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

Институт математики, физики и информационных технологий
(*Наименование института*)

Кафедра «Прикладная математика и информатика»
(*Наименование кафедры*)

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

Бизнес-информатика
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему «Автоматизация тестирования пользовательского интерфейса»

Студент Н.Ф. Еремин
(И.О. Фамилия) (личная подпись)

Руководитель Н.Н. Казаченок
(ученая степень, звание, И.О. Фамилия)

Тольятти 2021

Аннотация

Бакалаврская работа выполнена на тему «Автоматизация тестирования пользовательского интерфейса».

Цель работы заключается в разработке модуля автоматизированного тестирования пользовательского интерфейса.

Объектом исследования выступает процесс тестирования веб-приложения компании в ООО «Гильдия ПРО».

Предметом исследования выступает автоматизация процесса тестирования пользовательского интерфейса веб-приложений.

Во введении определена актуальность темы, цель работы, задачи для достижения поставленной цели.

В ходе выполнения выпускной квалификационной работы была дана технико-экономическая характеристика организации, произведено концептуальное моделирование предметной области. Определенны технологии и инструменты для проектирования модуля автоматизированного тестирования, разработаны UML диаграммы вариантов использования, разработана логическая и физическая модель данных, подготовлено рабочее окружение для разработки автоматических тестов, разработана и представлена панель управление для создания и поддержки автоматизированных тестов, рассчитана экономическая эффективность.

В заключении подведены итоги, достигнутые в ходе выполнения работы.

Бакалаврская работа состоит из 61 страниц и включает 35 рисунков, 6 таблиц. Список используемой литературы содержит 20 источников.

Оглавление

Введение

Программный продукт является результатом длительной работы команды разработчиков, по итогам которой в разработанном программном обеспечении (ПО) взаимодействуют огромное количество компонентов, что влечет за собой появление различного рода ошибок и недочетов разработанного ПО.

В разработке ПО, помимо главной задачи реализовать функционал, существует не менее важная задача по контролю качество ПО. Для поиска возможных багов и обеспечение качества, одним из надежных способов проверки качества, является внедрение тестирования. Тестирование содержит в себе проверку всех процессов жизненного цикла программного обеспечения, для поиска и описания дефектов, оценки качества разрабатываемого продукта.

За последнее время в области тестировании программного обеспечения, было разработано большое количество практик и методологий для оптимизации процессов тестирования программного обеспечения и увеличения качества работы подразделений тестирования. Одним из популярных решений является внедрение автоматизированного программного обеспечения для проверки работоспособности разрабатываемого продукта.

В автоматизированном тестировании ПО, вся работа по подготовке тестовых данных, запуска и выполнения шагов для проверки качества во время тестирования осуществляется при помощи инструментов автоматизации тестирования, которые эмулируют поведения пользователя при его взаимодействии с пользовательским интерфейсом программы.

Актуальность темы обусловлена необходимостью автоматизировать тестирование пользовательского интерфейса, благодаря автоматизации которого увеличиться тестовое покрытие, будет сэкономлено время на

тестирование старого функционала, сокращено время на обнаружения ошибок и ускориться время их исправления.

Цель работы заключается в разработке модуля автоматизированного тестирования пользовательского интерфейса.

Для достижения цели необходимо решить следующие задачи:

- исследовать автоматизированное тестирование;
- спроектировать модуль автоматизированного тестирования;
- реализовать модуль автоматизированного тестирования.

Объектом исследования выступает процесс тестирования веб-приложения компании в ООО «Гильдия ПРО».

Предметом исследования выступает автоматизация процесса тестирования пользовательского интерфейса веб-приложений.

Бакалаврская работа состоит из ведения, основной части (3 главы), заключения и списка используемой литературы.

В первой главе приведена технико-экономическая характеристика организации, разобрано понятие тестирования программного обеспечения, составлены преимущества и недостатки автоматизированного тестирования, разработана концептуальная модель тестирования, определены требования к разрабатываемому инструменту.

Во второй главе производится процесс проектирования. Выбираются технологии и инструменты для проектирования системы, производится моделирование системы тестирования, разрабатывается логическая и физическая модели данных.

В третьей главе произведен выбор технологий для разработки модуля автоматизированного тестирования, описан процесс разработки, представлен основной функционал и результаты тестирования разработанного модуля.

Глава 1 Анализ автоматизации тестирования

1.1 Технико-экономическая характеристика организации ООО «Гильдия ПРО»

Компания «Гильдия ПРО» создана в 2012 году, ее составляют профессионалы своего дела, имеющие за плечами многолетний опыт работы в web-индустрии. Штат сотрудников включает web-дизайнеров, проектировщиков интерфейсов, программистов, seo-специалистов, копирайтеров. Опытные проект-менеджеры обеспечивают успешное выполнение работ на всех этапах и точное соблюдение сроков.

Компания предлагает наиболее удобный, простой и красивый способ решения поставленной задачи.

Основная деятельность:

- IT-аутсорсинг. Компания формирует команду, которая будет работать с проектом, избавляя от необходимости содержать штат IT-специалистов. Предлагает пути развития проекта, выступая в роли экспертов в области тех-разработки, дизайна и юзабилити. Руководитель проекта обеспечивает эффективное взаимодействие с клиентом и отвечает за сроки выполнения задач;
- разработка web-проектов любой сложности. Компания имеет большой опыт работы с сайтами разных направлений. Компания проводит глубокое погружение в проект, поиск вариантов развития на этапе разработки, которое позволяет избежать необходимости переделывать ресурс в будущем. Создает проекты, которые работают и развиваются. Все IT-продукты, разработанные компанией, имеют долгий срок жизни.

Помимо основной деятельности компания занимается интернет-маркетингом, комплексным аудитом и сопровождением сайтов, разработкой мобильных приложений и собственных интернет-проектов.

Основные подразделения компании:

– администрация,
– подразделение разработки,
– отдел продвижения.

Подразделение разработки состоит из нескольких команд, каждая из которых занимается отдельно своим проектом.

Организационная структура подразделения разработки одного из проектов представлена на рисунке 1.

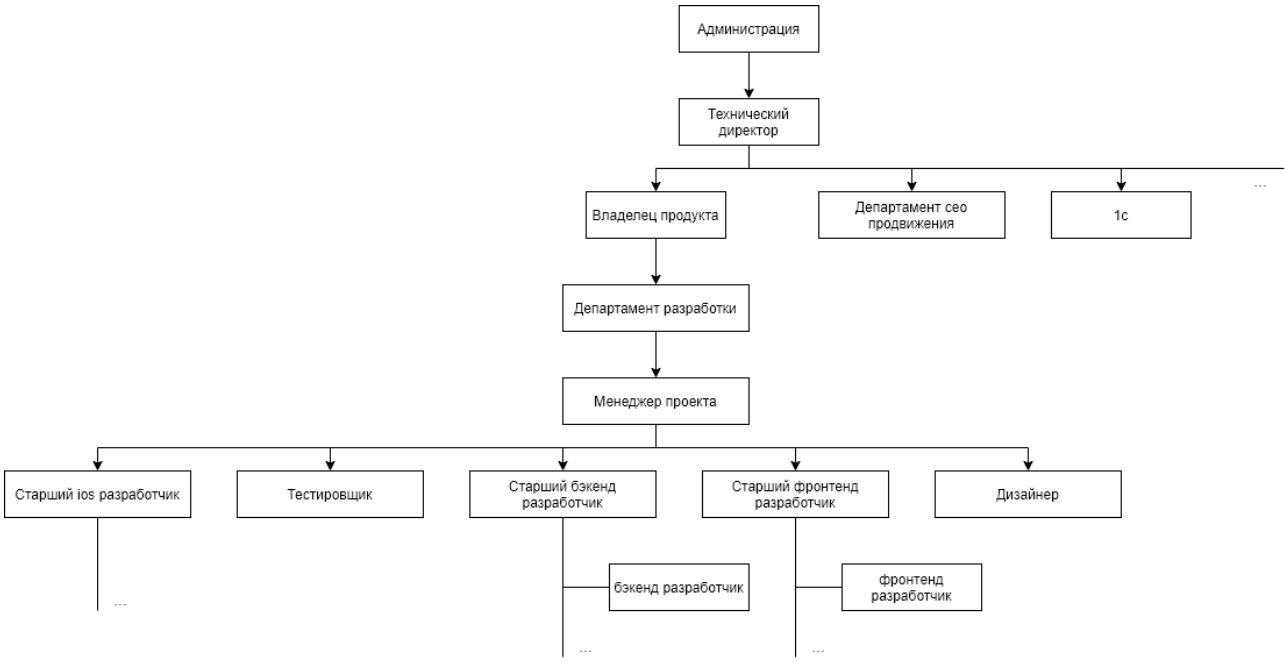

Рисунок 1 - Организационная структура подразделения разработки

Для коммуникаций между сотрудниками подразделения используются

– Telegram и discord, для общения по связи;
– Sharex и lightshot, для передачи скриншотов и записей экрана.

Для управления проектом используется онлайн-сервис Trello, где для организации задач используется доска с карточками и столбцами, которые распределяются по определенным типам:

– Site Backlog, в этом столбце хранятся задачи, которые находятся «в разработке» или на стадии идеи;
– Planning, здесь команда обсуждает будущую реализацию задачи;

- To Do, в столбце находятся карточки, которые необходимо взять в работу;
- Returned, если в результате проверки выполнения задачи на любом из этапов, проверяющий находит ошибку/ошибки в реализации задачи, то карточка возвращается в данный столбец;
- In Progress, здесь находятся задачи, над которыми в данный момент работает исполнитель;
- Hold, если по каким-то причинам исполнитель делает паузу в реализации задачи, её карточка должна быть перенесена в столбец Hold;
- Done, перенос карточки в данный столбец - сигнал для проверки задачи тестировщиком;
- Test, во время выполнения тестов задача находится в данном столбце;
- Review, перенос карточки в данный столбец - сигнал для проверки задачи на качественность выполнения задачи со стороны разработки;
- Release candidate, в данном столбце хранятся задачи, которые прошли все этапы тестирования и готовые к релизу;
- Release, в данный столбец задачи попадают незамедлительно после выхода в релиз.

Основные этапы работы над проектом:

- разработка новых бизнес-идей, наполнение бэклога проекта;
- обсуждение реализации задач;
- составление технического задания (заполнение карточки trello), для дальнейшего выполнения задачи;
- реализация задач;
- тестирование задач;

- проверка задачи (ревью) на качественность выполнение (соблюдение общего стиля проекта, масштабируемость, компетентность и т.д.);
- приемочное тестирование;
- релиз задачи;
- после-релизное тестирование.

Поступающие задачи от клиента обрабатываются проект-менеджером, который наполняет бэклог проекта. Вместе с главным разработчиком обсуждают реализацию, уточняют требования к разработке. После составление технического задания, на обсуждение определяется ответственный за реализацию задачи.

Тестировщики в своею очередь могут протестировать задачу на этапе планирования на предмет логичности составленного технического задания, составить тестовый план, обсудить детали реализации и внедрения нового функционала.

1.2 Анализ тестирования веб-приложений

1.2.1 Понятие тестирования программного обеспечения

Тестирование ПО – проверка программы для определения, что функциональность разработанного продукта соответствует ее заявленным требованиями. Тестирование обеспечивает выявление фактов расхождений с требованиями [4].

Целью тестирования является проверка соответствия программного обеспечения предъявленным требованиям, укрепление уверенности в качестве программного обеспечения и поиск очевидных ошибок, которые должны быть выявлены до момента принятия приложения в работу.

В зависимости от проекта цель тестирования может сильно изменятся, ниже были выделены основные цели тестирования:

- проверка выполнения всех установленных требований. Каждый разрабатываемой продукт имеет техническое задание (ТЗ). Оно определяет, как должна выглядеть программа после реализации поставленное задачи. В ТЗ указаны соответствующие требования, а тестировщики, должны проверить, все ли требования из ТЗ не только реализованы, но и правильно функционируют;
- предотвращение ошибок. Тестирование – это не только поиск ошибок в уже внедренном продукте. Заранее проверив техническое задание, тестировщик может указать на потенциальные проблемы, которые могут возникнуть в ходе разработки. И если заранее знать о таких проблемах, можно избежать вполне реальных ошибок в будущем;
- поиск ошибок в разрабатываемом продукте является важной частью тестирования;
- тестировщики могут показать текущее состояние продукта, выраженное в количестве ошибок, путем составления. Заинтересованные стороны (например, проект-менеджер) могут оценить текущие проблемы и решить, стоит ли выпускать продукт или нет;
- снизить риск ненадлежащего качества продукта. Чем лучше тесты, тем ниже риск пропустить критические ошибки.

Обеспечение качества разработки программного обеспечения (QA) – это процесс, который предотвращает ошибки в конечном продукте и обеспечивает, что компания выпускает действительно качественное приложение.

Контроль качества (QC) занимается проверкой уже готового продукта, а QA занимается проверкой на всех этапах разработки программного обеспечения.

Сравнение контроля качества и обеспечения качества разработки программного обеспечение представлено в таблице 1.

Таблица 1 – Сравнение QC и QA

Характеристика	Quality Control (QC)	Quality Assurance (QA)
Определение	Процесс, направленный на выполнение требований к качеству	процесс, направленный на обеспечение уверенности, что требования к качеству будут выполнены
Цель	Найти и устранить дефекты	Предотвратить возникновение дефектов
Ориентация на	Производимый продукт	Процессы создания продукта
Ответственный	Команда тестирования	Все участники, участвующие в создании продукта
Активности	Анализ документации, подготовка тестирования, тестирование, отчетность	Описание процессов, аудиты, подбор инструментов, внедрение культуры

Работа QA начиняется на начальном этапе разработки программного обеспечения, во время анализа требований. Тестировщики проверяют функциональные требования на предмет их четкости, последовательности, полноты, выполнимости.

1.2.3 Особенности тестирования веб-приложений

Тестирование веб-приложений заключается в имитации действия типичного пользователя на странице веб-сайта, проверка всех механизмом предоставляемых для пользователя, на сколько удобен и привлекателен внешний вид, совместимость с различными браузерами, соответствие заданным требованиям, тестирования производительности и безопасности приложения, поиск ошибок и оформление баг-репортов для разработчиков, которые будут исправлять найденные ошибки.

При тестировании, нужно учитывать, что веб-приложение – это клиент-серверное приложение, в котором браузер выступает в роли клиента. Бизнес-логика приложения может выполняться как на стороне клиента, так и на стороне веб-сервера, который обрабатывает запросы от пользователей приложения.

Веб-приложение состоит из следующих компонентов:

- веб-страницы, которые состоят из кода HTML, CSS и Javascript, а также могут содержать дополнительные элементы, такие как изображения или видео. Веб-страницы отображаются непосредственно на странице пользователя;
- веб-сервер, принимает HTTP-запросы от пользователей и отдает им HTTP-ответы. Ответы от сервера содержат веб-страницы для отображения или данные, используемые Javascript для обновления веб-страницы. Веб-сервер реализуются с использованием языков программирования, таких как Nodejs, PHP, Python и других;
- база данных, в которой хранится информация приложения. Она не является частью веб-сервера, но большинство разработчиков использует ее при создании веб-приложений.

Отображения страниц веб-приложения не всегда зависит от сервера. Например, SPA (одностраничные приложения) приложения, которые динамически обновляются при помощи JavaScript, загружаются в виде обычных HTML страниц.

Чаще всего взаимодействие с сервером осуществляется при помощи веб-форм, которые состоят из различных текстовых полей, полей для загрузки фотографий, документов и т.д. При тестировании веб-форм следует обращать внимание на их валидацию как на стороне сервера, так и на стороне клиента, например, ввел ли пользователь данные, корректен ли введённый email адрес и пр.

При взаимодействии с базой данных (сервером), стоит обращать внимание на скорость обработки запросов, обработки неправильных данных и последующее отображение этих данных на клиенте.

Стоит также уделять вниманию валидации верстки, семантичности (если есть таковые требования на проекте). Для валидации верстки можно использовать готовые онлайн инструменты для автоматического тестирования.

1.2.4 Виды тестирования веб-приложений

Виды тестирования веб-приложения определяются в зависимости проекта, команды тестировщика и их ресурса.

Основные виды тестирования веб-приложения приведены ниже:

- функциональное тестирование;
- тестирование удобства использования веб-приложения;
- тестирование интерфейса;
- тестирование производительности
- тестирование безопасности.

Функциональное тестирование подразумевает полную проверку механизмов веб-приложения в соответствии с заявленными требованиями в ТЗ. Сюда входит тестирование основного функционала, например, проверка работоспособности добавления в корзину, отображения страниц товаров, правильность заполнения и валидации пользовательских форм и т.д.

Тестирование удобства определяет взаимодействия пользователя с веб-приложением, чтобы можно было выявить и устранить недостатки, при использовании различных элементов на странице на предмет их логичности и понятности. При таком тестирование обращается внимание на обучаемость пользователей с навигацией, общим видом и т.д.

Тестирование пользовательского интерфейса оценивает графический интерфейс веб-приложения. В ходе проверки оценивается внешний вид интерфейса, проверка на соответствие с предъявленными макетами для разработки. На этом этапе важно оценить совместимость веб-приложения с различными браузерами и версиями браузеров.

При тестировании производительности веб-приложение проверяется, выдерживает ли оно высокие нагрузки, скорость соединения и загрузки, на сколько различные функциональные действия на странице нагружают клиентский веб-браузер, например, используя поиск, реализованный на клиентской части в списках, таблицах и т.д. Как часто отправляются запросы на сервер при взаимодействии с интерфейсом, можно ли уменьшить их

количество, с имитировав задержку при вводе данных, после которого произойдет запрос на сервер.

Тестирование безопасности заключается в проверки данных, которые приходят в веб-браузер клиента в виде переменных, передаваемых в HTML шаблон, JSON ответов от сервера, содержащая различную конфиденциальную информацию. Эта информация может привести к потере доходов, предоставление возможных лазеек при угрозе с целью взлома и вымогательства. Тестирование отправляемых данных, которые заполняются пользователями на веб-странице, с целью выявление уязвимости в системе, правильности обработки этих данных, при помощи SQL, XSS инъекция.

1.3 Автоматизированное тестирование

Использование автоматизированного тестирования подразумевает применение инструментов для повышения контроля выполнения автотестов, сравнения ожидаемых результатов работы с реальными.

Автоматизация может быть очень выгодной, если построить хороший процесс внедрения. Внедрение автоматизации – это не одноразовое мероприятие, поскольку необходимо тратить время на поддержание автоматизированных тестов в актуальном состоянии.

Иногда результаты тестирования требуются в течение нескольких часов, особенно при тестировании недавно внедренной функциональности. В этом случае эффективнее перейти на ручное тестирование и получить результаты как можно скорее.

Чтобы получить максимальную отдачу, необходимо обращать внимание как на качество самих тестов, так и на качество процессов внедрения автоматизации, а также понимать, когда автоматизировать, а когда нет.

Автоматизация имеет смысл в следующих случаях:

- основная цель – сэкономить время команды проекта;
- тесты необходимо проводить для каждой сборки приложения;

– проект длительный или сложный;
– много времени и ресурсов тратится при прохождении тест-планов;
– проводятся нагрузочные или стресс-тесты;
– текущие усилия по тестированию необходимо сократить, чтобы уложиться в определенный срок.

Ниже перечислены случаи, когда автоматизация не имеет смысла:

– для тестирования необходимы человеческий интеллект и интуиция;
– процесс тестирования ограничивается интуитивным или исследовательским тестированием;
– существующие требования к функциональности часто меняются;
– тестирование необходимо провести только один раз.

Для эффективности нужно сочетать ручное тестирование и автоматизацию. Чтобы повысить эффективность автоматизации тестирования, следует:

– чаще запускать автотесты, чтобы быстрее находить ошибки;
– группировать тесты по функциональности в наборы тестов, чтобы облегчить обновление связанных с ними тестовых примеров, когда происходят изменения в приложение;
– создавать отчеты со скриншотами и записью логов, чтобы было проще найти ошибку;
– разрабатывать автотесты так, чтобы изменения в интерфейсе не приводило к последующему изменению автотеста.

Автоматизация повышает скорость тестирования и сохраняет ресурсы команды QA, сокращает количество тестов, которые необходимо выполнять вручную, но при этом не исключают ручного тестирования.

1.3.1 Преимущества и недостатки автоматизации тестирования

С помощью автоматизированных тестов можно увеличить тестовое покрытие, благодаря многократному повторению тестовых сценариев с различными входными данными [11].

Перечислим основные преимущества автоматизации:

– отсутствие «человеческих ошибок» во время выполнения, тестовый сценарий не допускает ошибок по невнимательности;
– скорость выполнения превышает возможности человека;
– Выполнение в фоновом режиме – пока тесты выполняются, можно заниматься другими задачами;
– способность автоматизированных тестов выполнять тестовые сценарии, которые не поддаются человеческому контролю из-за их сложности, скорости или других факторов.

Ниже перечислены недостатки:

– требуется высококвалифицированный персонал, поскольку автоматизация – это проект со своими требованиями, планами и т.д.;
– затраты на внедрение инструментов автоматизации;
– существует множество инструментов автоматизации, что усложняет проблему выбора того или иного и может привести к финансовым затратам, необходимости обучения сотрудников или поиска опытных специалистов.

Не всегда можно применить автоматизацию. Рассмотрим эти случаи ниже:

– тестирование интерфейса на соответствия макетам (расположение элементов, размер шрифта и пр.);
– тестирование на юзабилити использования интерфейса;
– тестирование нового функционала «на ходу».

Исходя из вышесказанного, для внедрения автоматизации потребуется немало усилий и финансов, что повышает риски проекта, в котором планируется внедрить автоматизацию, но эффективность, скорость и надежность тестирование увеличиться многократно [11].

1.3.2 Подходы к автоматизации тестирования пользовательского интерфейса

Существует несколько подходов для автоматизации тестирования, наиболее распространёнными являются:

- использование инструментов для записи и воспроизведения (record/playback);
- написание тестовых сценариев (test scripting);
- использование готовых фреймворков;
- тестирование с использованием ключевых слов (keyword based).

Использование инструментов для записи и воспроизведения, подразумевает собой запись действий тестировщика во время ручного тестирования пользовательского интерфейса.

При написании тестовых сценариев, используется языки программирования, для которых разработаны инструменты автоматизации тестирования интерфейса. Поддержка таких тестовых сценариев требует более высокого уровня подготовки тестировщика в отличие от записи и воспроизведения.

Использование готовых фреймворков на основе инструментов для тестирования облегчают процесс написания тестовых сценариев, предоставляя удобный кодовый интерфейс с использованием ключевых слов.

Тестирование с использованием ключевых слов подразумевают собой описание последовательных действий с указанием их параметров, например, в формате JSON. Фреймворк, который отвечает за реализацию действий, прогоняет внутри себя JSON с наборами данных. Такой подход позволяет разрабатывать тесты, не имея особых навыков в программирование, но требует программиста для подготовки такого тестового окружения.

В таблице 2 приведены преимущества и недостатки подходов автоматизации тестирования [11].

Таблица 2 – список преимуществ и недостатков подходов автоматизации тестирования

Подход	Преимущества	Недостатки
Использование инструментов для записи и воспроизведения (record/playback)	Простота в использование, высокая скорость создания тестовых сценариев.	Низкая поддерживаемость, при любых изменениях необходимо дорабатывать тестовые сценарии.
Написание тестовых сценариев (test scripting)	Гибкость в написание тестовых сценариев	Сложная поддержка, необходимы хорошие навыки в программирование
Использование готовых фреймворков	Мощность и гибкость.	Необходимо изучать инструмент, иметь навыки в программирование
Тестирование с использованием ключевых слов (keyword based)	Не нужно иметь навыков в программирование, легко поддерживается, проще изменять тестовые сценарии	Необходимо наличие программиста для подготовки рабочего окружения, маленькая гибкость в написание тестовых сценариев

Наличие навыков в программирование у тестировщиков, а также наличие программистов в организации способных развернуть рабочее окружение для подразделений тестирования, позволяют использовать фреймворки автоматизации тестирования пользовательского интерфейса.

Но помимо использования готового фреймворка для автоматизации тестирования, успех разработки автоматизированных тестов зависит от выбора паттернов проектирования, которые позволяют повысить надежность и функциональность тестовых сценариев.

Наиболее популярным паттерном для разработки автоматизированных тестов для тестирования пользовательского интерфейса является PageObject.

PageObject подразумевает разделение логики тестов и логику управления конкретной страницей. Для понимая PageObject проще представить его в виде объекта с различными свойствами, отвечающими за получения элементов html-страницы, методы для выполнения типичных действий на странице, которыми можно будет воспользоваться во время написания автоматизированных тестов, например, метод открытия страницы, метод добавления товара в корзину и т.д.

Пример архитектуры PageObject представлен на рисунке 2.

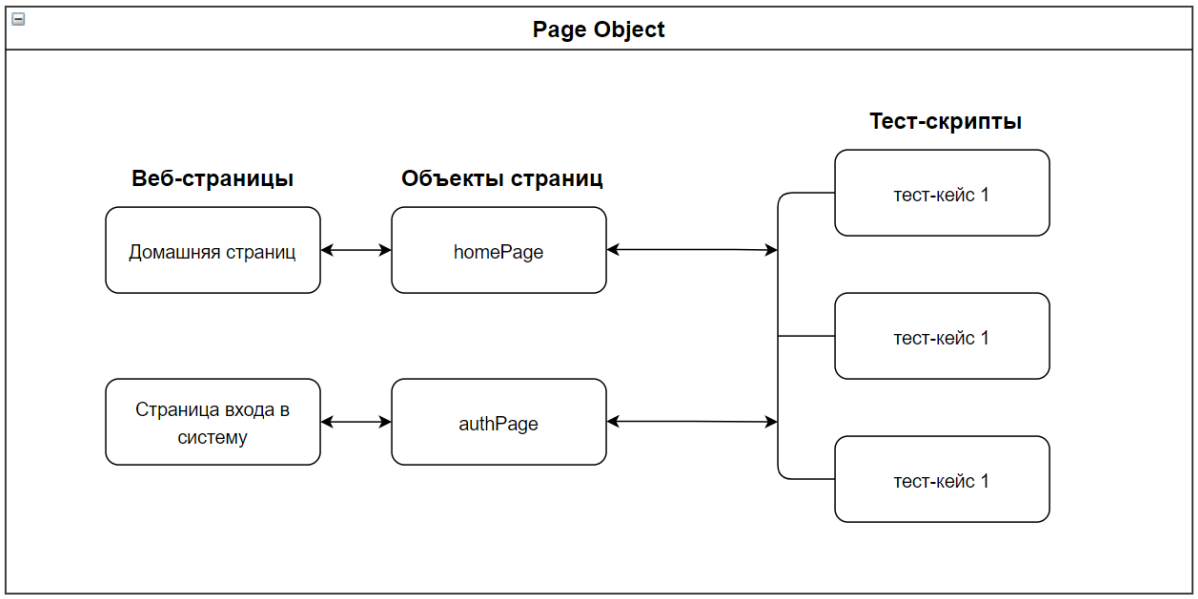

Рисунок 2 - Пример PageObject архитектуры

На рисунке 2 изображена, между взаимосвязей веб-страницы и тестовым сценарием находится прослойка в виде PageObject страницы, которая предоставляет механизмы для управления страницей.

Помимо удобного хранения информации о страницах, PageObject позволит экономить время на рефакторинг кода, в случае если изменился дизайн или программисты, отвечающие за разработку пользовательского интерфейса, изменили селекторы html-элементов на которых была завязана логика выборки этих элементов.

Помимо PageObject стоит отметить еще два менее популярных способа написания тестов:

– screenplay,

– presenter first.

Screenplay это тот же PageObject, только его суть заключается в применение принципов проектирования SOLID, что естественно очень усложняет процесс разработки автоматизированных тестов для

тестировщиков, которые не имеют хорошего опыта в разработки на одном из языков программирования.

Использование Presenter first паттерна заключается в применение MVC (Model View Controller) архитектуры при написании автоматизированных тестов с использование TDD (Test Driven Development) подхода.

1.3.3 Обзор технологий для тестирования веб-приложений

Фреймворк для автоматизации тестирования – это успешно разработанное и широко используемое решение, которое объединяет лучшие аспекты тестирования с помощью данных и ключевых слов.

У фреймворков, как и у любых других инструментов есть свои недостатки и преимущества, которые приведены в таблице 3.

Таблица 2 – Преимущества и недостатки фреймворков

Преимущества	Недостатки
Большое сообщество, в котором можно найти много полезного материала и документации. Высокий уровень масштабирования. Есть большое количество готовых решений, практик.	Необходимо изучить фреймворк. В случае если на проекте используются разные фреймворки, необходимо переучиваться. Если прекращается поддержка фреймворка, необходимо изучать и внедрять новый.

Ниже приведены преимущества JavaScript-фреймворков:

- скорость написания тестов значительно выше, чем на Java или C#;
- ниже порог входа для старта проекта;
- взаимодействия внутри команды с фронтенд-разработчиками;
- большое количество готовых решений очень разных проблем, которые возникают.

Недостатки:

- менее стабильные решения, иногда на решение проблем самого фреймворка тратиться очень много времени;

– для написания хороших тестов нужно понимание, как работает JavaScript.

Ниже рассмотрим самые популярные фреймворки для автоматизации тестирования на JavaScript.

Selenium одна из самых популярных платформ автоматизации тестирования с открытым исходным кодом, которая применяется для веб-приложений. Selenium является основой для огромного количества других инструментов тестирования.

Особенности:

– фреймворк поддерживает различные языки программирования, включая JavaScript, Java, PHP, Ruby, C # и Python;
– selenium легко адаптируется и совместим с кроссплатформенными и кросс-браузерными системами;
– поддерживается широкий спектр API и библиотек, которые могут быть изменены в соответствии с конкретными требованиями разработки;
– тестировщики могут настроить расширенные тестовые сценарии для удовлетворения любой сложности.

Кроме того, Selenium имеет одну из крупнейших сетей поддержки, развитое сообщество и встроенный инструмент воспроизведения Selenium IDE для тестирования без необходимости изучения определенного языка сценариев.

Платформа WebdriverIO созданная для автоматизации современных веб-приложений.

Плюсы:

– webdriverIO – это кастомная имплементация W3C WebDriver API;
– поддержка синхронного кода;
– удобная базовая настройка с помощью встроенного интерфейса для командной строки wdio;
– поддерживает почти все тестовые фреймворки BDD и TDD;

– поддерживает удобную библиотеку webdrivercss для сравнения CSS-стилей элементов на странице.

Минусы:

– большое различие между последними версиями;
– большой объём модулей, может усложнить этап изучения.

Nightwatch.js является автоматизированной системой тестирования для веб-приложений и веб-сайтов. Написан на Node.js, внутри использует API WebDriver.

Плюсы:

– похож на WebdriverIO и тоже является кастомной имплементацией W3C WebDriver API;
– легко добавить новую функцию;
– не нужно выбирать фреймворк для BDD и TDD;
– легок в изучение;
– готовая архитектура для PageObject паттерна.

Минусы:

– нет поддержки мобильного тестирования;
– меньше поддержки, чем у WebdriverIO.

WebdriverIO и Nightwatch.js могут быть совместимы с Selenium для разработки автоматизированных тестов.

1.4 Концептуальное моделирование автоматизации тестирования

На данный момент тестирование не автоматизировано, тестировщики вручную проверяют функционал после внесения разной степени сложности изменений в проекте.

В качестве методологии моделирования был выбран стандарт IDEF0, на основе которого можно создать иерархическую модель бизнес-процессов, разбив процесс на подпроцессы, выделив вход, управление, механизмы и выход.

В результате анализа деятельности подразделения была разработана модель тестирования IDEF0 «Как есть». Существующий подход предполагает огромное количество рутиной работы, значительных человеческих ресурсов ручного тестирования. Контекстная диаграмма тестирования «Как есть» представлена на рисунке 3.

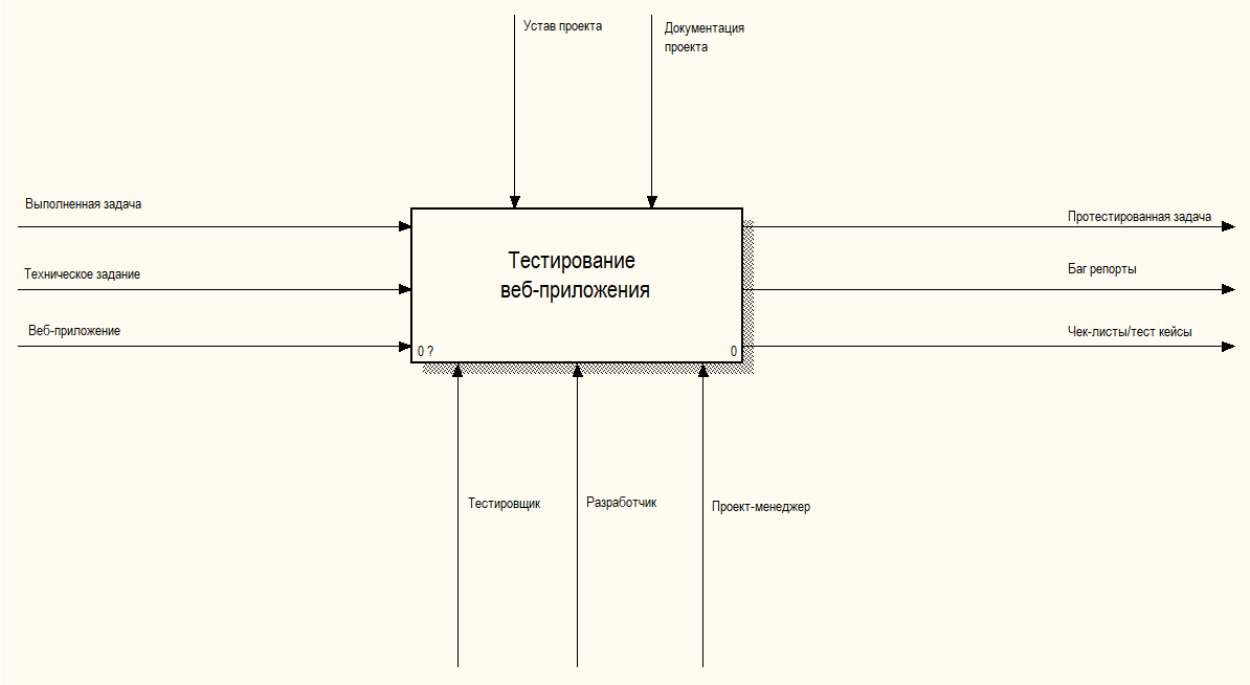

Рисунок 3 - Контекстная диаграмма тестирования «Как есть»

Входными данными являются выполненная задача, которая приходят на тестирования от разработчиков. Управление осуществляется с использованием внутреннего устава проекта, методик тестирования и документации. В процессе участвует разработчик, тестировщик и проектный менеджер, который проводит приемку протестированной задачи.

Для процесса тестирования проведем декомпозицию, которая отобразит основные этапы тестирования. Декомпозиция диаграммы «Как есть» изображена на рисунке 4.

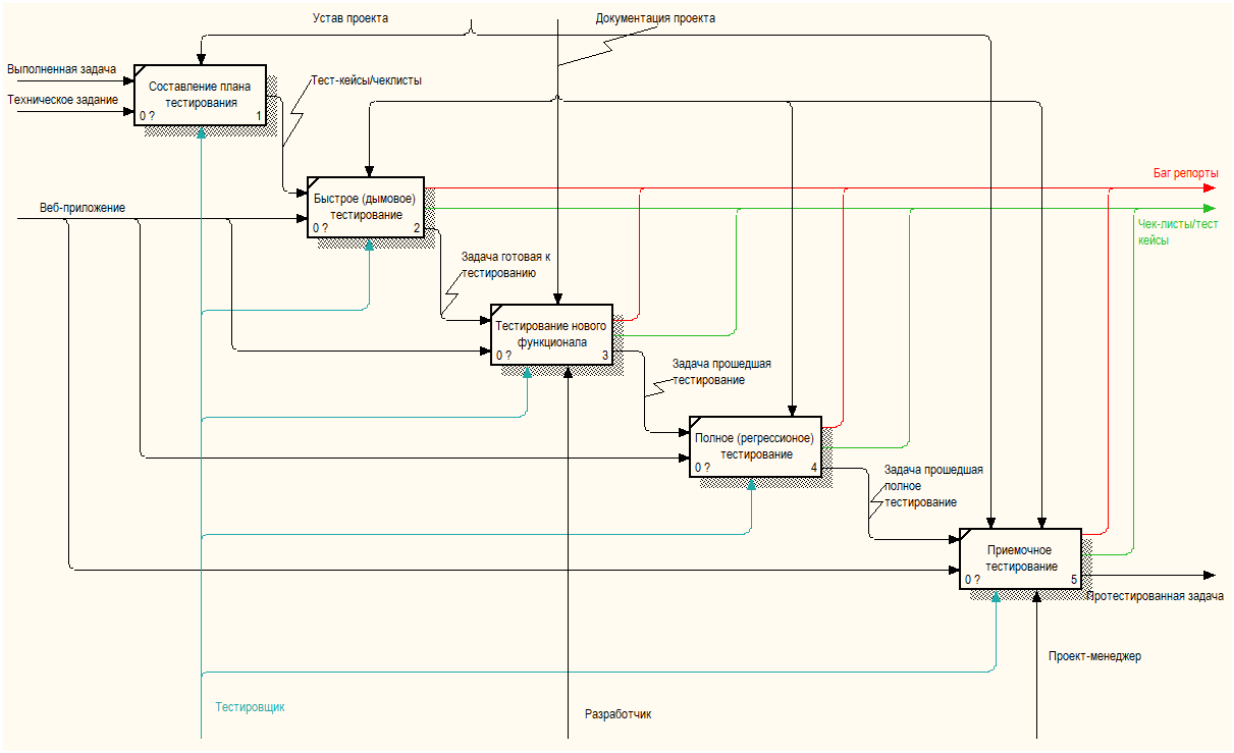

Рисунок 4 - Декомпозиция диаграммы тестирования «Как есть»

Выявленные недостатки ручного тестирования:

– тестирование занимает огромное количество времени;
– необходимо дополнительное тестирование после релиза задач;
– большое количество ручных тестов и не хватает времени на регулярное проведение полного тестирования;
– большой промежуток времени между обнаружением ошибки, ее обнаружением и исправлением.

Автоматизация решит проблему подготовки исходных данных (заглушек) для тестирования, представляющих данные пользователей, документы и т.п. Позволит использовать одни и те же тесты несколько раз.

Тесты выполняются быстрее, что позволит снизить издержки (высвобождается время, которое раньше уходило на ручное тестирование). Будет сокращено время на обнаружение ошибок и их исправления.

Модель процесса тестирования «Как должно быть» отображает новые изменения с учетом внедрения автоматизированного тестирования. Диаграмма и ее декомпозиция представлена на рисунках 5-6.

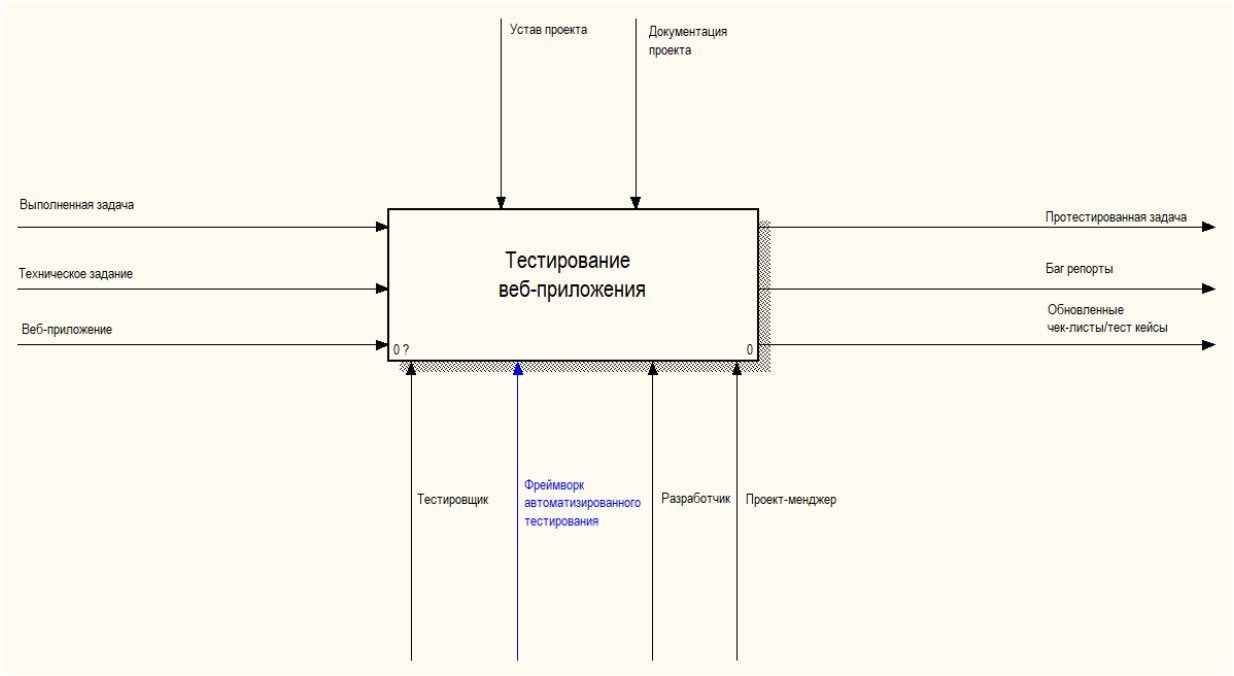

Рисунок 5 - Диаграмма бизнес-процесса тестирования «Как должно быть»

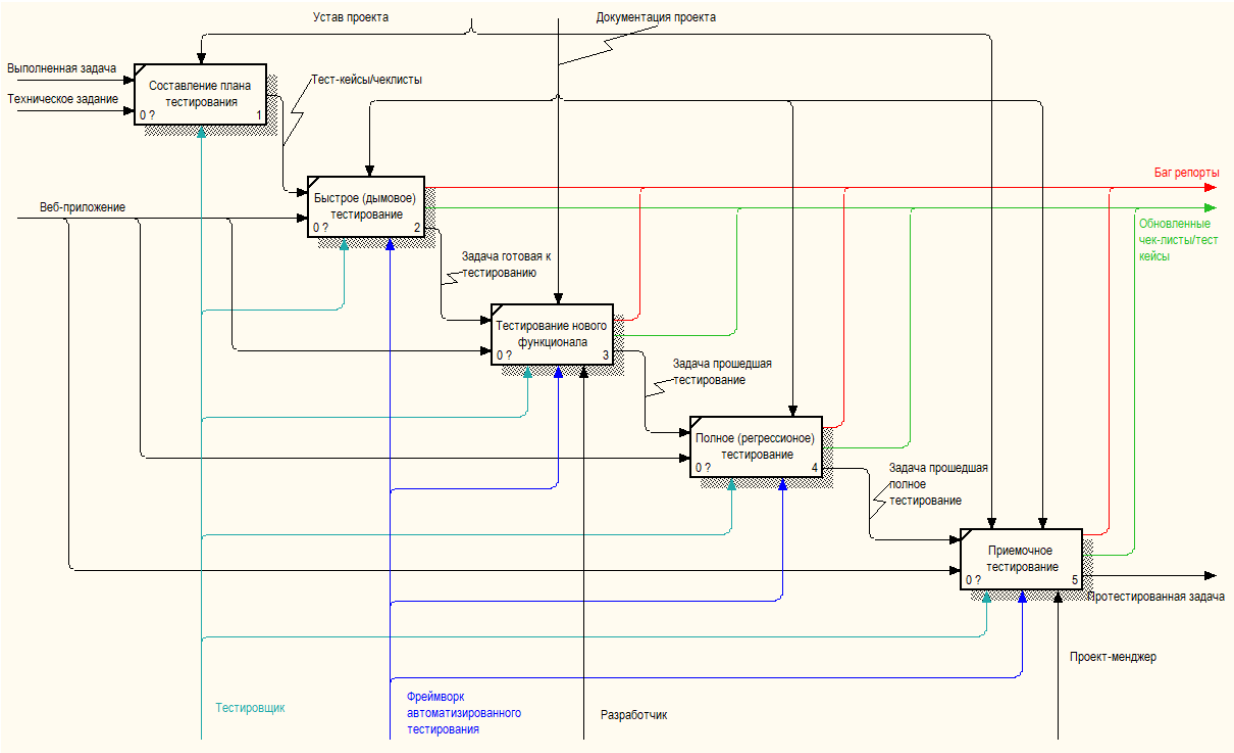

Рисунок 6 - Декомпозиция диаграммы бизнес-процесса тестирования «Как должно быть»

Теперь для проверки общих тестовых сценариев будет использоваться внедряемый фреймворк автоматизированного тестирования, это позволит значительно сократить время на тестирование. При регулярном запуске тестовых сценариев по определенному времени, команда будет знать текущую работоспособность проекта, что позволит сократить время на обнаружение ошибки и ее устранение.

1.5 Постановка задачи на разработку модуля автоматизированного тестирования

Основной задачей является необходимость создания модуля автоматизированного тестирования пользовательского интерфейса с использованием фреймворка автоматизированного тестирования.

Разработанный модуль позволит тестировщикам более удобно организовывать свою деятельность в подразделение, сократить время на регрессионное тестирование и уделить больше вниманию деталям внедряемого (нового) функционала, увеличить качество разрабатываемого проекта, снизив количество появляющихся багов, а также упростит проверку функциональности проекта в реальном времени.

Для тестового окружения, разработать панель управления, которая позволит создавать и поддерживать тестовые сценарии.

Основные требования к разрабатываемому модулю:

- возможность разработки тестовых сценариев с использованием паттерна PageObject;
- запуск одного или нескольких сценариев;
- возможность записи скриншотов или логов;
- уведомление о результатах.

Для разрабатываемой панели управления предусмотреть интуитивно понятный интерфейс для разработки тестов, оповещения пользователя о различных не удачных/удачных действиях на страницах приложения.

Итогом данной работы будет готовое рабочее окружение для написания автоматизированных тестов и разработанная панель управления для поддержки тестовых сценариев.

Выводы по первой главе

В первой главе была приведена технико-экономическая характеристика организации, определено понятие тестирования программного обеспечения, рассмотрены особенности тестирование веб-приложений, составлены преимущества и недостатки автоматизированного тестирования, проведен обзор инструментов тестирования, выполнена разработка и анализ модели «Как есть» и на ее основе разработана модель «Как должно быть», были поставлены задачи и определенны требования.

Глава 2 Проектирование модуля автоматизированного тестирования

2.1 Выбор технологии для проектирования

В методологии моделирования существует несколько подходов к построению и представлению моделей бизнес-процессов, наиболее важным из которых является функциональное моделирование.

В подходе функционального моделирования основным элементом является функция (операция), а бизнес-процесс представляется как набор функций, которые используют определенные ресурсы для преобразования входов процесса в выходы. Характерной особенностью подхода функционального моделирования является четкое разграничение между данными и функциями, которые их обрабатывают [13].

Один из самых популярных языков моделирования является UML (унифицированный язык моделирования), он применяется для проектирования и объектно-ориентированного анализа. Его можно использовать для визуализации, конструирования и документирования программ. UML предназначен для упрощения общения и взаимодействия участников проекта, сокращения времени на объяснение и усвоение информации, облегчения документирования.

Основными элементами UML являются сущности, отношения и диаграммы. Сущности являются основной абстракцией языка, отношения связывают сущности между собой, а диаграммы группируют коллекции сущностей, представляющих интерес.

Минусы:

- необходимо время на разработку;
- необходимо иметь определённые знания диаграмм и их нотации.

Плюсы:

- простота использования по сравнения с idef0;
- возможность посмотреть на задачу с разных точек зрения;

– диаграммы просты для чтения после быстрого ознакомления с их синтаксисом.

Для моделирования будет использоваться нотация языка моделирования UML.

2.2 Выбор инструмента для проектирования

Существует более десятки технологий моделирования бизнес-процессов. В качестве примера, был проведен сравнительный анализ двух наиболее популярных инструментов:

– Microsoft Visio;
– Draw.io.

Microsoft Visio является платным инструментов, представляет функционал создания графических документов, его можно использовать для создания технических проектов, моделей, диаграмм, бизнес-планов и т.д. Microsoft Visio имеет ограниченное количество опций для создания диаграмм IDEF0, но этого вполне достаточно.

Он предоставляет следующие функции:

– стандартные фигуры, обладающие широкими возможностями, представляют элементы нотации UML и поддерживают создание схем UML. Фигуры запрограммированы таким образом, чтобы их поведение соответствовало семантике UML;
– легкодоступные диалоговые окна Свойства UML, в которых можно добавлять имена, атрибуты, операции и другие свойства элементов UML;
– проводник по моделям UML предоставляет представление в виде дерева модели и средство для перехода от одного представления к другому.

Draw.io – это бесплатный онлайн-инструмент, позволяющий создавать всевозможные блок-схемы с использованием большого количества фигур и

объектов. Draw.io позволяет визуализировать всю инфраструктуру, имея предопределенные шаблоны, разделенные на категории – от графиков до дизайна программного обеспечения.

Особенности:

- нет ограничений на количество размеров;
- шаблоны присутствуют в самой разработке программного обеспечения;
- позволяет сохранить модель в любом месте.

Для описания различных блок-схем и UML диаграмм был выбран инструмент Draw.io.

2.3 Диаграмма вариантов использования

В качестве моделирования процесса тестирования была выбрана диаграмма вариантов использования (прецедентов).

В диаграмме вариантов использования присутствует 3 основных элемента:

- участник – роль исполняемая при взаимодействии с прецедентами или сущностями. Участником может быть система, класс, человек;
- прецедент – описание поведения системы с точки зрения пользователя;
- рамки системы – прямоугольник с названием в верхней части и прецедентами внутри.

Часть дублирующийся информации в модели прецедентов можно устранить указанием связей между прецедентами:

- обобщение прецедента – стрелка с не закрашенным треугольником (треугольник ставится у более общего прецедента);
- включение прецедента – пунктирная стрелка со стереотипом «include»;

– расширение прецедента – пунктирная стрелка со стереотипом «extend» (стрелка входит в расширяемый прецедент, в дополнительном разделе которого может быть указана точка расширения и, возможно в виде комментария, условие расширения).

Диаграмма вариантов использования тестировщика изображена на рисунке 7.

Рисунок 7 - Диаграмма вариантов использования тестировщика

Выделим основные функции для детального отображения взаимодействий в разрабатываемой системе во время тестирования веб-приложений на диаграмме вариантов использования, которая изображена на рисунке 8.

Рисунок 8 - Диаграмма вариантов использования автоматизированной системы

На диаграмме изображены 2 актера, «Тестировщик» имеет три прецедента:

- проверка задачи, включает тестирование нового функционала, запуск автотестов, тестирование нового функционала, уведомление о результатах;
- поддержка тест-кейсов и чек-листов тестирования в актуальном состояние;
- подготовка тестовых данных, окружения;
- поддержка автотестов.

Актер «Панель управления», которая предоставляет интерфейс для функционала:

- создания, обновления, удаления автотестов;
- отображение списка тестов и результатов тестирования;
- запуск автотестов.

Актер «Фреймворк автоматизированного тестирования», который предоставляет набор ключевых слов для написания автотестов, механизм для их запуска и эмуляции действия пользователя на веб-странице.

Далее будет рассмотрено проектирование логической модели данных для панели управления, которая будет включать хранение отчетной информации автоматизированных тестов.

2.4 Логическая и физическая модель данных

Концептуальная модель данных – это диаграмма, изображающая отношения между объектами и их свойствами в общепринятой нотации, в ней визуальном виде описываются отношений между объектами данных и их свойствами.

Диаграмма сущность/отношения (объект/связь) называют ER-диаграммой или EDR (entity-relationship diagram). Сама модель сущность-связь была предложена профессором Peter Pin-Shen Chen (Питер Чен) в 1976 году [4]. Правила написания и условные обозначения ER-диаграммы называют нотацией.

Чен предложил и это приняли следующие условные обозначения для ER-диаграмм:

- сущность или объект обозначать прямоугольником;
- отношения обозначать ромбом;
- атрибуты объектов, обозначаются овалом;
- если сущность связана с отношением, то их связь обозначается прямой линией со стрелкой. Необязательная связь обозначается пунктирной линией. Мощная связь обозначается двойной линией.

Для моделирования будет использоваться нотация, предложенная Питером Ченом.

Концептуальная ER-модель, построенная на основе методологии Питера Чена, представлена на рисунке 9.

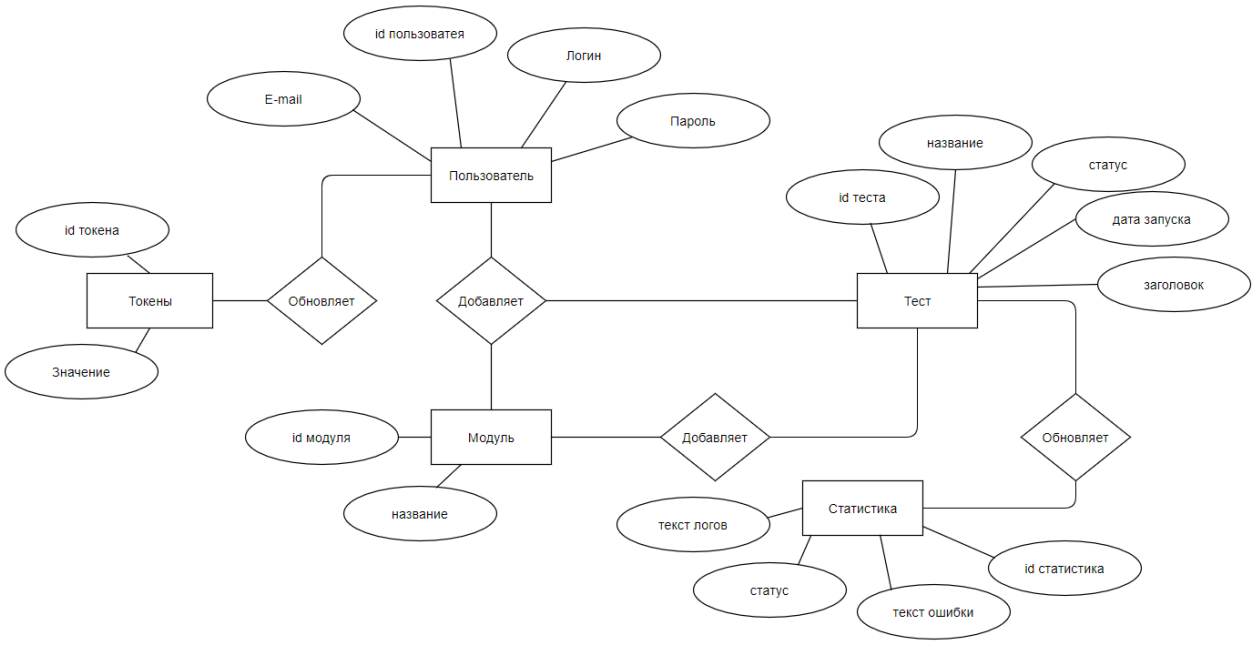

Рисунок 9 - Концептуальная ER-модель

На основе концептуальной модели, была спроектирована логическая модель, представленная на рисунке 10. Физическая модель строится на основе логической модели, представленной на рисунке 11.

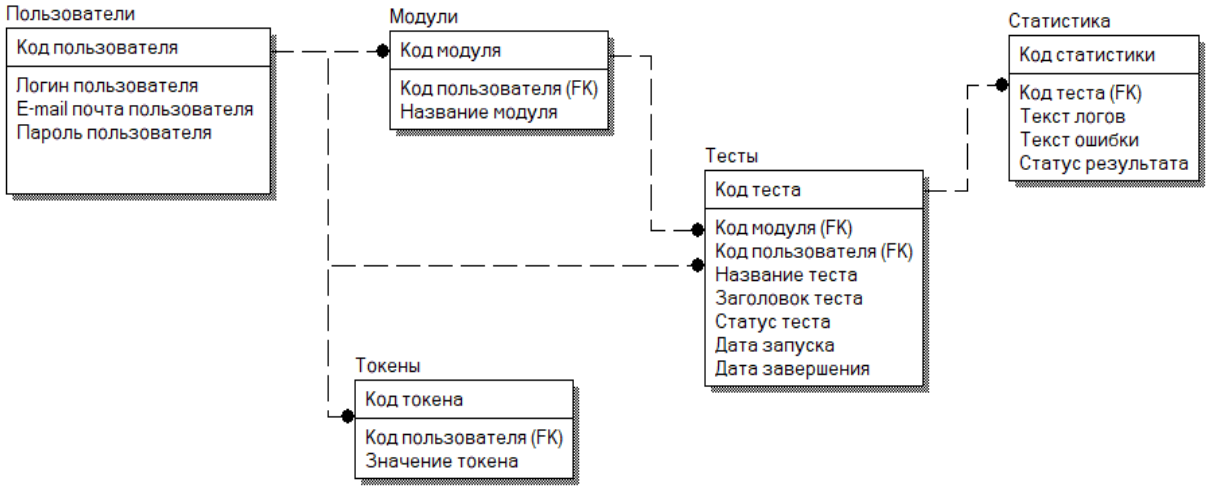

Рисунок 10 - Логическая модель данных

В логической модели присутствует 5 сущностей:

– пользователи,

– модули,
– тесты,
– статистика,
– токены.

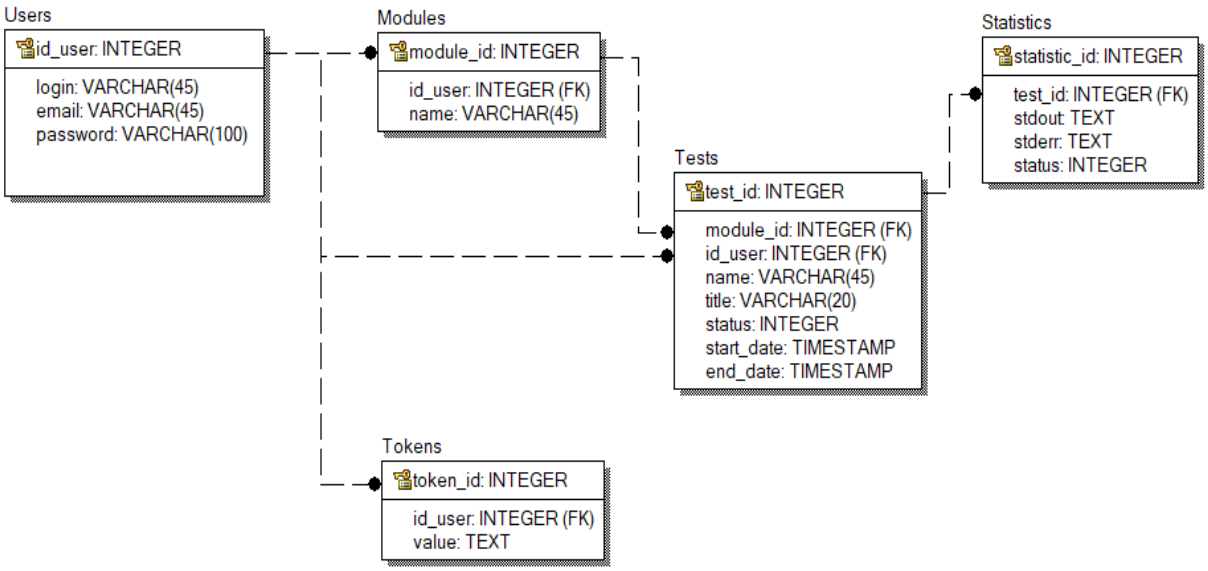

Рисунок 11 - Физическая модель данных

Сущность «Пользователи» имеет атрибут «id», который обозначает первичный ключ. «Пользователи» связаны с сущностью «Модули» связью 1:M, с сущностю 1:M «Токены».

Сущность «Модули» имеет атрибут «id», который обозначает первичный ключ. «Модули» связаны с сущностью «Тесты» связью 1:M.

Сущность «Тесты» имеет атрибут «id», который обозначает первичный ключ. «Тесты» связаны с сущностью «Статистика» связью 1:M.

Сущность «Статистика» имеет атрибут «id», который обозначает первичный ключ.

Сущность «Токены» имеет атрибут «id», который обозначает первичный ключ.

Структура таблиц приведена в таблице 4.

Таблица 4 – Структура таблиц базы данных

Название поля	Тип поля	Описание поля
Users (Пользователи)		
user_id	integer	Первичный ключ
login	varchar(45)	Логин пользователя
email	varchar(45)	Почта пользователя
password	varchar(100)	Пароль пользователя
Modules (Модули)		
module_id	integer	Первичный ключ
user_id	integer	Внешний ключ
name	varchar(45)	Название модуля
Tests (Тесты)		
test_id	integer	Первичный ключ
module_id	integer	Внешний ключ
user_id	integer	Внешний ключ
name	varchar(45)	Название теста
title	varchar(45)	Заголовок теста
status	integer	Статус теста
start_date	timestamp	Дата запуска
end_date	timestamp	Дата завершения
Statistics (Статистика)		
statistic_id	integer	Первичный ключ
test_id	integer	Внешний ключ
stdout	text(9999)	Текст выходной информации во время выполнения автотеста
stderr	text(9999)	Текст ошибки
status	integer	Статус
Tokens (Токены)		
token_id	integer	Первичный ключ
user_id	integer	Внешний ключ
value	text(9999)	Значение токена

Выводы по второй главе

Во второй главе был рассмотрен процесс проектирования системы. Были определенны технологии и инструменты для проектирования системы, были разработана диаграмма вариантов использования, разработана логическая и физическая модели данных.

Глава 3 Реализация модуля автоматизированного тестирования

3.1 Выбор технологий для разработки модуля автоматизированного тестирования

Для реализации модуля автоматизированного тестирования, были выбраны следующие технологии и инструменты:

- автоматизированное тестирование – Nightwatch.js;
- разработка серверной части панели управление – Node.js;
- хранение данных – СУБД MySQL;
- разработки клиентской части – JavaScript фреймворк React.js.

Nightwatch.js – написанный на Node.js открытый инструментарий, направленный на предоставление полноценного сквозного (end-to-end) тестирования веб-приложений, браузерных приложений и веб-сайтов с помощью Selenium и JavaScript. Он предоставляет большой набор команд и утверждений для выполнения операций с элементами DOM.

Nightwatch.js предоставляет следующие возможности из коробки:

- встроенный Test Runner: набор средств запуска тестирования из командной строки с поддержкой Grunt для выполнения автоматизированных тестов;
- стратегия тестирования: поддерживаются различные способы запуска тестов (параллельно, последовательно или по группам и тегам);
- облачные сервисы: поддержка интеграции с провайдерами облачного тестирования на Selenium, такими, как LambdaTest;
- Selenium Server: возможность автоматического управления автономным сервером Selenium с помощью встроенной отчетности JUnit XML;

- утверждения, CSS и XPath: некоторые команды и утверждения для операций DOM, а также селекторы CSS и XPath, используемые для идентификации элементов на странице;
- непрерывная интеграция: фреймворк может использоваться для интеграции тестов с системами автоматизации сборки (Jenkins, TeamCity и т.п.).

В качестве разработки клиентской части используется фреймворк React.js, разработанный компанией Facebook. Он предоставляет высокую скорость, простоту, масштабируемость и используется для разработки пользовательский интерфейсов, одностраничных и мобильных приложений.

Node.js - платформа, позволяющая писать код на стороне сервера для веб-сайтов и веб-приложений, а также программы командной строки, запускаемые вне браузера. Node.js реализует парадигму «JavaScript для всего», вместо того чтобы использовать разные языки для фронтенд и бэкенд, для разработки веб-приложений используется один язык программирования.

В качестве инструмента для хранения данных была выбрана система управления базами данных MySQL. Она не требует сложных процедур конфигурации, имеет довольно высокий уровень безопасности и является свободной СУБД, разработку и поддержку которой осуществляет корпорация Oracle.

Система Node.js и СУБД MySQL могут эксплуатироваться на разных базовых системах, включая самые популярные Linux, Microsoft Windows. Использование этой связки программного обеспечения в операционной системе Linux позволит получить в свое распоряжение полностью бесплатную базовую систему, для приобретения которой не требуется никаких предварительных расходов.

3.2 Организация рабочего окружения автоматизированных тестов

В Nightwatch.js для написания тестов используется 2 стиля:

– экспортируемый объект javascript, где ключ является названием теста, а значение, тест в виде обычной функции;

– синтаксис BDD.

В качестве примера был разработан тест проверки работоспособности поиска поисковой системы яндекс. Листинг автотестов представлен на рисунках 12-13.

```
describe('Demo', function () {
    test('first test', function (browser) {
        browser
            .url('https://yandex.ru/')
            .assert.visible('.search2__input input')
            .setValue('.search2__input input', 'reactjs')
            .click('.search2__button button')
            .assert.containsText('.entity-search__title', 'React')
            .end();
    });
});
```

Рисунок 12 - Листинг автотеста в стиле BDD

```
module.exports = {
    'first test': function (browser) {
        browser
            .url('https://yandex.ru/')
            .assert.visible('.search2__input input')
            .setValue('.search2__input input', 'reactjs')
            .click('.search2__button button')
            .assert.containsText('.entity-search__title', 'React')
            .end();
    }
}
```

Рисунок 13 - Листинг автотеста в объектном стиле JavaScript

Организационная структура файлов представлена на рисунке 14.

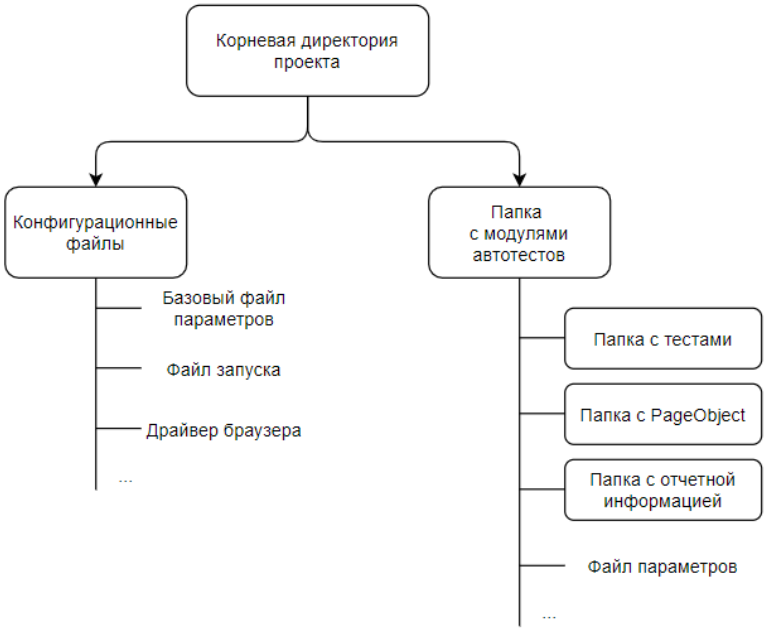

Рисунок 14 - Организационная структура автотестов

Все тесты хранятся в папке «data» (папка с модулями автотестов) в репозитории проекта и сгруппированы по разным папкам (модулям). Для запуска автотеста необходимо подготовить строку запуска с использованием npx (инструмент управления зависимостей JavaScript), для этого необходимо указать путь до конфигурационного файла и путь до файла теста. Пример командной строки изображен на рисунке 15.

```
npx nightwatch --config ./config.json --test ./tests/index.js
```

Рисунок 15 - CLI команда для запуска теста

Результаты выходной информации в консоль после удачного и неудачного запуска теста представлены на рисунках 16-17.

```
[Index] Test Suite
==================
\ Connecting to localhost on port 9515...

i Connected to localhost on port 9515 (562ms).
  Using: chrome (94.0.4606.71) on Windows platform.

√ Running first test:

√ Testing if element <.search2__input input> is visible (28ms)
√ Testing if element <.entity-search__title> contains text 'React' (930ms)

OK. 2 assertions passed. (2.605s)
```

Рисунок 16 - Результат выходной информации после удачного запуска теста

```
FAILED: 1 assertions failed, 1 errors and  1 passed (7.088s)
_____________________________________________

TEST FAILURE: 1 error during execution; 1 assertions failed, 1 passed (8.033s)

 x index
 - first test (7.088s)
   Testing if element <.entity-search__title> contains text 'React fail' in 5000ms
 - expected "contains text 'React fail'" but got: "element could not be located" (
5436ms)
       at Object.first test (D:\work\concore\agrocore\tests\nightwatch\specs\index
.js:8:21)
       at processTicksAndRejections (internal/process/task_queues.js:95:5)

  NoSuchElementError: An error occurred while running .getText() command on <.enti
ty-search__title>:
   {"sessionId":"8112e9da463ce074c70b02d50cbb8526","status":0,"value":[]}
       at processTicksAndRejections (internal/process/task_queues.js:95:5)
```

Рисунок 17 - Результат выходной информации после неудачного

Для более удобной разработки и поддержки автотестов, можно использовать паттерн PageObject. Для этого необходимо создать директорию, в которой будут содержаться файлы PageObject страниц и указать ее путь в конфигурационном фале в свойстве «page_objects_path». Nightwatch.js автоматически соберет все файлы страниц и сопоставит их название в соответствие с названием файла. Пример директории PageObject страниц изображен на рисунке 18.

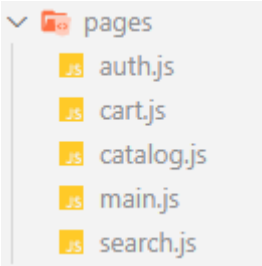

Рисунок 18 - Директория PageObject страниц

Пример PageObject страницы изображен на рисунке 19.

```
module.exports = {
    url: 'https://www.yandex.ru/',
    elements: {
        searchInput: {
            selector: '.search2__input input'
        },
        submitSearchInput: {
            selector: '.search2__button button'
        },
        entitySearchTitle: {
            selector: '.entity-search__title'
        },
    }
}
```

Рисунок 19 - Пример PageObject страницы

Пример теста с использованием PageObject страницы изображен на рисунке 20.

```
module.exports = {
    'first test': function (browser) {
        const yandex = browser.page.yandex();

        yandex.navigate()
            .assert.visible('@searchInput')
            .setValue('@searchInput', 'reactjs')
            .click('@submitSearchInput')
            .assert.containsText('@entitySearchTitle', 'React');

        browser.end();
    }
}
```

Рисунок 20 - Пример теста с использованием PageObject страницы

В данном параграфе были разобраны примеры листинга автотестов, организационная структура файлов, способ запуска автотестов. Далее рассмотрим разработку панели управления автоматизированных тестов.

3.3 Разработка панели управления автоматизированных тестов

Организационная структура файлов клиентской части изображена на рисунке 21.

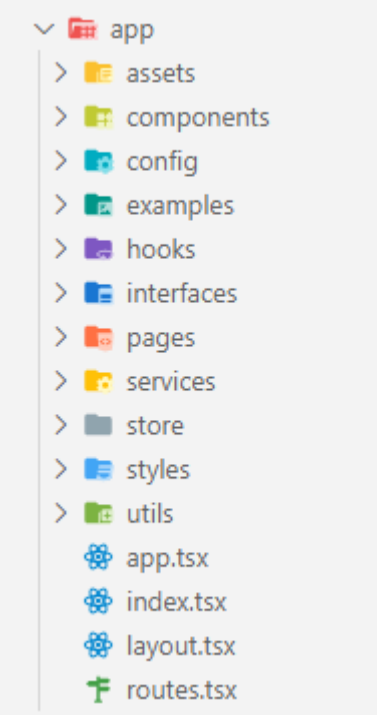

Рисунок 21 - Организационная структура файлов клиентской части

Файлы app содержит корневой главный компонент системы, layout содержит базовый шаблон, которой оборачивает страницы в компонент высшего порядка (HOC), routes объект с постраничной маршрутизацией. В файле index инициализируется приложения. Листинг кода «страницы теста» предоставлен в приложении А.

Все компоненты приложения расположены в соответствующей папке в зависимости от логики использования:

- Assets, содержит различные статические файлы, которые встраиваются в интерфейс (картинки, иконки, шрифты и т.п.);
- Components, содержит компоненты, которое используются во всем приложение;
- Config, в ней находятся различные настраиваемы объекты параметров, которые используются в приложение в зависимости от типа сборки (production, development), например, объект со списком всех api endpoints;
- Examples, хранятся готовые образцы (сниппеты) компонентов, объектов хранилища и пр.;
- Pages, содержит основные классы страниц;
- Services, содержит объекты для взаимодействия с сервером$
- Store, хранилище redux;
- Styles, содержит общие стили приложения;
- Utils, хранятся различные функции-хелперы.

Приложение состоит из меню навигации и 5 основных страниц:

- список модулей,
- страница модуля,
- список тестов,
- страница теста,
- история запусков.

Для создания модулей и тестов используются модальные окна, формы ввода данных изображены на рисунках 22-23.

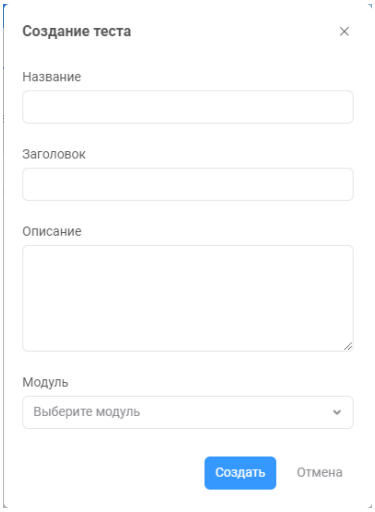

Рисунок 22 - Модальное окно создания теста

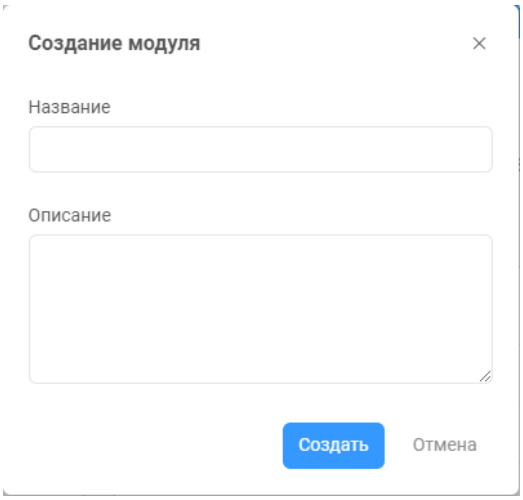

Рисунок 23 - Модальное окно создания модуля

Для вывода списков тестов используется таблица, в которой отображены название, автор, статус, дата запуска и создания, страница модуля изображена на рисунке 24.

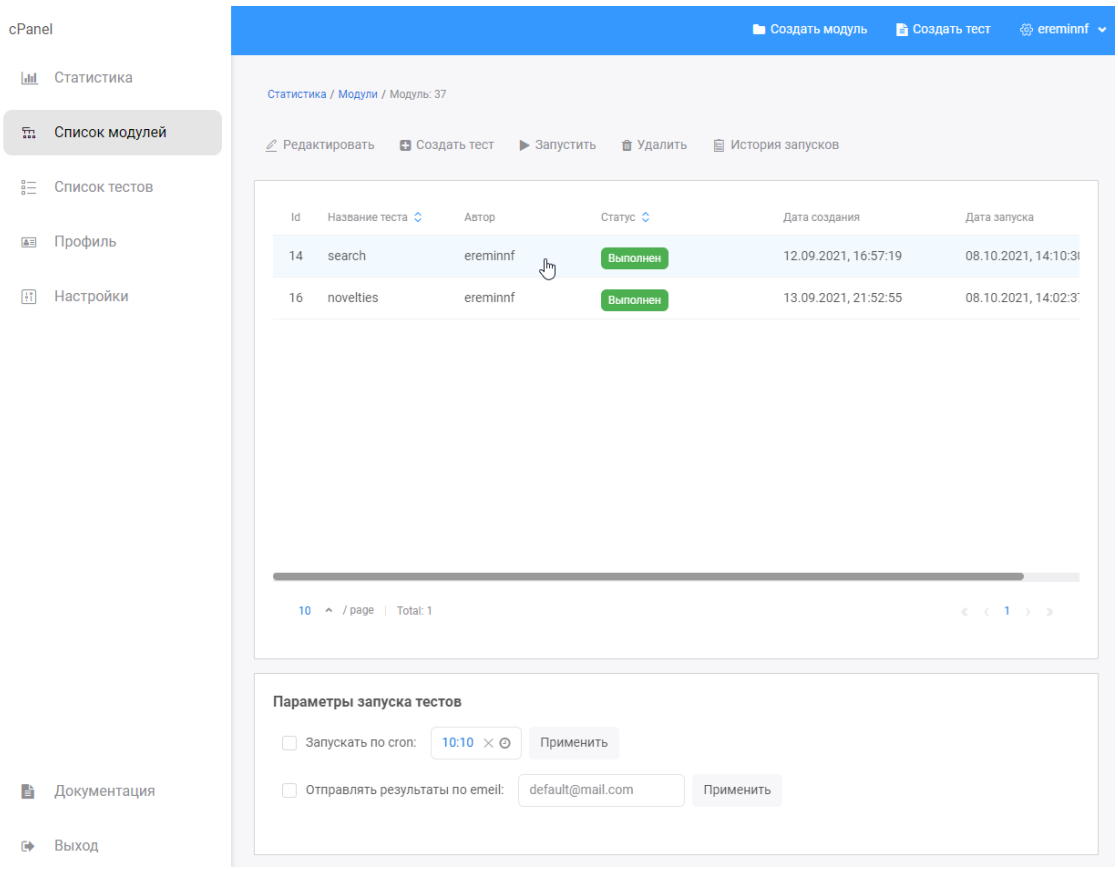

Рисунок 24 - Страница модуля

На странице теста расположены кнопки управления, форма редактирования теста и статистика запуска последнего запуска, страница теста изображена на рисунке 25.

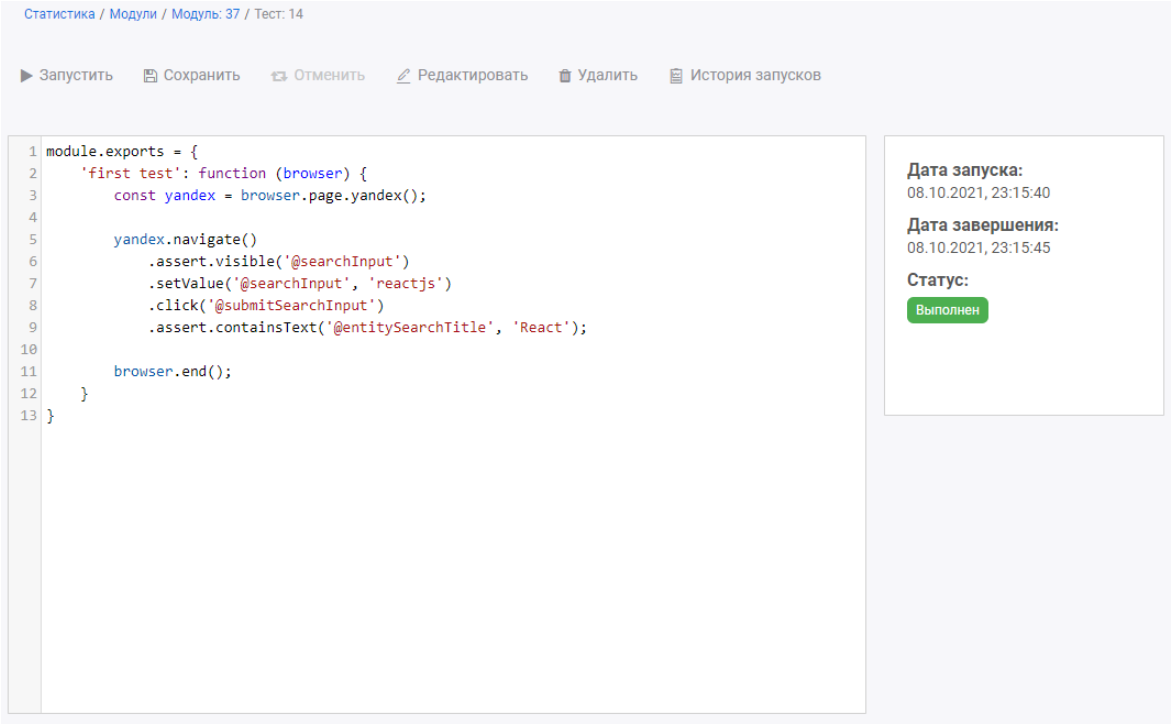

Рисунок 25 - Страница теста

Страница истории теста изображена на рисунке 26.

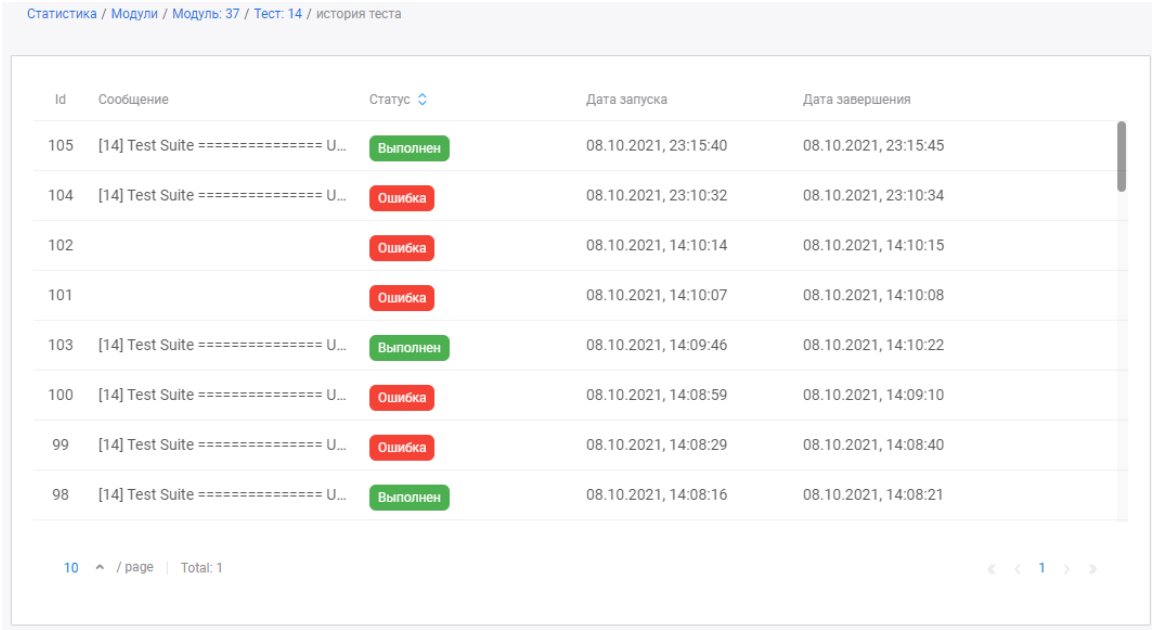

Рисунок 26 - Страница истории теста

Список основных серверных модулей приложения отражены в таблице 5.

Таблица 5 - Список серверных модулей

Название модуля	Описание
Auth	Механизмы обработки авторизации
Module	Механизм для управления модулями автотестов
Statistic	Механизм для обработки статистики автотестов
Test	Механизм для управления автотестами
User	Механизм для управления пользователями

Организационная структура файлов серверной части изображена на рисунке 27.

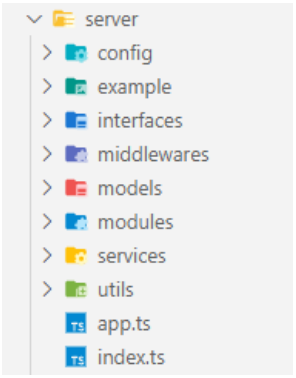

Рисунок 27 - Организационная структура файлов серверной части

В файле app содержит экземпляр серверного приложения, в нем инициализируется маршрутизация, подключаются различные утилиты и middleware обработчики. Листинг приложения предоставлен в приложение Б. В файле index запускается приложение, подключается база данных.

Основные папки:

– Middlewares, содержит промежуточные функции для обработки запросов;
– Models, объекты с запросами и доступа к базе данных;
– Modules, содержит объекты контроллеров и маршрутизаций для обработки апи запросов, запуска автотестов, авторизации и пр.;

– Services, прослойка между моделью и контроллером, принимает параметры запроса и возвращает данные БД.

Пример результат на api запрос получения данных по тесту изображен на рисунке 28.

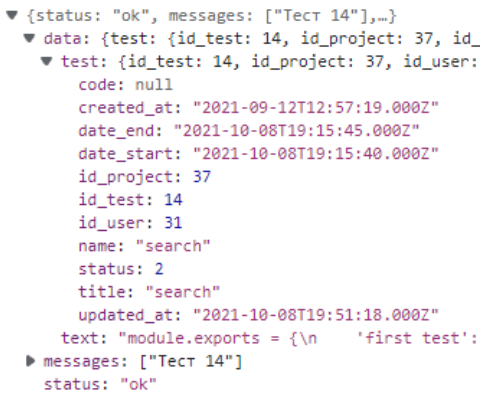

Рисунок 28 - Пример api ответа

В результате запроса приходит json объект с полями data (данные, посылаемые сервером), messages (массив сообщений, если есть) и status (статус ответа – error или ok).

3.4 Тестирование разработанного модуля

Для проверки работоспособности системы, были разработаны следующие наборы тестов:

– авторизация пользователя;
– создание нового модуля;
– создание нового теста;
– запуск теста;
– оповещение о результатах.

Ожидаемыми результатами будет оповещение пользователя в случае появление ошибок, а также, если действие было успешно выполнено.

Для проверки теста «Авторизация пользователя», были введены неверные данные для того, чтобы выяснить, как приложение реагирует на

веденные данные. Изображение с тестированием формы авторизации представлено на рисунке 29.

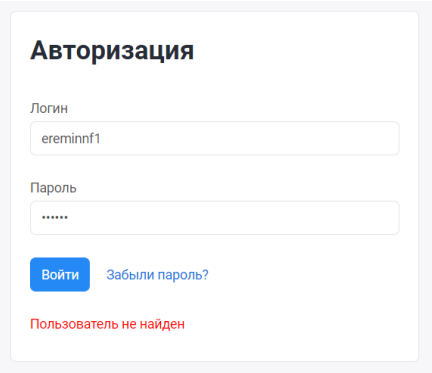

Рисунок 29 - Вывод сообщения об ошибке не удачной авторизации

Для проверки создание модуля, было вызвано модальное окно для создания модуля, при не веденном название, появляется уведомление об ошибке, которое указано на рисунке 30.

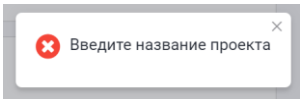

Рисунок 30 - Уведомление об ошибке при создании модуля

При вводе валидных данных, появляется уведомление об успешном создание модуля и открывается страница модуля. Уведомления изображено на рисунке 31.

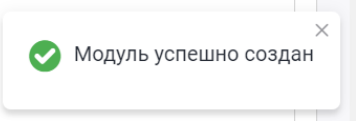

Рисунок 31 - Уведомление об успешном создание модуля

Для тестирования создание теста, было вызвано модальное окно, на панели навигации. При вводе не валидных данных, появляются подсказки

под конкретным полем. Форма с не валидными данными изображена на рисунке 32.

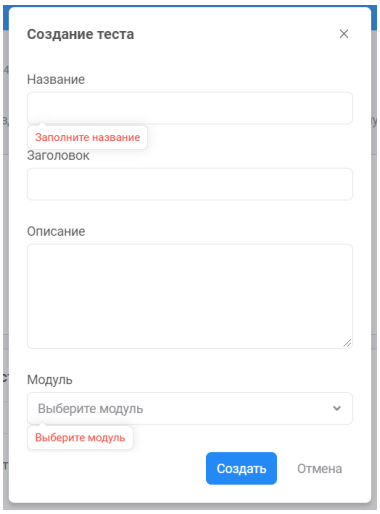

Рисунок 32 - Валидация формы создания теста

Для тестирования запуска теста, необходимо перейти на страницу теста и нажать кнопку «Запустить» на панели управления, после чего появиться уведомление о запуске теста, которое изображено на рисунке 33.

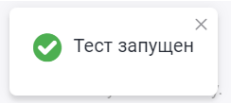

Рисунок 33 - Уведомление о запуске теста

После первого запуска теста, на странице теста появляются блок со статистикой последнего запуска, изображенный на рисунке 34.

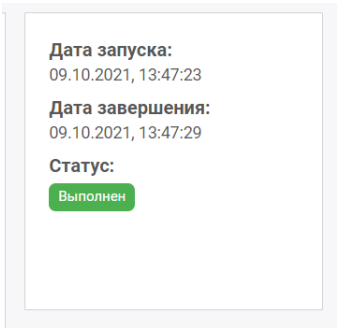

Рисунок 34 - Блок со статистикой последнего запуска

После запуска теста, будет отправлено сообщение на почту со статистикой запуска. Сообщение о результатах изображено на рисунке 35.

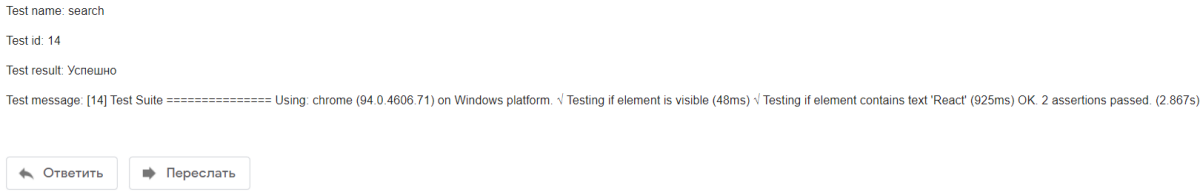

Рисунок 35 - Сообщение на почте о результатах запуска теста

В результате прохождения тестов 1 – 5 были получены результаты, которые полностью соответствует предполагаемым ожиданиям.

3.5 Оценка экономической эффективности автоматизации тестирования

3.5.1 Выбор и обоснование методики расчета экономической эффективности

Для оценки экономической эффективности, необходимо определить какие изменения будут оцениваться при внедрении автоматизации тестирования. Экономическая эффективность от внедрения модуля автоматизированного тестирования будет достигаться за счет:

– сокращения трат человеческих ресурсов ручного тестирования. Написанные тесты можно переиспользовать, их выполнение будет

значительно быстрее, а значит будет сокращено время на тестирование уже внедренного функционала на случай, если новые правки повлекли за собой сбои в работе в различных модулях веб-приложения.

– автоматизированные тесты можно будет запускать и без участия тестировщика, что позволит следить за состоянием приложения по определенному времени и в случае его неработоспособности время на обнаружение ошибки и ее исправлением будет сокращено.

Наиболее подходящей методикой расчета экономической эффективности является расчетом затрат на разработку и поддержку автоматизированного тестирования по сравнению с текущим ручным тестированием.

Для расчета прямого эффекта от внедрения разработанного модуля необходимо рассмотреть показатели трудовых и стоимостных затрат.

К трудовым показателям относятся:

– абсолютное снижение трудовых затрат, рассчитываемое по формуле:

$$\Delta T = T_0 - T_1, \quad (1)$$

где:

T_0 – время, затрачиваемое на выполнение тестирования в базовом варианте;

T_1 – время, затрачиваемое на выполнение тестирования в разрабатываемом варианте.

– коэффициент относительного снижения трудовых затрат $К_Т$ (в процентах), для расчета которого используется следующая формула:

$$K_T = (\Delta T \ / T_0) \cdot 100\%; \quad (2)$$

- индекс снижения трудовых затрат, рассчитываемый следующим образом:

$$Y_T = (T_0/T_1) \cdot 100\%; \quad (3)$$

К стоимостным показателям относятся следующие:

- абсолютное снижение стоимостных затрат:

$$\Delta C = C_0 - C_1, \quad (4)$$

где:

C_0 – стоимостные затраты на обработку информации по базовому варианту;

C_1 – стоимостные затраты на обработку информации по предлагаемому варианту.

- коэффициент относительного снижения стоимостных затрат $К_С$ (в процентах), определяемый по следующей формуле:

$$K_C = (\Delta C / C_0) \cdot 100\%, \quad (5)$$

- индекс снижения стоимостных затрат, рассчитываемый по формуле

$$Y_C = C_0 / C_1, \quad (6)$$

Коэффициенты K_C и Y_C характеризуют рост производительности труда за счет внедрения более экономичного варианта проектного решения.

Помимо рассмотренных показателей целесообразно также рассчитать срок окупаемости затрат на внедрение проекта (T_{OK}):

$$T_{OK} = K_П / \Delta C, \quad (7)$$

где:

$K_{\text{П}}$ – капитальные затраты на создание проекта.

$$K_{\text{П}} = \text{С}_{\text{ЗАРП}} + \text{С}_{\text{ОБОР}} + \text{С}_{\text{НАКЛ}}, \quad (8)$$

где:

$\text{С}_{\text{ЗАРП}}$ – заработная плата программиста;

$\text{С}_{\text{ОБОР}}$ – затраты на обеспечение необходимым оборудованием;

$\text{С}_{\text{НАКЛ}}$ – накладные расходы;

В данном параграфе для обоснования экономической эффективности был выбран расчет прямой эффективности. Для определения эффективности будет проведено сравнение времени, затрат на ручное тестирование и с использованием средств автоматизации тестирования.

3.5.2 Расчет показателей экономической эффективности

Рабочий день тестировщика состоит из восьмичасового рабочего дня, часть времени тестировщик тратит на общую планерку, что составляет около одного рабочего часа, подготовку тестовых данных по запросу менеджеров и разработчиков примерно один час, на проверку работоспособности приложений по запросу менеджеров или в случае появления ошибок в логах системы тестировщик ежедневно тратит один рабочий час.

Сколько тратиться времени на тестирование нового функционала точно сказать сложно, все зависит от поставленных разработчикам задач, их выполнение может длиться от получаса до нескольких недель.

По экспертной оценке, сотрудников организации, в среднем тестировщик на проверку нового функционала тратит от одного до четырех рабочих часов в день, в это время входит проверка не только нового функционала, но и проверка уже рабочего функционала на случай, если новые правки повлекли за собой появление ошибок в других компонентах приложения.

Средняя заработная плата тестировщика составляет 40 000 руб./месяц, в среднем месяц состоит из 22 рабочих дней, 176 часов. Из этого получается, что стоимость 1 рабочего часа тестировщика составляет около 230 руб./час. При среднем времени тестирование функционала 4 рабочих часа (240 минут) в день, траты на ручное тестирование составляет 920 руб./день.

При автоматизации тестирования среднее время выполнения тестов будет составлять около 10 минут.

На основание описанной ранее методики выполним расчет показателей экономической эффективности. Расчет показателей экономической эффективности приведен в таблице 6.

Таблица 6 – Расчет показателей экономической эффективности проекта

	Затраты		Абсолютное изменение затрат	Коэффициент изменения затрат	Индекс изменения затрат
	Базовый вариант	Проектный вариант			
Трудоемкость	$T_{0,}$ час	$T_{1,}$ час	(1)	(2)	(3)
	4	0.5	3.5	87.5	114
Стоимость	$C_{0,}$ руб.	$C_{1,}$ руб.	(4)	(5)	(6)
	920	115	805	87.5	8

Рассчитаем затраты на разработку, так как работы по разработке и внедрению могут проводиться на оборудовании, ранее установленном на рабочих местах, то $C_{ОБОР}$ равны нулю, накладные расходы $C_{НАКЛ}$ составят 3% от суммы всех.

Расчет затрат на выплату заработной платы и накладные расходы следует вычислить по соотношениям и того, что время разработки и составляет 1 месяц и будет заниматься один программист, с заработной платой 60 000 руб./месяц.

$$K_{П} = (60\,000 + 0) \cdot 1.03 = 61\,800 \text{ руб.}$$

Рассчитаем срок окупаемости, учитывая подразделение тестирования на одном проекте состоит из 1 тестировщика:

$$T_{OK} = 61\,800 \,/\, 802 \;=\; 77 \text{ дней}$$

С учетом развития проекта в будущем нужно будет учитывать, что выполнения тестов может занимать более чем 10 минут.

Выводы по третьей главе

В третьей главе был произведен выбор технологий для разработки модули автоматизированного тестирования, было описана организация рабочего окружения автоматизированных тестов, предоставлен функционал и результаты тестирования разработанного модуля, произведены расчеты экономической эффективности автоматизации тестирования.

Заключение

Задачей выпускной квалификационной работы было изучение тестирования веб-приложений, подготовка окружения для написания автоматизированных тестов и разработка панели управления для создания, анализа и поддержки автоматизированных тестов.

Для решения поставленных задач был проведен анализ предметной области автоматизированного тестирования, произведено концептуальное моделирование предметной области, спроектированы схемы тестирования IDEF0 «Как есть» и «Как должно быть», определены и поставлены задачи.

В ходе проектирования были выбраны технологии и инструменты для проектирования модуля автоматизированного тестирования, разработаны UML диаграммы вариантов использования тестирования, разработана логическая и физическая модели данных.

При разработке модуля автоматизированного тестирования были определенны используемые технологии, подготовлено рабочее окружения для написания автоматических тестов, разработано панель управления для запуска, создания и поддержки автоматических тестов. Произведено тестирование разработанной панели управления и предоставлены результаты.

Для внедрения полученных результатов в организацию необходимо решить ряд проблем, связанных с интеграцией с сервисами управления системой контроля версий, подготовки документации и обучению сотрудников.

Разработанный модуль имеет большой потенциал для дальнейшего развития. Могут быть продуманы алгоритмы решения задачи, улучшена система работы с данными, добавлены новые инструменты для тестирования и отчетности. Разработанная панель управление может быть расширена новыми видами статистики, параметрами запуска и интеграциями с различными сервисами управления проектами.

Список используемой литературы

1. Глухова, Т.В. Способы и средства моделирования бизнес-процессов предприятия [Электронный ресурс] // Российский журнал менеджмента 2018. URL: https://cyberleninka.ru/article/n/sposoby-i-sredstva-modelirovaniya-biznes-protsessov-predpriyatiya (дата обращения: 24.07.2021).
2. Использование диаграммы вариантов использования UML при проектировании программного обеспечения. [Электронный ресурс] // URL. https://habr.com/ru/post/566218/ (дата обращения: 22.09.2021).
3. Кара-Ушанов В.Ю. Модель «Сущность связь» – Екатеринбург, 2017. – 63 с.
4. Котляров В.П. Основы тестирования программного обеспечения. Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. – 334 с.
5. Куликов, С.С. Тестирование программного обеспечения – Минск «Четыре четверти», 2017. – 310 с.
6. Майерс Гленфорд, Баджетт Том. Искусство тестирования программ – Вильямс, 2020. – 272 с.
7. Марк Тиленс Томас. React в действии – Питер, 2019. – 368 с.
8. Методология моделирования бизнес-процессов. [Электронный ресурс] // URL. https://piter-soft.ru/knowledge/glossary/process/metodologiya-modelirovaniya-biznes-protsessov.htm (дата обращения: 30.06.2021).
9. Официальный сайт ReactJS с документацией на русском языке. [Электронный ресурс] // URL. https://ru.reactjs.org/ (дата обращения: 19.06.2021).
10. Серебрякова, Т. А. Тюриков, И. А. Сравнительный анализ case – средств [Электронный ресурс] // Colloquium-journal 2019. URL: https://cyberleninka.ru/article/n/sravnitelnyi-analiz-case-sredstv (дата обращения: 24.07.2021).

11. Терехов, А.Н. Платонова, М.В. Моделирование бизнес-процессов в цифровую эпоху [Электронный ресурс] // Российский журнал менеджмента 2019. URL: https://cyberleninka.ru/article/n/modelirovanie-biznes-protsessov-v-tsifrovuyu-epohu (дата обращения: 24.07.2021).

12. Цуканова О. А. Методология и инструментарий моделирования бизнес-процессов – Универститет ИТМО, 2015. – 100 с.

13. IDEF0. Знакомство с нотацией и пример использования. [Электронный ресурс] // URL. https://itnan.ru/post.php?c=1&p=322832 (дата обращения: 24.07.2021).

14. UI-автоматизация или почему стоит посмотреть в сторону JavaScript. [Электронный ресурс] // URL. https://dou.ua/lenta/articles/automation-js-frameworks/ (дата обращения: 24.09.2021).

15. Difference Between QA and QC. [Электронный ресурс] // URL. https://www.softwaretestinghelp.com/quality-assurance-vs-quality-control/ (дата обращения: 29.09.2021).

16. Martin Fowler. PageObject. [Электронный ресурс] // URL. https://martinfowler.com/bliki/PageObject.html (дата обращения: 22.09.2021).

17. Nightwatch Documentation. [Электронный ресурс] // URL. https://nightwatchjs.org/ (дата обращения: 04.10.2021).

18. Selenium Tests with Mocha and Chai in JavaScript. [Электронный ресурс] // URL. https://nehalist.io/selenium-tests-with-mocha-and-chai-in-javascript/ (дата обращения: 22.09.2021).

19. Web Application Testing. [Электронный ресурс] // URL. https://www.softwaretestinghelp.com/web-application-testing/ (дата обращения: 30.09.2021).

20. What Is Automation Testing. [Электронный ресурс] // URL. https://www.softwaretestinghelp.com/automation-testing-tutorial-1/ (дата обращения: 29.09.2021).

Приложение А

Листинг программного кода клиентского компонента «Страница теста»

```
import React from 'react';
import { useParams, useLocation, NavLink } from 'react-router-dom';
import { Button, Icon, Loader, Nav, Tag } from 'rsuite';
import styled from 'styled-components';
import { UnControlled as CodeMirror } from 'react-codemirror2';
import 'codemirror/lib/codemirror.css'
import 'codemirror/addon/fold/foldcode'
import 'codemirror/theme/nord.css'
import 'codemirror/mode/javascript/javascript.js'
import _ from 'lodash';
import { NotificationSuccess, NotificationError } from '../../components/notification/notification';
import testService from '../../services/test.service';
import { useModal } from '../../hooks/useModal';
import { TestDeleteConfirmModal } from './components/deleteConfirmModal';
import { SettingContent } from '../../components/pages/settingContent';
import { ControlContent } from '../../components/pages/controlContent';
import { Editor } from '../../components/editor/editor';
import { ITestModelState } from '../../../server/models/test.model';
import { TestUpdateModal } from './components/updateModal';

export type ITestPagesProps = {

}

interface RouteParams {
 id_test: string;
 id: string;
}

const Grid = styled.div`
 display: grid;
 grid-template-columns: 1fr 255px;
 grid-template-rows: auto 255px 255px auto;
 grid-column-gap: 16px;
 grid-row-gap: 16px;
 position: relative;

 .content {
 margin: 0;
 }

 .rs-loader-backdrop {
 z-index: 9;
 }
`;

const TestContent = styled.div.attrs({
 className: "content"
})`
 grid-row-start: 2;
 grid-row-end: 4;
```

```
 background: transparent;
 padding: 0;
 border: none;
`;

const ProgressContent = styled.div.attrs({
 className: "content"
})`
 border: 1px solid lightgray;
 border-radius: 1px;
 grid-row-start: 3;
 grid-row-end: 4;
`;

const StatisticContent = styled.div.attrs({
 className: "content"
})`
 border: 1px solid lightgray;
 border-radius: 1px;
 grid-row-start: 2;
 grid-row-end: 3;
`;

const TestPages: React.FC<ITestPagesProps> = ({ }) => {
 const { id_test, id } = useParams<RouteParams>();
 const [loading, setloading] = React.useState(true);
 const refEditor = React.useRef<any>(null);
 const [state, setState] = React.useState<'none' | 'change' | 'save' | 'play'>('none');
 const [code, setCode] = React.useState('');
 const [test, setTest] = React.useState<ITestModelState['dto']>(null);
 const confirmDeleteModal = useModal();
 const updateModal = useModal();

 React.useEffect(() => {
 testService
 .getViaFile({
 id_project: parseInt(id),
 id_test: parseInt(id_test)
 })
 .then(res => {
 if (res.data?.text) {
 refEditor.current.editor.setValue(res.data.text);
 setCode(res.data.text);
 setTest(res.data.test);
 }
 })
 .catch(err => {
 NotificationError(err.message)
 })
 .finally(() => {
 setTimeout(() => {
 setloading(false);
 }, 300);
 })
```

```
}, []);

function onPlayHandler() {
setloading(false);
const prevState = state;

testService
.start({
id_project: parseInt(id),
id_test: parseInt(id_test),
})
.then(res => {
if (res.status === 'ok') {
setState('play');
return NotificationSuccess('Тест запущен');
}

NotificationError(res.messages[0] || '');
})
.catch(err => {
NotificationError(err.message)
})
.finally(() => {
setTimeout(() => {
setloading(false);
setState(prevState);
}, 300);
})
}

function onSaveHandler() {
setloading(false);
testService
.updateViaFile({
id_project: parseInt(id),
id_test: parseInt(id_test),
text: refEditor.current.editor.getValue()
})
.then(res => {
if (res.status === 'ok') {
return NotificationSuccess('Тест обновлен');
}

NotificationError(res.messages[0] || '');
})
.catch(err => {
NotificationError(err.message)
})
.finally(() => {
setTimeout(() => {
setloading(false);
setState('save');
}, 300);
})
```

```
        }

        function onCancelHandler() {
        setState('change');

        setloading(false);
        testService
        .updateViaFile({
        id_project: parseInt(id),
        id_test: parseInt(id_test),
        text: code
        })
        .then(res => {
        if (res.status === 'ok') {
        refEditor.current.editor.setValue(code);
        return NotificationSuccess('Изменения отменены');
        }

        NotificationError(res.messages[0] || '');
        })
        .catch(err => {
        NotificationError(err.message)
        })
        .finally(() => {
        setTimeout(() => {
        setloading(false);
        setState('save');
        }, 300);
        })
        }

        function onDeleteHandler() {
        // setState('delete')
        confirmDeleteModal.toggle(true);
        }

        function onChangeHandler(editor: any, data: any, value: any) {
        // console.log({ value }, _.escape(value), _.unescape(_.escape(value)));
        setState('change')
        }

        const onUpdateHandler = () => {
        updateModal.toggle(true);
        }

        return (
        <Grid>
        <ControlContent>
        <Nav.Item icon={<Icon icon="play" />} onSelect={onPlayHandler} disabled={state ===
'play'}>Запустить</Nav.Item>
        <Nav.Item icon={<Icon icon="save" />} onSelect={onSaveHandler} disabled={state !==
'change'}>Сохранить</Nav.Item>
        <Nav.Item icon={<Icon icon="retweet" />} onSelect={onCancelHandler} disabled={state !==
'save'}>Отменить</Nav.Item>
```

```
            <Nav.Item icon={<Icon icon="edit" />} onSelect={onUpdateHandler} disabled={state ===
'play'}>Редактировать</Nav.Item>
            <Nav.Item icon={<Icon icon="trash" />} onSelect={onDeleteHandler}>Удалить</Nav.Item>
            <Nav.Item icon={<Icon icon="building2" />} componentClass={NavLink}
to={`/projects/${id}/${id_test}/history`}>История запусков</Nav.Item>
            </ControlContent>
            <TestContent>
            <Editor onChange={onChangeHandler} ref={refEditor} />
            </TestContent>
            {
            test && test?.status === 1 ?
            <ProgressContent>
            Прогресс в real-time
            </ProgressContent> : ''
            }
            {
            test && test?.status !== 0 ?
            <StatisticContent>
            <div>
            <h6>Дата запуска:</h6>
            <span>{test?.date_end ? new Date(test.date_start).toLocaleString() : ''}</span>
            </div>
            {
            test.status !== 1 ?
            <div>
            <h6>Дата завершения:</h6>
            <span>{test?.date_end ? new Date(test.date_end).toLocaleString() : ''}</span>
            </div> : ''
            }
            <div>
            <h6>Статус:</h6>
            {
            {
            1: <Tag color="orange">Запущен</Tag>,
            2: <Tag color="green">Выполнен</Tag>,
            3: <Tag color="red">Ошибка</Tag>,
            }[test?.status as number]
            }
            </div>
            </StatisticContent> : ''
            }

            {loading ? <Loader backdrop content="loading..." vertical /> : ''}
            {
            test ?
            <>
            <TestUpdateModal {...updateModal} test={test} />
            <TestDeleteConfirmModal {...confirmDeleteModal} test={test} />
            </> : ''
            }
            </Grid>
            );
           }
           export { TestPages };
```

Приложение В

Листинг программного кода серверного приложения

```
import express, { NextFunction, Request, Response } from 'express'
import path from 'path'
import middlewares from './utils/middlewares';
import nunjucks from 'nunjucks'
import routers from './modules';
import ErrorService from './services/error.service';

const app = express();
const njk = nunjucks.configure(path.join(__dirname, '..', 'templates'), {
 autoescape: true,
 express: app,
 noCache: true,
 watch: true
});

app.set('view engine', 'njk');

app.use('/assets', express.static(path.resolve('./src/assets')));
app.use('/bundles', express.static(path.resolve('./dist/bundles')));
app.use('/static', express.static(path.resolve('./src/static')));

middlewares.forEach((middleware) => app.use(middleware));

app.use(routers);

app.use(function (error: Error, req: Request, res: Response, next: NextFunction) {
 console.log(error);

 if (error instanceof ErrorService) {
 if (error.code !== undefined) {
 res.status(error.code).json(error.get());
 } else {
 res.json(error.get());
 }
 } else {
 res.status(500).json(new ErrorService({
 messages: [error.message],
 stack: error.stack
 }).get());
 }
});

app.use(function (req: Request, res: Response) {
 res.status(404).send('404');
});

export default app
```