позволяет проводить тестирование функций оплаты налогов без реального взаимодействия с ФНС, что обеспечивает безопасность и контроль над тестовым окружением.

В итоге, использование тестовых двойников является эффективным средством для улучшения качества программного обеспечения, оптимизации процессов разработки и достижения более высокой производительности приложений. Такой подход дает возможность разработчикам и тестировщикам сосредоточиться на точном и целенаправленном тестировании функционала, минимизируя риски и ошибки. [11]

Для более ясного изложения, в дальнейшем в данной работе все типы тестовых двойников будут именоваться заглушками, или моками, а процесс использования заглушек – для краткости, мокированием.

На рисунке 1 представим процесс тестирования нового функционала локального приложения без использования моков.

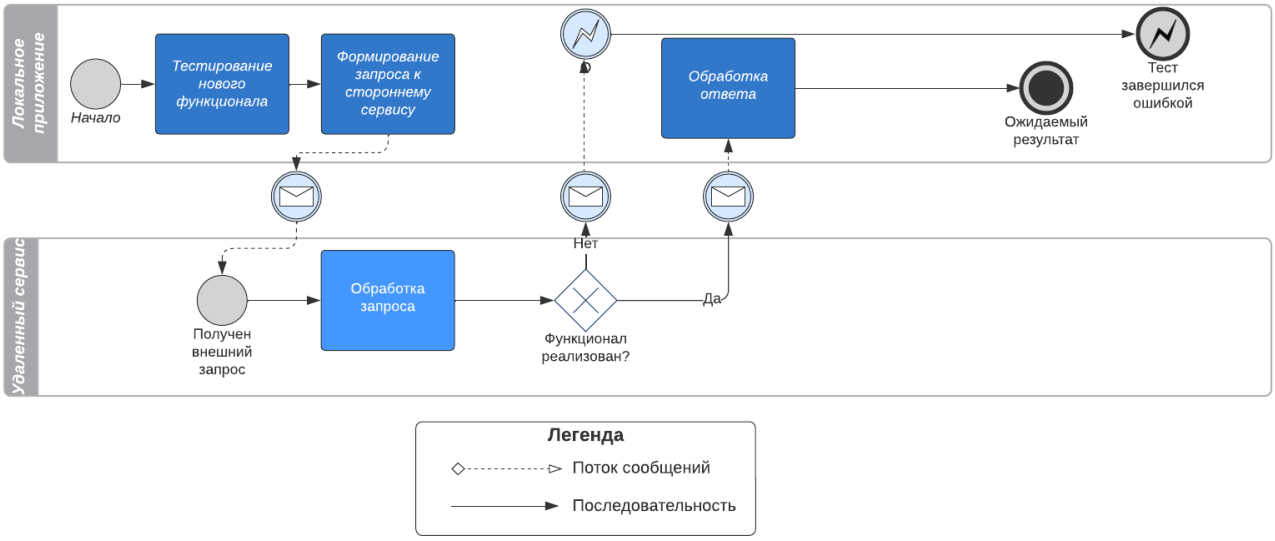


Рисунок 1 – BPMN-диаграмма процесса тестирования нового функционала приложения

Как видно из представленного рисунка, возможна ситуация, когда новый функционал реализован в локальном приложении, но по каким-то причинам недоступен на удаленном сервисе.

В таком случае, при запросе на предоставление такого функционала удаленный сервис будет возвращать ошибку, соответственно, бэкенд-часть локального приложения, получив сообщение с ошибкой, не сможет получить ожидаемый результат, а значит, тест будет провален.

Теперь рассмотрим, как изменится ситуация, если применить заглушку (мок). В таком случае, если тестировщик знает, что не получит ожидаемого ответа от удаленного сервиса, он может временно изменить настройки тестируемого приложения и совершить запрос не в реальный удаленный сервис, а в имитирующую его заглушку. Графически, такая ситуация представлена на рисунке 2.

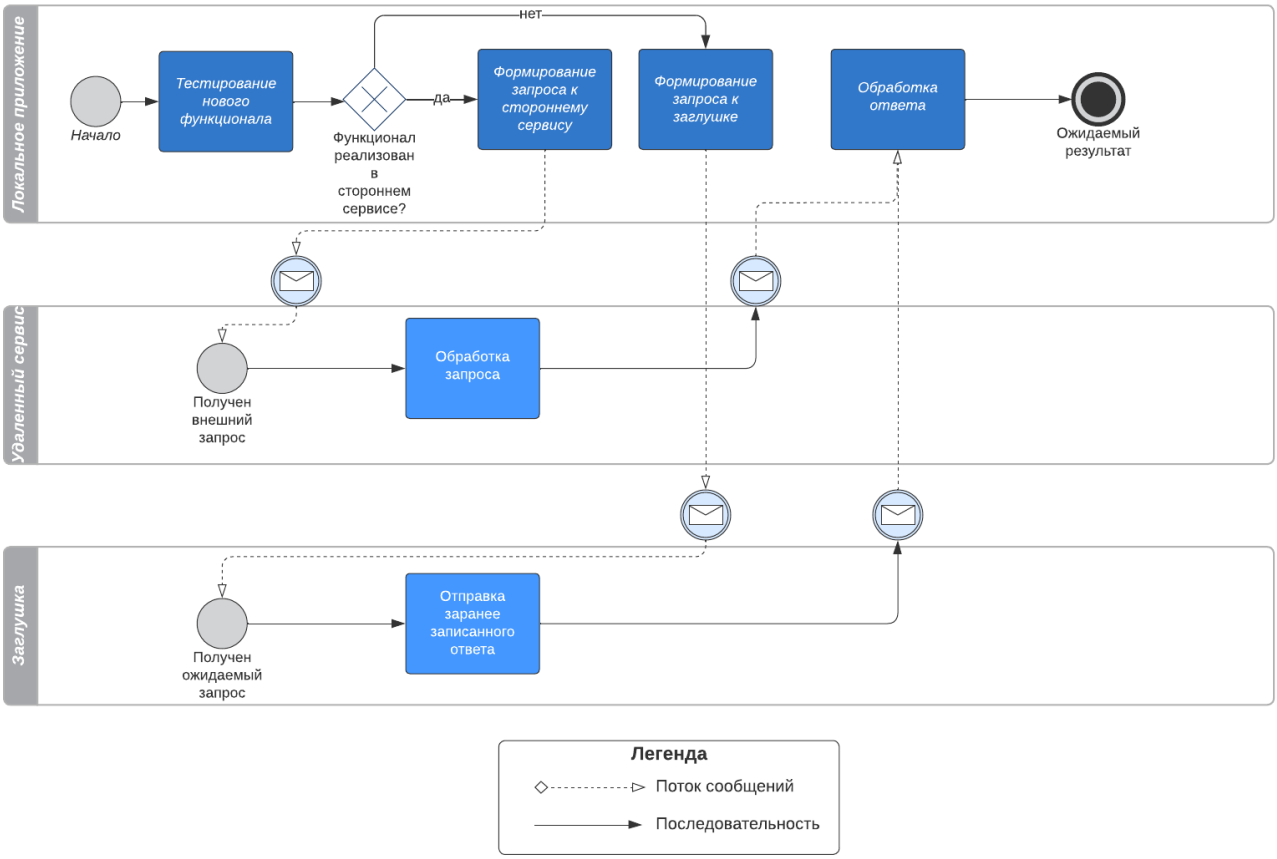


Рисунок 2 – BPMN-диаграмма процесса тестирования нового функционала приложения с использованием заглушки

При анализе представленной диаграммы становится очевидным, что необходимости замены всех ответов сервисов заглушками нет. Более того, такой подход часто может приводить к излишней зависимости от заглушек в тест-дизайне, что, в свою очередь, замедляет процесс разработки и может маскировать потенциальные проблемы в поведении системы. Осознание того, что заглушки являются лишь одним из инструментов, притом не всегда обязательным, открывает двери для оптимизации тестирования, позволяя сфокусироваться на более критичных участках приложения и подвергать тщательной проверке именно их работоспособность.

Тем не менее, стоит подчеркнуть, что в контексте методологии TDD (Test-Driven Development, или «разработка через тестирование»), заглушки играют важную роль. В TDD сначала создаются тесты для новых функций на основе требований аналитиков, а затем разрабатывается код, который должен

удовлетворять этим тестам. [17] Заглушки здесь выступают как средство для обеспечения изоляции тестов, что позволяет поддерживать высокую скорость разработки и сразу же выявлять ошибки на ранних этапах, значительно повышая качество конечного продукта. [30]

В Java-мире наибольшую популярность в создании тестовых двойников заслужили такие инструменты, как WireMock и MockServer.

WireMock это симулятор HTTP-сервера/прокси, который позволяет стабилизировать и тестировать веб-сервисы. Он может использоваться для возвращения предопределенных ответов на запросы и для записи запросов в тестовые цели. [27]

Mock Server позволяет легко мокировать любые системы, которые интегрируются через HTTP или HTTPS (например, внешние сервисы, нестабильные сервисы или еще не разработанные сервисы).

Postman Mock Services в целом повторяет функционал MockServer, но адаптирован для совместного использования со специализированной программой для тестировщиков Postman.

Таким образом, использование WireMock и MockServer (а также менее известных программ, например, Postman Mock Services или внутренних разработок компаний) становится центральным элементом в процессе разработки веб-приложений на Java, способствуя созданию стабильных и надежных программных решений. Эти инструменты помогают не только в диагностике и исправлении ошибок, но и способствуют более глубокому пониманию взаимодействия внутри приложения, что является ключевым для достижения оптимальной производительности и устойчивости программных продуктов.

Анализ современных решений на рынке инструментов для создания тестовых двойников подтверждает, что основной акцент в этих разработках делается на повышение удобства использования и налаживание интеграции с распространёнными средами программирования. Они предлагают продвинутые функции для симуляции HTTP-сервисов, позволяя

разработчикам эффективно создавать моки для сетевых запросов, что значительно упрощает процесс тестирования взаимодействий между различными компонентами системы.

Дополнительно отметим, что широко известные инструменты, такие как WireMock и Mock Server, продолжают удерживать лидерские позиции на рынке благодаря своей гибкости и многофункциональности. Эти инструменты предоставляют разработчикам мощные возможности для написания и управления тестами, включая проверку поведения объектов и составление сложных сценариев тестирования, что делает их востребованными в профессиональном сообществе.

Однако, текущая политическая и экономическая ситуация порождает определённые риски и сложности в использовании иностранных разработок в критически важных отраслях, включая информационные технологии. Это обуславливает необходимость разработки отечественных аналогов, которые были бы не только сопоставимы по функциональности с западными аналогами, но и адаптированы к специфике российского рынка.

Наглядно сравним характеристики существующих программных продуктов в табличном виде:

Таблица 1 – Обзор и анализ аналогов программного обеспечения

Параметр	WireMock	Mock Server, Postman	Корпоративная разработка	Разрабатываемая программа (Mock Manager)
Отечественная	-	-	+	+
Широко применимая	+	+	-	+
Поддержка веб-приложений	+	+	+	+
Визуальный интерфейс	-	+	+	+

Создание российского решения для тестирования программного обеспечения должно учитывать такие факторы, как локализация интерфейсов, соответствие национальным стандартам безопасности и возможность интеграции с уже используемыми в стране технологическими стеками. Это позволит не только удовлетворить текущие потребности отечественных разработчиков и тестировщиков, но и обеспечить более высокий уровень независимости от иностранных технологий, повышая самодостаточность и устойчивость национальной IT-инфраструктуры.

Таким образом, важно стимулировать и поддерживать разработку национальных программных продуктов в области тестирования и качества программного обеспечения, что способствует не только технологическому суверенитету, но и развитию технологической экосистемы страны. В этом контексте ключевую роль могут сыграть государственные инициативы и частные инвестиции в отечественные IT-проекты, нацеленные на создание и внедрение надёжных и эффективных инструментов для тестирования.

1.2 Функциональные и нефункциональные требования в методологии FURPS+

Разработка проекта начинается с тщательного определения функциональных требований, которые служат основой для всех дальнейших этапов проектирования и реализации. Эти требования не только описывают желаемые возможности и способности системы, но и критически важны для её успешной реализации. Четкое и полное описание функций обеспечивает прозрачность проекта и уменьшает риски возникновения недоразумений в процессе его разработки.

Функциональные требования (Functional)

а) управление заглушками по принципу CRUD**:**

1) создание новой заглушки (mock) с указанием параметров запроса и ответа (CREATE);

2) просмотр списка всех доступных заглушек с пагинацией (READ);
3) редактирование существующих заглушек (UPDATE);
4) удаление заглушек (DELETE).

б) журналирование:
1) автоматическое логирование всех операций с заглушками (создание, изменение, удаление);
2) просмотр журнала операций с возможностью фильтрации по дате, типу операции и другим параметрам.

в) пользовательский интерфейс:
1) интерактивный веб-интерфейс для управления заглушками и просмотра журнала;
2) формы для создания и редактирования заглушек;
3) визуализация списка заглушек и журналов с поддержкой сортировки и пагинации.

Нефункциональные требования (Usability, Reliability, Performance, Supportability + Others)

а) юзабилити (Usability):
1) интуитивно понятный интерфейс, доступный для новых пользователей без предварительного обучения.

б) надежность (Reliability)**:**
1) обеспечение стабильной работы приложения с автоматическим восстановлением после сбоев;
2) регулярное резервное копирование данных.

в) производительность (Performance)**:**
1) быстрая обработка запросов на создание, редактирование и удаление моков;
2) оптимизация загрузки страниц и ответов сервера, чтобы время ответа не превышало 2 секунды при стандартном использовании.

г) поддерживаемость (Supportability)**:**
1) легкость в обслуживании и обновлении компонентов системы;

2) наличие документации по коду и системе.

д) другие требования (Others)**:**

1) безопасность: Реализация мер безопасности, таких как аутентификация и авторизация пользователей;
2) совместимость: Поддержка основных современных браузеров (Chrome, Firefox, Safari, Edge).

Потребности пользователей:

- разработчики: нуждаются в удобном инструменте для создания и тестирования заглушек API для ускорения процесса разработки.
- QA инженеры: используют приложение для настройки заглушек для тестирования поведения системы при различных сценариях ответов от внешних сервисов.
- администраторы системы: интересуются журналом операций для аудита и мониторинга изменений в конфигурации заглушек.

Дополнительные аспекты для учета:

- масштабируемость и производительность: важно предусмотреть возможности масштабирования системы для обработки возрастающего объема данных и увеличения числа пользователей без снижения производительности; [1]
- совместимость и интеграция: система должна быть спроектирована таким образом, чтобы обеспечивать легкую интеграцию с другими инструментами и платформами, что повысит её универсальность и привлекательность на рынке; [4]
- поддержка и усовершенствование проекта: проект должен иметь задел для дальнейшего совершенствования и иметь возможность внесения изменений в программный код при необходимости.[5]

Резюмируя вышесказанное, необходимо отметить, что реализация проекта по созданию сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах может считаться успешной не в том

случае, если выполнены формальные функциональные требования и приложение просто работает, но только тогда, когда созданный сервис также действительно удовлетворяет потребностям потенциальных пользователей, а также соответствует современным стандартам разработки программного обеспечения.

1.3 Формирование бизнес-целей и требований ИТ-проекта по созданию сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах

До того, как приступать к созданию программного продукта, сначала необходимо определить, для чего он нужен (бизнес цели), и уже в соответствии с ними двигаться дальше. Для разрабатываемого сервиса, автор выбрал следующие цели:

- сокращение времени разработки и выпуска новых продуктов;
- повышение качества программного обеспечения за счет более эффективного и обширного тестирования;
- оптимизация затрат на поддержку и исправление ошибок после выпуска продукта. [23]

Следующим этапом, необходимо сформировать календарный план разработки. [19] В данном конкретном случае, автор в целом ограничен требованиями Университета по срокам предоставления выполненного дипломного проекта. Однако, поскольку работа была начата во время окончания предыдущего семестра, а именно – в новогодние праздники, бюджет времени проекта получилось значительно расширить.

Календарный план:

- фаза разработки: 3 месяца (20.12.2023-20.03.2024);
- фаза тестирования и доработки: 1 месяц (20.03.2024-20.04.2024);
- фаза оценки и доработки по результатам предзащиты: 1 месяц

(20.04.2024-15.05.2024).

Таким образом, заблаговременное планирование позволило выделить больше времени на проект и организовать по-настоящему эффективную работу, оставив достаточно времени на продумывание применяемых технических решений.

1.4 Постановка задачи на разработку программного обеспечения

Постановка задачи на разработку сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах требует сначала четко спланировать и описать архитектурные и функциональные особенности приложения. Подробнее рассмотрим их в этом разделе.

Общие архитектурные особенности:

- модульная структура: решено разрабатывать сервис с использованием модульного подхода, где каждый модуль отвечает за выполнение определенного набора функций. Это значительно упростит процесс разработки, обеспечит некоторую безопасность создаваемого сервиса и в дальнейшем поможет проще усовершенствовать приложение;
- веб-ориентированность: поскольку сервис разрабатывается для применения пользователями, не являющимися Java-разработчиками, он должен иметь понятный и удобный графический интерфейс. Для данного приложения автор выбрал веб-интерфейс, поскольку он хорошо адаптирован под многопользовательскую эксплуатацию и сравнительно прост;
- интеграция с внешними системами: практически все современные системы поддерживают внешнюю интеграцию по API, а данный проект, являясь по сути эмулятором, сам предназначен для имитации ответов API сторонних сервисов;
- безопасность: необходимо использовать архитектурные и

технические решения для обеспечения безопасности приложения. Трезво оценивая имеющиеся навыки и доступные технологии, было выбрано использование многослойной архитектуры, а также использование логина и пароля для аутентификации.

Функциональные особенности создаваемого сервиса:

- управление тестовыми двойными объектами: сервис должен позволять создавать, редактировать, удалять и управлять макетами ответов API (заглушками). Это включает в себя определение URL-адресов (хотя сам хост остается неизменным), методов запросов, параметров (например, таймаута ответа) и тел запросов/ответов;
- хранение данных: система должна обеспечивать надежное хранение данных о макетах с возможностью быстрого поиска и извлечения информации. В прототипе приложения с этой целью был использован обычный JSON-файл, но в дальнейшем для улучшения производительности это решение было заменено на использование базы данных SQL;
- пользовательский интерфейс: необходимо разработать интерфейс, который будет интуитивно понятен конечным пользователям и позволит им эффективно взаимодействовать с функциями системы.
- логирование и мониторинг: Система должна вести журнал событий (логи), что позволит отслеживать действия пользователей и системные события;
- тестирование и устранение ошибок (дебаггинг): необходимо предусмотреть возможности для тестирования созданных макетов ответов API и отладки в случае возникновения ошибок.

Результаты, которые должно обеспечивать ПО:

- эффективность тестирования: Сокращение времени, необходимого на подготовку и выполнение тестов за счет автоматизации создания макетов ответов;
- повышение качества разработки: Улучшение качества и надежности разработанного программного обеспечения за счет более

тщательного тестирования;

- оперативное внедрение изменений: Быстрая адаптация и реализация изменений в API благодаря гибким настройкам макетов.

Эта постановка задачи устанавливает четкие рамки для разработки и определяет ключевые направления дальнейших действий по проектированию и реализации системы MockManager.

1.5 Выбор технологического стека для реализации сервиса

Для повышения эффективности разработки тестов и обеспечения качества в проекте был выбран фреймворк Spring, который дополнен рядом важных зависимостей. Этот выбор не случаен: благодаря своим выдающимся возможностям по созданию стабильных, масштабируемых и легко поддерживаемых микросервисов, Spring Boot является оптимальным решением для динамически развивающихся IT-проектов. Эта технология значительно улучшает качество тестирования и повышает надежность работы системы, в том числе за счет более эффективного управления заглушками. [10]

В рамках проекта активно использовались дополнительные возможности Spring для упрощения настройки и повышения безопасности приложения. В частности, Spring Boot облегчил начальную конфигурацию и запуск приложения, а Spring Security предоставил надежные инструменты для реализации механизмов аутентификации и авторизации пользователей, что критично для соблюдения стандартов безопасности в современных веб-приложениях.

Благодаря использованию Spring, автор смог избежать множества рутинных задач, связанных с интеграцией и управлением компонентами приложения. Всё это стало возможным благодаря гибкой системе управления конфигурациями и внедрением зависимостей, которую предоставляет фреймворк. [31]

Одним из ключевых преимуществ использования Spring в проекте было внедрение JPA (Java Persistence API), что позволило абстрагироваться от прямых запросов к базе данных. Сущности JPA и репозитории, расширяющие JPA Repository, упрощают работу с данными, автоматизируя создание и обработку запросов к базе данных без необходимости написания сложного SQL-кода. Это значительно упрощает разработку и поддержку приложения, сокращая время на разработку и потенциальные ошибки в обработке данных. [26]

В качестве базы данных была выбрана легковесная H2 database, которая идеально подходит для разработки и тестирования благодаря своей простоте настройки и встроенной поддержке. Эта БД запускается вместе с приложением и не требует отдельной установки, что делает процесс разработки намного проще. При необходимости перехода на традиционную SQL базу данных, достаточно будет просто изменить настройки в файле application.properties, что добавляет гибкости при масштабировании проекта или переходе к более крупной эксплуатационной среде.

Таким образом, выбор фреймворка Spring и его экосистемы для проекта обеспечил высокий уровень модульности и масштабируемости приложения, что является ключевым для поддержания высоких стандартов качества и безопасности в современной разработке программного обеспечения.

Вывод по первой главе

Подводя итог сказанному в первой главе настоящей работы, отметим, что нам удалось:

- описать подходы к автоматизации создания макетов ответов API;
- провести сравнительный анализ существующих программных решений и прийти к выводу о целесообразности разработки сервиса для создания двойных тестовых объектов;
- описать функциональные и нефункциональные требования к

создаваемому сервису;

- сформировать бизнес-цели и требования к проекту по разработке такого сервиса;
- поставить перед собой четко сформулированные задачи;
- выбрать, на основе какого технологического стека будет реализован проект по разработке вышеупомянутого сервиса.

Таким образом, завершено теоретическое обоснование создания Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах, и можем приступить к его моделированию.

Глава 2 Проектирование программного обеспечения Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах

2.1 Выбор методологии проектирования программного обеспечения

Для разработки сервиса была выбрана методология RAD (Rapid Application Development), которая позволяет значительно сократить время на разработку и быстро адаптироваться к изменениям требований.

RAD (Rapid Application Development) представляет собой гибкий подход к разработке программного обеспечения, акцентирующий внимание на быстрой разработке и итеративном процессе создания приложений. Основные принципы методологии RAD включают в себя:

- быстрая разработка: Использование инструментов и технологий, позволяющих ускорить процесс разработки программного обеспечения.
- итеративный процесс: Проектирование и разработка ПО происходят в несколько итераций, каждая из которых завершается созданием работоспособной версии продукта.
- пользовательская вовлеченность: Активное участие конечных пользователей в процессе разработки для обеспечения соответствия продукта их ожиданиям и требованиям.

Графически, процесс разработки программного обеспечения по методологии RAD может быть представлена следующим образом.

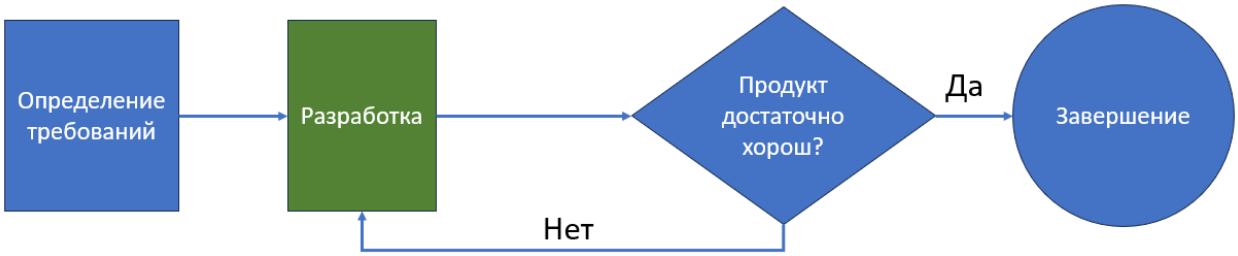


Рисунок 3 – Методология разработки RAD

Использование методологии RAD для разработки Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах позволило достичь следующих преимуществ:

- сокращение времени разработки: Быстрое создание работоспособных версий системы позволило сократить общее время на разработку.
- улучшенное соответствие требованиям: Регулярная обратная связь от пользователей обеспечила высокую степень соответствия системы их ожиданиям и потребностям.
- гибкость и адаптивность: Возможность быстро вносить изменения в проект на основе новых требований и идей.

В заключение отметим, что для команды, состоящей из одного человека, методологию RAD автор счел оптимальной. Ее применение позволило обеспечить высокую скорость и гибкость разработки, и позволило отказаться от применения командных инструментов разработки, что было бы избыточно.

2.2 Логическое моделирование программного обеспечения

Цель проекта: Разработка информационной системы для управления заглушками (mocks) в разработке программного обеспечения, что позволяет моделировать поведение компонентов системы в условиях, когда некоторые

из них либо недоступны, либо ещё не реализованы. [18]

Отличительными особенностями проекта являются возможности:

– создание заглушек различных типов, таких как JSON, XML, SOAP и другие форматы данных. Это позволит разработчикам эффективно моделировать работу компонентов системы;
– настройка таймаута для заглушек. Эта функция позволит устанавливать произвольные временные задержки для заглушек, что поможет в тестировании и отладке приложения;
– организация системы журналирования для отслеживания всех событий, происходящих в приложении. Благодаря этой подсистеме разработчики смогут получить полное представление о работе приложения и эффективно реагировать на возможные проблемы. [9]

Опишем требуемую функциональность приложения в графическом виде, с использованием диаграммы вариантов использования (диаграммы прецедентов).

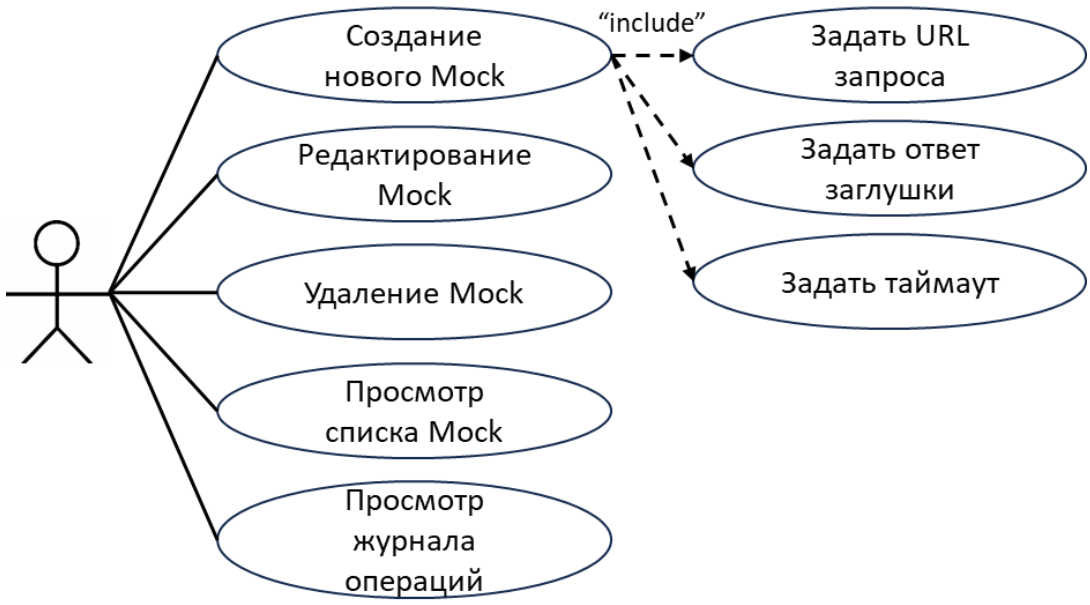


Рисунок 4 – Диаграмма вариантов использования (диаграмма прецедентов) для разрабатываемого приложения MockManager

Как видно из представленной диаграммы, программа обладает всем необходимым функционалом, при этом довольно проста.

В ходе дальнейшего развития приложения (за рамками данного дипломного проекта), имеет смысл добавить управление правами пользователей, а также поддержку скриптовых языков, например Groovy, для обработки поступающей в запросе информации и подстройки шаблона под требуемые параметры.[24] Однако, трудозатраты на такой функционал слишком велики для одного человека и его невозможно реализовать в разумное время.

Для того, чтобы непосредственная реализация приложения была максимально быстрой, необходимо заранее продумать его структуру. Обычно результаты размышлений представляются в графическом виде, путем создания UML диаграммы классов. Такая диаграмма демонстрирует существующие в приложении классы, их взаимосвязь и структуру каждого класса (поля и методы).

Особенно важно создание таких диаграмм в больших командах, включающих специалистов разного профиля, поскольку они позволяют легко представить, как будет работать приложение даже людям, не имеющим специального образования.

Рассмотрим структуру создаваемого приложения, представив ее в виде UML-диаграммы классов на рисунке 5 ниже.

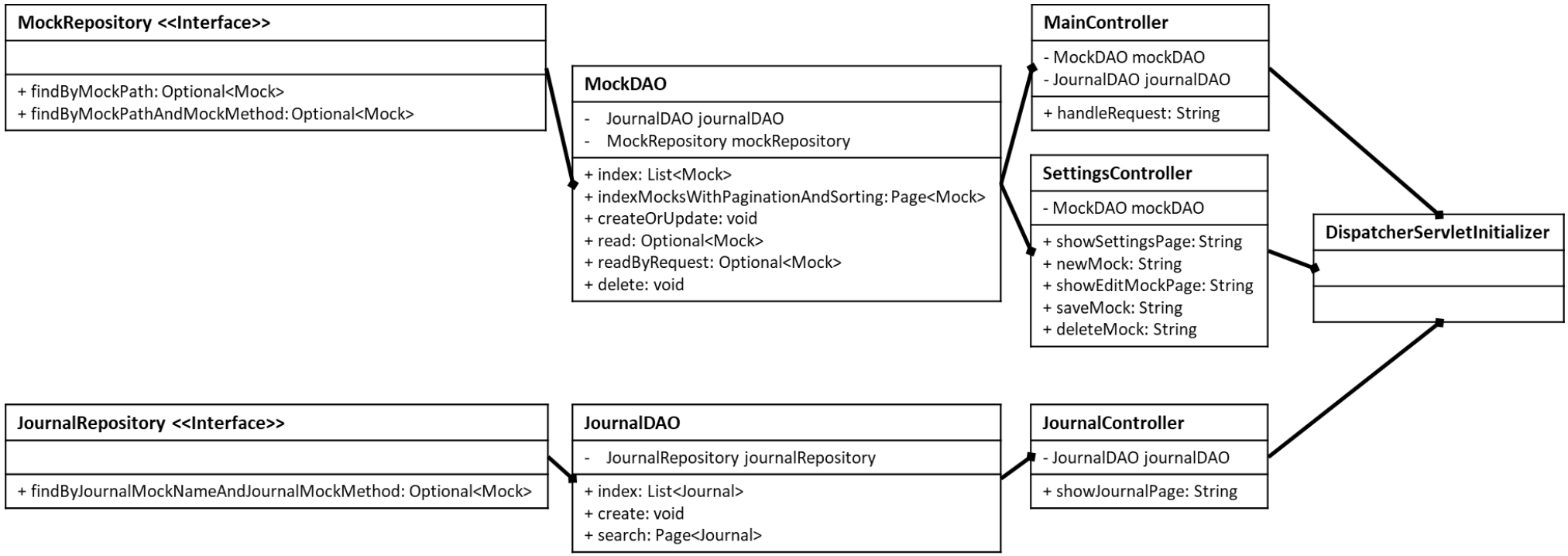


Рисунок 5 – UML диаграмма классов разрабатываемого приложения MockManager

Подробнее опишем информацию, представленную в диаграмме классов на рисунке 5, который находится на предыдущей странице.

Программа имеет два специализированных интерфейса для работы с базой данных – MockRepository и JournalRepository, каждый из которых отвечает за работу с конкретной таблицей. Эти интерфейсы дополнены специфическими методами, которые призваны обеспечить возможность сортировки. [12]

Классы DAO служат прослойкой между пользовательским веб-интерфейсом и вышеописанными Repository, что помогает ограничить доступные пользователю методы взаимодействия с базами данных с целью безопасности. [2]

Классы Controller отвечают за обработку непосредственно действий пользователя на веб-странице, возвращая адрес искомой страницы в формате String. [29]

DispatcherServletInitializer отслеживает действия пользователя и передает их в соответствующий контроллер.

На рисунке 6 представлен снимок экрана, демонстрирующий работу созданного программного продукта. Для тестирования применялась специализированная программа – Postman, позволяющая имитировать API-запросы и анализировать получаемый ответ.

Postman практически не применяется при автоматизированном тестировании, но широко используется при ручном, а значит, позволяет наглядно демонстрировать, какое сообщение мы отправляем, имитируя запрос бэкенд-части нашего локального приложения, и какое мы получаем в ответ от удаленного сервиса.

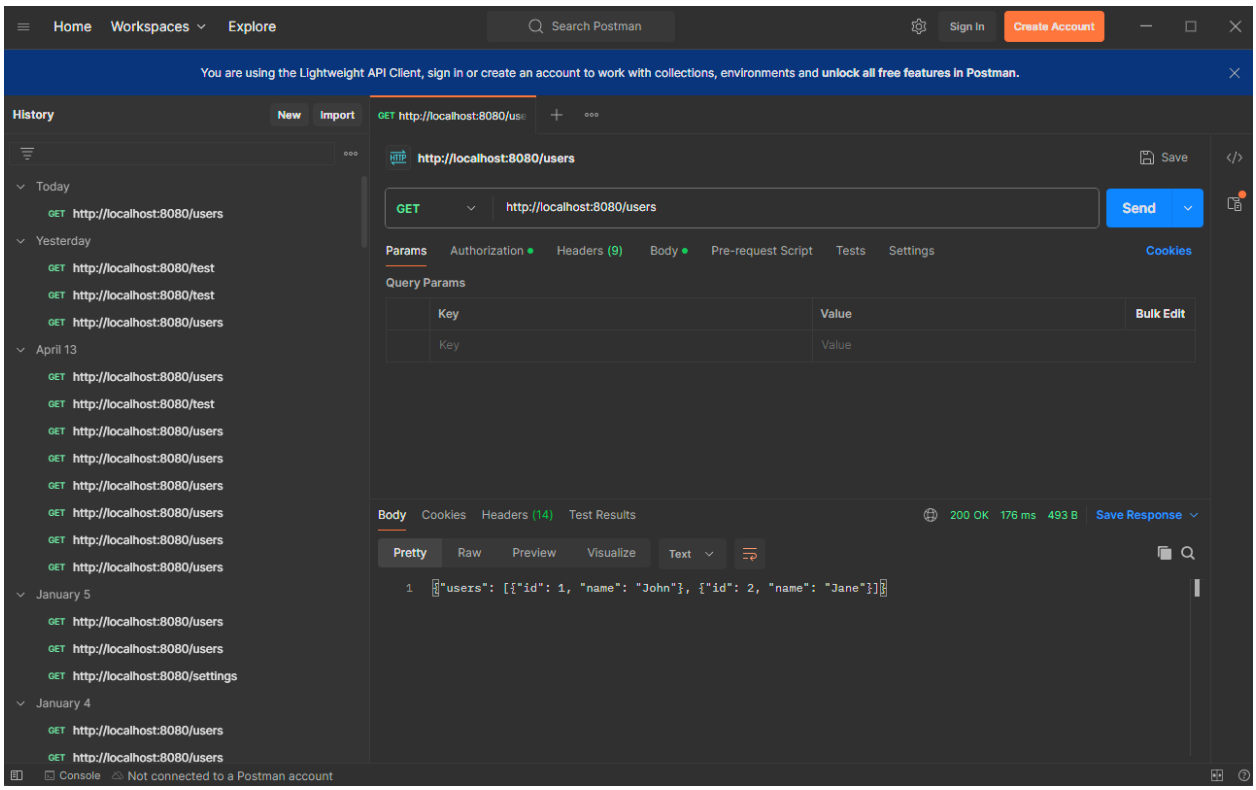


Рисунок 6 – Демонстрация работы приложения с помощью приложения Postman (заглушка возвращает сохраненный ответ)

Как можно видеть из представленного рисунка, GET-запрос на URL “/users” приводит к возврату заранее положенного в заглушку файла JSON, то есть, программа работает корректно.

2.3 Архитектура и особенности реализации Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах

2.3.1 Архитектурный подход

Проект «MockManager» простроен на использовании монолитной архитектуры.[7] Этот подход упрощает процессы разработки и поддержки. В условиях разработки в одиночку, это существенно ускоряет процессы CI/CD (Continuous Integration/Continuous Deployment). [13, 28]

Рисунок 6 демонстрирует структуру проекта «MockManager», где

каждый класс представлен в виде отдельного компонента, со строго определенной функциональностью. Такая модульная организация обеспечивает легкость внесения изменений и обновлений в систему, минимизируя риски возможных сбоев и повышая общую надежность архитектуры.

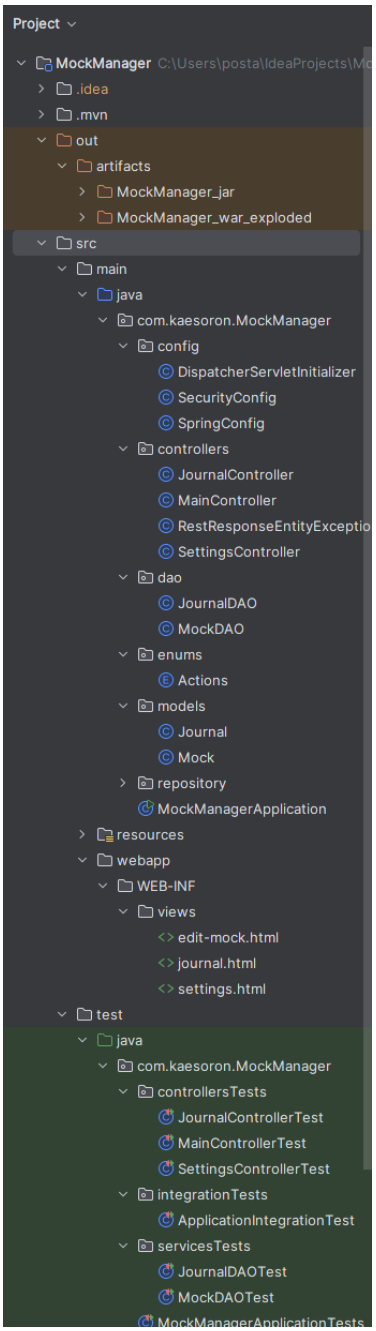


Рисунок 7 – Структура проекта MockManager

Как можно видеть из приведенной структуры, проект делится на 2 основных раздела: сам продукт (каталог main) и тестовое покрытие проекта (папка tests).

Структура проекта соответствует современным стандартам программирования и включает в себя несколько основных элементов: [6]

- важными компонентами структуры проекта являются файлы конфигурации, которые определяют настройки и параметры работы приложения;
- другим важным элементом структуры проекта является слой представления, включающий веб-страницы, написанные с помощью шаблонизатора. Этот слой отвечает за отображение информации для пользователей и взаимодействие с ними;
- кроме того, в структуре проекта присутствует слой контроллеров, ответственных за обработку событий на страницах. Они обеспечивают связь между пользовательским интерфейсом и бизнес-логикой приложения;
- не менее важным элементом структуры проекта является слой DAO, который предназначен для передачи запросов от контроллеров в репозитории. Этот слой обеспечивает доступ к данным и исключает возможность недокументированного взаимодействия с репозиториями;
- репозиторий играет ключевую роль в обеспечении взаимодействия между приложением и базой данных. На диаграмме он представлен свернутым, чтобы упростить визуализацию. Этот компонент отвечает за выполнение запросов к базе данных, обеспечивая доступ к нужной информации и обновление данных при необходимости.

Для создания таблиц в базе данных в соответствии с описанной в классах схемой (Journal, Mock) необходимы файлы сущностей. Эти файлы содержат описания объектов, которые будут представлены в базе данных в

виде таблиц. Каждая сущность соответствует определенному объекту в приложении и содержит информацию о его свойствах и связях с другими объектами.

При разработке приложения важно правильно структурировать репозиторий и определить сущности, которые будут храниться в базе данных. Это поможет обеспечить эффективное взаимодействие с базой данных и улучшить производительность приложения. Кроме того, правильно спроектированный репозиторий позволит удобно работать с данными и обеспечит возможность масштабирования при необходимости.

2.3.2 Технологический стек

представляет собой набор инструментов, фреймворков и технологий, используемых при создании приложения. От тщательно продуманного набора технологий зависит не только качество и производительность разрабатываемого продукта, но и удобство его поддержки и масштабирования.

В данном конкретном проекте выбраны следующие инструменты:

- Spring Boot – для создания RESTful сервисов;
- H2 Database – встраиваемая база данных для упрощения разработки и тестирования;
- Thymeleaf и Bootstrap – для создания динамичных и адаптивных веб-страниц;
- JUnit и Mockito – для модульного и интеграционного тестирования.

Spring Boot на момент написания настоящей работы является де-факто стандартом для создания веб-приложений, основанных на Java. Организация связности компонентов приложения с использованием аннотаций создает высокий уровень удобства и делает избыточным много строк кода, которые были бы необходимы, если бы Spring не использовался.

H2 Database – самое легкое в использовании решение из известных автору для организации базы данных SQL. Использование этой технологии

позволяет избежать «поднятия» отдельной базы данных и выделения ей собственного URL: база данных запускается одновременно с приложением и работает только пока то запущено.

Для создания фронтэнд-части выбрана технология Thymeleaf, поскольку автор не владеет JavaScript. Традиционный HTML не подходит, поскольку сервис требует динамического изменения содержимого страниц в соответствии с изменениями в базе данных. Таким образом, выбор инструментов на самом деле невелик. Bootstrap применяется буквально в нескольких местах для расширения функционала уже технологии Thymeleaf.

Для тестирования сервиса можно было бы обойтись ручными проверками, но поскольку было решено создавать проект в соответствии с современными стандартами, решено использовать автотесты, автоматически запускающиеся при наступлении определенного события. Для написания автотестов фактическим стандартом являются JUnit и Mockito, а для автозапуска – CI/CD pipelines, которые можно настроить непосредственно в репозитории GitHub. Таким образом, при появлении ошибок в работе сервиса, мы узнаем об этом практически сразу после внесения изменений в проект. [20, 21]

Теперь рассмотрим, какие именно зависимости применялись в проекте для того, чтобы он заработал. Для имплементации зависимостей автор использовал Maven, а не Gradle, хотя и функционал, на взгляд автора, очень похож и отличается в основном синтаксис.

Рассмотрим далее все использованные зависимости, условно разбив их на несколько категорий, и опишем их важность для разработки нашего сервиса.

Основные зависимости. Необходимо описать зависимости, создающие основу сервиса:

- Spring Boot Starter Web (org.springframework.boot:spring-boot-starter-web): Эта зависимость является стандартной для создания веб-

приложений на базе Spring MVC, а наше приложение – именно такое. Стартер позволяет использовать встроенный Tomcat как HTTP сервер, что значительно упрощает развертывание приложения. Автора такой подход полностью устраивает;

– Spring Boot Starter Security (org.springframework.boot:spring-boot-starter-security): обеспечивает интеграцию в приложение Spring Security, что позволяет добавить функционал аутентификации и авторизации. Ранее в работе упоминалось, что авторизация в приложении как раз планируется. Не менее важно то, что два стартера от одного разработчика не конфликтуют, то есть, например, с Thymeleaf Starter комбинировать функционал не получилось бы;

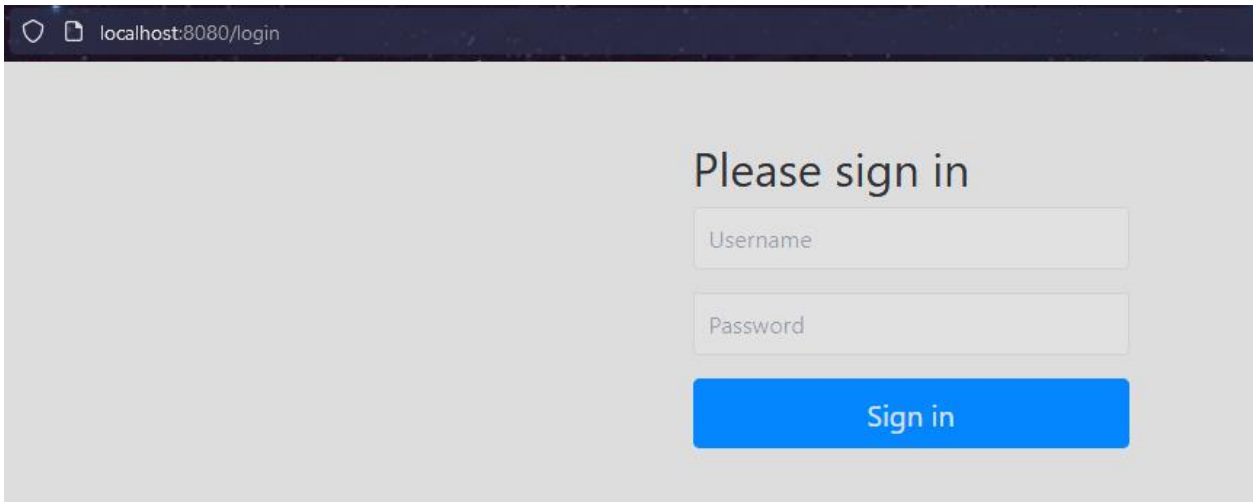


Рисунок 8 – Демонстрация окна авторизации для доступа к сервису, созданного с помощью Spring Security

– Spring Boot Starter Data JPA (org.springframework.boot:spring-boot-starter-data-jpa): еще один стартер от того же разработчика, который существенно упрощает работу с доступом к данным. Он включает поддержку Hibernate, Spring Data JPA и HikariCP в качестве пула соединений к базе данных, что повышает производительность и

эффективность работы с данными. В данном проекте интересен в основном за счет ряда полезных аннотаций;

- H2 Database (com.h2database:h2): как уже упоминалось выше, это встраиваемая непосредственно в приложение база данных, предоставляющая возможность работать без необходимости развертывания отдельного сервера баз данных. Это позволило избавиться от необходимости использовать тяжеловесные MySQL или PostgreSQL, обеспечивая нас такой же функциональностью;
- Jackson Databind (com.fasterxml.jackson.core:jackson-databind): предназначена для десериализации JSON данных в Java объекты (т.н. POJO) и сериализации Java объектов обратно в JSON. Без этого или сходного парсера (например, GSON), Java приложению очень сложно работать с объектами такого формата;
- Thymeleaf (org.thymeleaf:thymeleaf): это шаблонизатор для веб-страниц, который позволяет динамически изменять их содержание, если они написаны на HTML. В данном случае, это незаменимый инструмент, иначе, например, при добавлении новой заглушки пришлось бы каждый раз переписывать код страницы.

Тестирование:

- Spring Boot Starter Test (org.springframework.boot:spring-boot-starter-test): Этот стартер необходим нам, поскольку мы планируем использовать автотесты;
- Hamcrest (org.hamcrest:hamcrest): библиотека, которая предоставляет возможности для создания утверждений в тестах. Это очень удобно для проверки данных, получаемых при выполнении тестов: например, можно проверить, что полученные данные соответствуют чему-то, больше, меньше или пусты;

Вспомогательные зависимости:

- SLF4J API (org.slf4j:slf4j-api): эта зависимость нужна нам для

организации нормального логирования в создаваемом нами сервисе. Также стоит отметить, что это одно из самых популярных решений для логирования;

– Hibernate Core (org.hibernate.orm:hibernate-core): один из ведущих фреймворков для объектно-реляционного отображения (ORM). Нам он нужен для легкой реализации Entity-сущностей с помощью аннотаций и позволяет избежать ручного создания базы данных и ручного прописывания SQL-запросов.

Можно было бы также применить Lombok, чтобы избежать создания геттеров и сеттеров для каждого поля, однако автор считает это избыточным для проекта на его текущей стадии развития.

Мы рассмотрели все зависимости, применяемые автором при разработке Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах. Автор при выборе зависимостей старался руководствоваться принципами разумности и целесообразности [15, 16], чтобы, с одной стороны, обеспечить весь необходимый функционал, а с другой, не переусложнить проект внедрением излишних зависимостей.

2.4 Моделирование данных Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах

Информационная модель представляет собой основу для построения базы данных, включающую в себя различные компоненты и сущности. Обычно представляют физическую и информационную модели базы данных, но в данном случае таблиц будет всего две, и, по мнению автора, физическую модель можно опустить.

На рисунке 8 представлена информационная модель базы данных в

графическом виде.

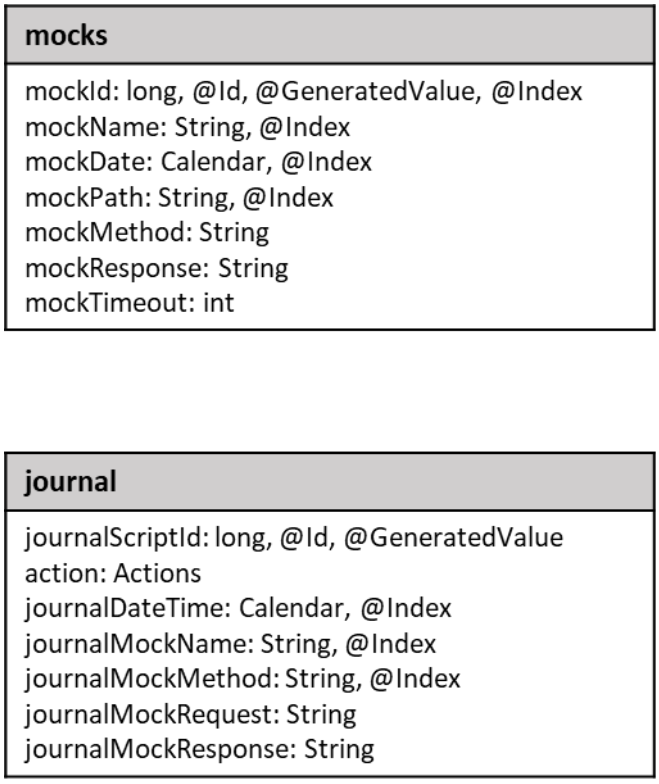


Рисунок 9 – Информационная модель базы данных

Как видно из представленной на рисунке 7 модели, в базе данных всего две таблицы и они не связаны между собой. Это не ошибка автора, но сделано специально. Конечно, было бы проще хранить в Journal объект типа Mock, но тогда журнал содержал бы некорректные записи для измененных и удаленных заглушек.

Недостатком же такого построения базы данных является дублирование части полей: mockName, mockMethod, mockResponse.

Для создания базы данных использовался функционал Java Persistence API (JPA), который позволяет вместо ручного создания таблиц базы данных, использовать классы сущностей, которые подробнее рассмотрим ниже.

Один из ключевых элементов этой модели – Mock, который является виртуальным представлением реального компонента. Реализованный в виде

сущности (@Entity), объект содержит несколько важных атрибутов, таких как идентификатор, имя, метод запроса, путь запроса, тело ответа и задержка ответа. Он позволяет имитировать поведение реальных компонентов внутри системы, что очень полезно при разработке и тестировании программного обеспечения.

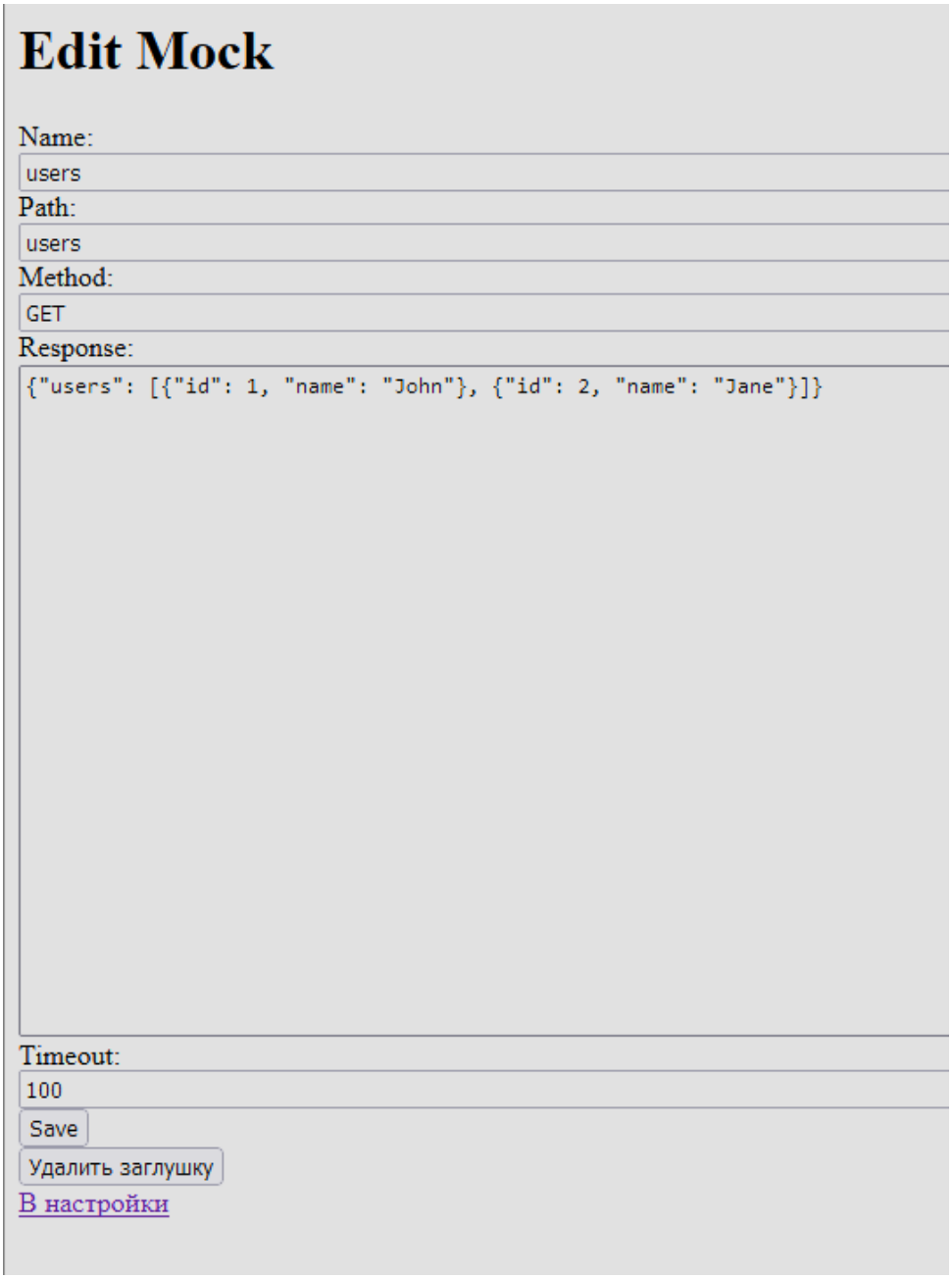


Рисунок 10 – Редактирование существующей заглушки

Для создания новой заглушки, используется точно такая же форма, но не содержащая никаких данных.

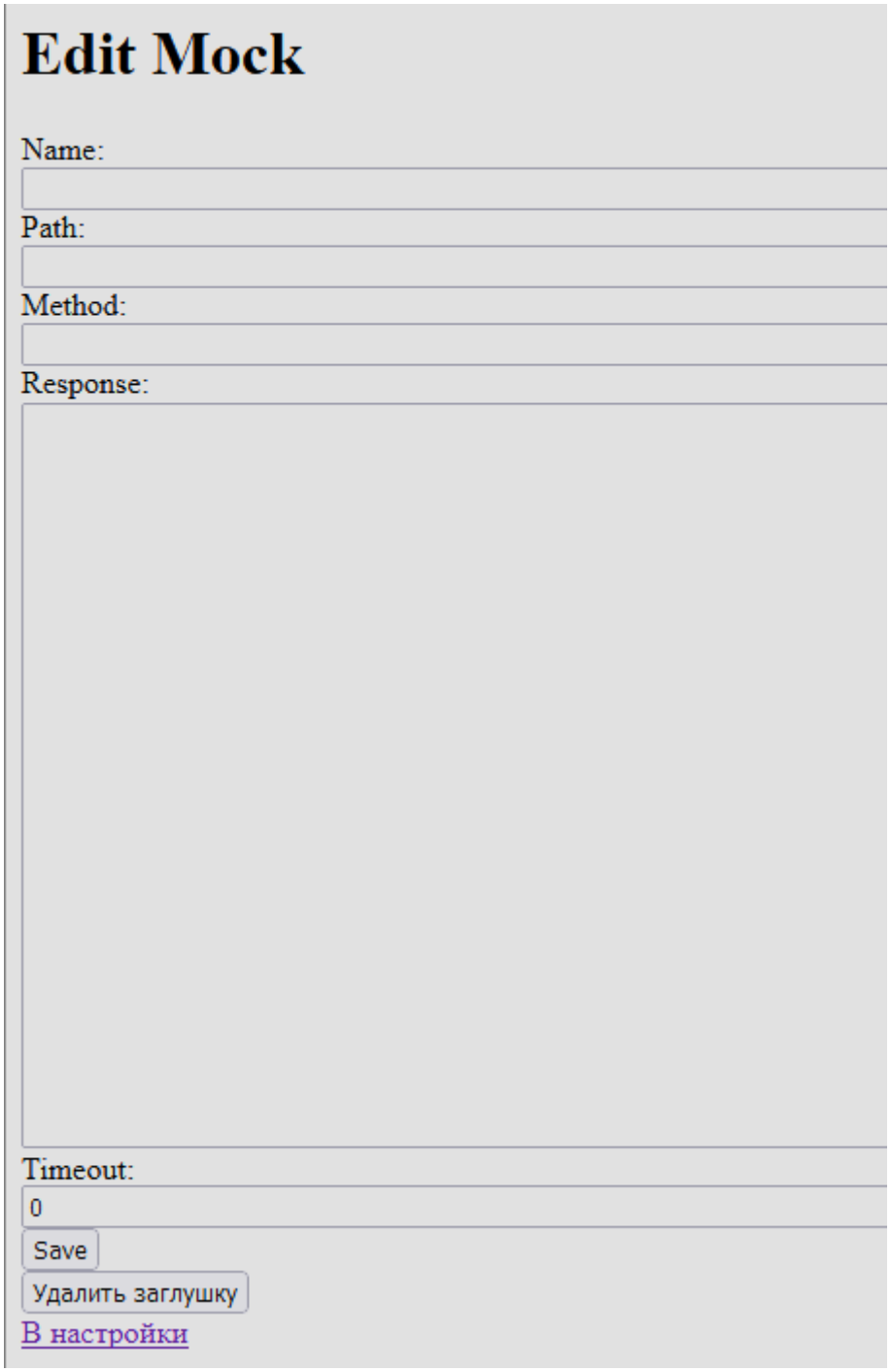
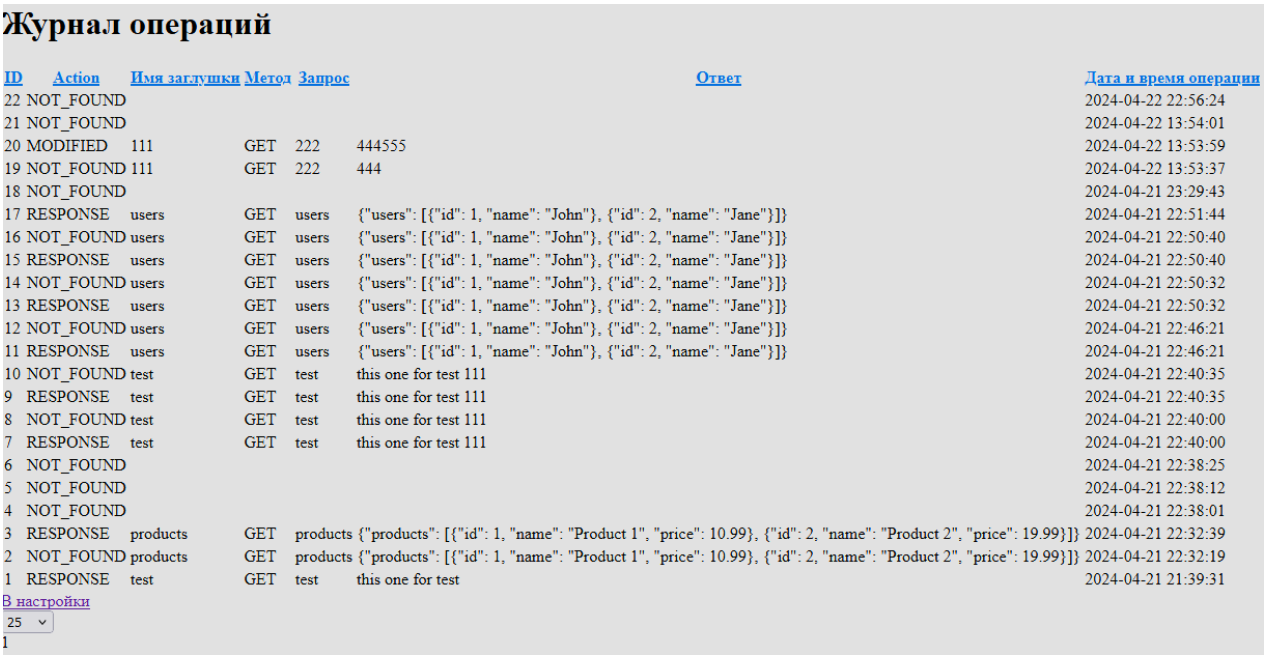


Рисунок 11 – Создание новой заглушки

Еще одним важным элементом информационной модели является Journal, или журнал событий. Этот журнал фиксирует все операции,

проводимые с объектами Mock, и содержит информацию о идентификаторе записи, названии заглушки, совершенном действии, а также дате и времени этого действия. Журнал событий играет важную роль в отслеживании и анализе действий, происходящих в системе, что помогает разработчикам и администраторам лучше понимать происходящие процессы.

Фронтенд-представление журналирования всех действий с моками выглядит следующим образом:



Журнал операций

ID	Action	Имя заглушки	Метод	Запрос	Ответ	Дата и время операции
22	NOT_FOUND					2024-04-22 22:56:24
21	NOT_FOUND					2024-04-22 13:54:01
20	MODIFIED	111	GET	222	444555	2024-04-22 13:53:59
19	NOT_FOUND	111	GET	222	444	2024-04-22 13:53:37
18	NOT_FOUND					2024-04-21 23:29:43
17	RESPONSE	users	GET	users	{"users": [{"id": 1, "name": "John"}, {"id": 2, "name": "Jane"}]}	2024-04-21 22:51:44
16	NOT_FOUND	users	GET	users	{"users": [{"id": 1, "name": "John"}, {"id": 2, "name": "Jane"}]}	2024-04-21 22:50:40
15	RESPONSE	users	GET	users	{"users": [{"id": 1, "name": "John"}, {"id": 2, "name": "Jane"}]}	2024-04-21 22:50:40
14	NOT_FOUND	users	GET	users	{"users": [{"id": 1, "name": "John"}, {"id": 2, "name": "Jane"}]}	2024-04-21 22:50:32
13	RESPONSE	users	GET	users	{"users": [{"id": 1, "name": "John"}, {"id": 2, "name": "Jane"}]}	2024-04-21 22:50:32
12	NOT_FOUND	users	GET	users	{"users": [{"id": 1, "name": "John"}, {"id": 2, "name": "Jane"}]}	2024-04-21 22:46:21
11	RESPONSE	users	GET	users	{"users": [{"id": 1, "name": "John"}, {"id": 2, "name": "Jane"}]}	2024-04-21 22:46:21
10	NOT_FOUND	test	GET	test	this one for test 111	2024-04-21 22:40:35
9	RESPONSE	test	GET	test	this one for test 111	2024-04-21 22:40:35
8	NOT_FOUND	test	GET	test	this one for test 111	2024-04-21 22:40:00
7	RESPONSE	test	GET	test	this one for test 111	2024-04-21 22:40:00
6	NOT_FOUND					2024-04-21 22:38:25
5	NOT_FOUND					2024-04-21 22:38:12
4	NOT_FOUND					2024-04-21 22:38:01
3	RESPONSE	products	GET	products	{"products": [{"id": 1, "name": "Product 1", "price": 10.99}, {"id": 2, "name": "Product 2", "price": 19.99}]}	2024-04-21 22:32:39
2	NOT_FOUND	products	GET	products	{"products": [{"id": 1, "name": "Product 1", "price": 10.99}, {"id": 2, "name": "Product 2", "price": 19.99}]}	2024-04-21 22:32:19
1	RESPONSE	test	GET	test	this one for test	2024-04-21 21:39:31

В настройки

25

1

Рисунок 12 – Демонстрация работы журналирования

Удаление записей из журнала невозможно [15], в связи с чем на рисунке виден процесс устранения некоторых багов при создании приложения.

Создание, редактирование, удаление и просмотр объектов Mock доступно из основного интерфейса приложения (см. рисунок 10).

Вывод по второй главе

По итогам второй главы настоящей выпускной квалификационной работы (бакалаврской работы), удалось сделать следующее:

- выбрать методологию проектирования создаваемого программного обеспечения. Была выбрана методология RAD (Rapid Application Development), как наиболее подходящая для команды, состоящей из единственного человека;
- провести логическое моделирование создаваемого Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах, включая диаграмму прецедентов и UML диаграмму классов;
- определиться с архитектурным подходом (монолитное приложение);
- выбрать технологический стек для реализации проекта (Spring Boot, H2 Database, Thymeleaf, Bootstrap, JUnit, Mockito);
- осуществить моделирование данных, определив структуру создаваемой базы данных;
- провести прототипирование – эскизы будущих страниц нашего сервиса в том виде, который они должны принять по окончании проекта.

Таким образом, теперь мы готовы к практической реализации Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах, а также к этапу тестирования программного продукта после его создания, чтобы убедиться в его соответствии поставленной в 1 главе настоящей бакалаврской работы (выпускной квалификационной работы) задаче.

Глава 3 Реализация и тестирование Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах

3.1 Реализация Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах

Для успешной реализации сервиса было необходимо тщательно разработать компоненты системы, которые будут обеспечивать ее функциональность и эффективность.

Одним из ключевых компонентов является MockDAO – класс, который отвечает за взаимодействие с базой данных для операций над объектами Mock. Этот компонент играет важную роль в обеспечении доступа к данным и их обновлении.

Другим важным компонентом является JournalDAO, который предназначен для работы с журналом событий. Задачей этого класса является фиксация всех операций, связанных с заглушками. Благодаря JournalDAO осуществляется отслеживание всех изменений и действий, что позволяет проводить анализ работы системы и выявлять потенциальные проблемы.

Не менее важным является также SettingsController – контроллер, который обрабатывает пользовательские запросы для управления заглушками, переданные через веб-интерфейс. Этот компонент обеспечивает взаимодействие пользователя с системой, позволяя управлять данными и настройками. Правильная настройка SettingsController позволяет пользователям эффективно управлять системой и осуществлять необходимые операции с моками.

Однако для успешной реализации проекта необходимо не только разработать и интегрировать эти компоненты, но и обеспечить их взаимодействие и совместную работу. Каждый из компонентов должен быть настроен и оптимизирован для эффективной работы в рамках системы в

целом. Только тщательное планирование, разработка и интеграция всех компонентов позволят достичь желаемого результата и обеспечить стабильную работу системы.

3.2 Тестирование Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах

В процессе разработки программного обеспечения, прототипирование и тестирование являются важнейшими и необходимыми этапами. [14]

Прототипирование представляет собой создание моделей или образцов интерфейса, которые помогают лучше понять функциональность и внешний вид будущей системы. [3] В контексте веб-разработки прототипирование часто осуществляется с использованием специализированных инструментов, позволяющих создавать демонстрационные интерфейсы для системы. Прототипирование подробнее расписано во 2 главе настоящей работы.

В данном случае, выполнена разработка веб-интерфейса на Thymeleaf. При создании было важно уделить внимание не только внешнему виду, но и функциональности. Например, добавление форм для создания, редактирования и удаления заглушек, а также реализация страницы журнала событий, в сущности, представляют собой всю необходимую для проекта функциональность.

Историю прототипирования и последовательное развитие приложения можно увидеть, если посмотреть историю внесения изменений (коммитов) для соответствующего репозитория на сайте GitHub, содержащем исходный код приложения.

Теперь рассмотрим, как выглядит основная страница разработанного приложения, для чего используем рисунок 13 ниже.

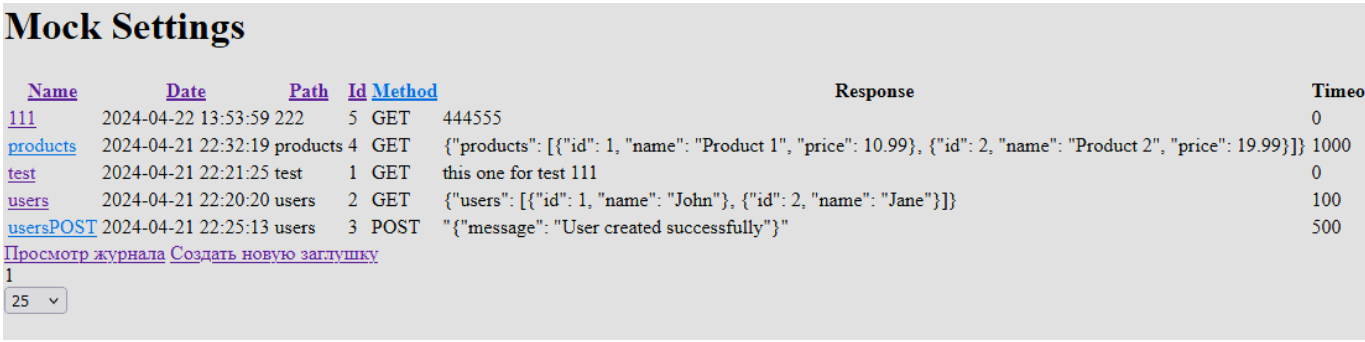


Рисунок 13 – Веб-интерфейс приложения со списком заглушек

Как видно из представленного буквально парой строк ранее рисунка 13, основная страница позволяет:

- осуществлять сортировку списка моков по значениям столбцов (при нажатии на название столбца);
- переходить в режим редактирования заглушки (при нажатии на ее название);
- просматривать журнал событий;
- создать новую заглушку.

Кроме того, на странице реализована пагинация (вывод списка заглушек не единым списком, а с разбивкой на страницы) и возможность настраивать количество записей на странице.

После завершения прототипирования, наступает этап тестирования, который не менее важен.

В данном проекте применены две разновидности тестов: модульные и интеграционные.

Модульные тесты позволяют проверить правильность работы отдельных компонентов системы, таких как MockDAO и JournalDAO. Использование инструментов, таких как JUnit и Mockito, помогает обеспечить корректное функционирование бизнес-логики и избежать ошибок на ранних этапах разработки. [22]

Интеграционные тесты позволяют проверить взаимодействие

различных компонентов системы и удостовериться в корректности работы веб-интерфейса. С использованием инструментов, таких как Spring Boot Test, можно эффективно проверить функциональность системы в целом и обнаружить возможные проблемы в работе интерфейса при взаимодействии с другими компонентами. В итоге, прототипирование и тестирование совместно способствуют созданию качественного и надежного программного продукта. [17, 25]

Рисунок 14 – Пример прохождения автотестов для проверки работы модуля SettingsController

Настройка автотестов позволяет отказаться от ручного тестирования и запускать тесты автоматически, например, при каждом внесении изменений (push) в ветку Git. Для этого требуется корректно настроить CI/CD автоматизированную последовательность (pipeline).

Вывод по третьей главе

Третья глава настоящей выпускной квалификационной работы (бакалаврской работы) посвящена непосредственной практической реализации Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах и его тестированию.

Также были созданы автоматизированные тесты, которые покрывают практически весь функционал созданной программы. Написанные тесты можно условно разделить на модульные (для тестирования взаимодействия отдельных классов) и интеграционные (для тестирования отдельных функций сервиса, включая взаимодействие всех необходимых компонентов).

Созданный сервис соответствует заранее составленным планам и схемам.

Заключение

В заключительной части выпускной квалификационной работы стоит особо подчеркнуть, что разработка Spring Boot Restful сервиса для создания тестовых двойных объектов в автоматизированных модульных тестах имеет значительный потенциал применения в российских компаниях, производящих программное обеспечение. Этот сервис позволяет эмулировать ответы от сторонних сервисов и баз данных, и является уникальной российской разработкой, однако, имеющей иностранные аналоги.

Внедрение разработанного автором сервиса в повседневную деятельность IT-компаний по разработке программного обеспечения должно значительно повысить эффективность и скорость тестирования за счет минимизации времени, необходимого для настройки и поддержки тестовых сред. Система предлагает ряд настроек, которые позволяют уже сейчас использовать сервис по прямому функциональному назначению.

Кроме того, целью выпускной квалификационной работы было не только создать инструмент, способный улучшить качество тестирования, но и обеспечить легкое взаимодействие между разработчиками и тестировщиками, что очень важно в современной среде разработки программного обеспечения, где сжатые сроки и высокое качество играют решающую роль. Легкое взаимодействие, по задумке автора, будет обеспечено за счет того, что разработчикам (или, возможно, DevOps) придется всего однажды установить и настроить приложение, а пользоваться им можно и не имея специальных навыков.

При создании сервиса, соблюдены современные стандарты по разработке и тестированию программного обеспечения. Программный код минимизирован, зависимости тщательно отобраны, процессы тестирования автоматизированы, программа находится в открытом доступе, что дает возможность развивать ее всем желающим.

Отметим, что в процессе разработки сервиса все поставленные цели были успешно достигнуты: начиная от подготовительных этапов (проектирования системной архитектуры и подготовки информационной модели базы данных) и заканчивая непосредственной реализацией и тестированием.

Результаты модульного (юнит) и интеграционного тестирования, а также анализ приложений-аналогов позволяют утверждать, что проект не только способен осуществлять задуманный при создании функционал, но и обладает потенциалом для дальнейшего развития и масштабирования. В сущности, на базе данного программного продукта можно строить бизнес, необходима только заинтересованность российских компаний в продукте.

Таким образом, Spring Boot Restful сервис для создания тестовых двойных объектов в автоматизированных модульных тестах, названный автором «MockManager» демонстрирует, как тщательно продуманный подход к разработке программного обеспечения, основанный на инновационных технологиях и интеграции ключевых функциональных возможностей, может привести к созданию мощного инструмента, который не только улучшает качество и скорость разработки продуктов, но и повышает уровень удовлетворенности и профессионального развития сотрудников.

Список используемой литературы и используемых источников

1. Балабанов Т.В. «Проектирование информационных систем». Москва: Финансы и статистика, 2019.
2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Д. «Приёмы объектно-ориентированного проектирования. Паттерны проектирования». СПб: Питер, 2016.
3. Иванов А.П., Смирнов С.А., Терехов А.Н. «Технологии программирования». СПб: БХВ-Петербург, 2018.
4. Кулигин С.В. «Управление программными проектами». Москва: Финансы и статистика, 2019.
5. Левин М.Р. «Управление разработкой информационных систем». Москва: Радио и связь, 2017.
6. Пантелеев А.В., Сенин И.В. «Язык программирования Java». Москва: Диалектика, 2021.
7. Попов Е.В. «Информационные системы и технологии». Москва: Юрайт, 2020.
8. Резчиков А.В. «Тестирование программного обеспечения». СПб: БХВ-Петербург, 2019.
9. Сидоренко В.Н. «Системное программирование». СПб: БХВ-Петербург, 2018.
10. Филлипс К., Терехов А.Н. «Архитектура программного обеспечения». СПб: Питер, 2019.
11. Юсупов Р.М., Новиков Ф.А., Буркова В.В. «Управление проектами: стандарты, методы, опыт». Москва: МИЭМ, 2020.
12. Ясаков С.П., Шукалович А.В. «Программирование в среде Java». Москва: БИНОМ. Лаборатория знаний, 2019.
13. Яковлев С.Н. «Основы разработки веб-приложений». СПб: Питер, 2020.

14. Якушин С.А. «Основы тестирования программного обеспечения». Москва: Форум, 2018.

15. Fowler, M. (2018). «Refactoring: Improving the Design of Existing Code». Addison-Wesley Professional.

16. Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1994). «Design Patterns: Elements of Reusable Object-Oriented Software». Addison-Wesley Professional.

17. Beck, K. (2003). «Test Driven Development: By Example». Addison-Wesley Professional.

18. Crispin, L., Gregory, J. (2009). «Agile Testing: A Practical Guide for Testers and Agile Teams». Addison-Wesley Professional.

19. Martin, R. C. (2008). «Clean Code: A Handbook of Agile Software Craftsmanship». Prentice Hall.

20. Fewster, M., Graham, D. (1999). «Software Test Automation». Addison-Wesley Professional.

21. Astels, D. (2003). «Test Driven Development: A Practical Guide». Prentice Hall.

22. Hunt, A., Thomas, D. (2000). «The Pragmatic Programmer: From Journeyman to Master». Addison-Wesley Professional.

23. Shore, J., Warden, S. (2008). «The Art of Agile Development». O'Reilly Media.

24. Stubblebine, T. (2005). «Regular Expression Pocket Reference: Regular Expressions for Perl, Ruby, PHP, Python, C, Java and .NET». O'Reilly Media.

25. Hohpe, G., Woolf, B. (2003). «Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions». Addison-Wesley Professional.

26. Ambler, S. W., Sadalage, P. J. (2006). «Refactoring Databases: Evolutionary Database Design». Addison-Wesley Professional.

27. Freeman, S., Pryce, N. (2009). «Growing Object-Oriented Software, Guided by Tests». Addison-Wesley Professional.

28. Beck, K., Andres, C. (2004). «Extreme Programming Explained: Embrace Change (2nd Edition)». Addison-Wesley Professional.

29. Kerievsky, J. (2004). «Refactoring to Patterns». Addison-Wesley Professional.

30. Koskela, L. (2007). «Test Driven: TDD and Acceptance TDD for Java Developers». Manning Publications.

31. Coplien, J. O., Bjørnvig, G. (2010). «Lean Architecture: for Agile Software Development». Wiley.