

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет
имени первого Президента России Б.Н. Ельцина»

Институт радиоэлектроники и информационных технологий – РТФ
Школа бакалавриата

ОТЧЕТ

По проекту
«Разработка серверного приложения для автоматизации сборки и выпуска
релизов»
по дисциплине «Проектный практикум»

Заказчик: СКБ Лаб

Куратор: Пиксаева А. Ю.

Студенты:

Савельева А. А.

Бобина А. А.

Лобанов А. А.

Жарков С. Н.

Насибов Ф.

Екатеринбург, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 Команда разработки	5
2 Определение проблемы	6
3 Анализ аналогов	8
3.1 Jenkins	8
3.2 GitHub Actions	8
4 Подходы к решению проблемы	10
5 Техническое задание	11
5.1 Функциональные требования	11
5.2 Нефункциональные требования	13
6 Стек и архитектура разработки	14
6.1 Frontend-разработка	14
6.2 Backend-разработка	14
7 План работы на семестр	16
8 Ход работы	18
8.1 Отчёт о работе тимлида-аналитика	18
8.2 Отчёт о работе дизайнера	18
8.3 Отчёт о работе frontend-разработчика	18
8.4 Отчёт о работе backend-разработчиков	18
8.4.1 Отчёт Лобанова Антона	18
8.4.2 Отчёт Насибова Фариза	18
9 Результаты	19
ЗАКЛЮЧЕНИЕ	20
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	21
ПРИЛОЖЕНИЕ А	24
ПРИЛОЖЕНИЕ Б	25

ВВЕДЕНИЕ

В современных условиях разработки программного обеспечения скорость, надежность и стабильность процессов выпуска обновлений играют ключевую роль в обеспечении конкурентоспособности компании.

Ручные методы сборки, тестирования и развертывания программного обеспечения не только замедляют выпуск, но увеличивают вероятность ошибок из-за человеческого фактора, что может привести к сбоям в работе продуктов, потере данных и репутационным рискам.

Согласно книге Джеза Хамбла и Дэйвида Фарли об автоматизации сборки, целью любой ИТ-компании является переход от ручного процесса поставки программного обеспечения к надёжному, предсказуемому, контролируемому и максимально автоматизированному процессу с чётко понятными и количественно измеряемыми рисками [1].

В связи с этим автоматизация процессов непрерывной интеграции и доставки (CI/CD) становится необходимостью для любой компании, занимающейся активной разработкой ПО.

Непрерывная интеграция и непрерывное развертывание (CI/CD) включают регулярное объединение кода в основной репозиторий и автоматизированное развертывание изменений в рабочую среду. Это позволяет обнаруживать и исправлять ошибки на ранних стадиях, улучшая качество программного обеспечения и снижая время выхода на рынок [2].

Целью данного проекта является создание серверного приложения, которое позволит автоматизировать ключевые этапы сборки, тестирования и выпуска релизов программного обеспечения для компании СКБ Лаб.

Решение должно обеспечить сокращение времени развертывания новых версий, минимизировать влияние человеческого фактора на процесс сборки, повысить стабильность выпускаемых обновлений и снизить нагрузку на DevOps-команды.

Для достижения этой цели необходимо выполнить следующие задачи:

- сбор и обработка требований заказчика;
- анализ существующих аналогов;
- разработка бэклога на семестр;
- моделирование и прототипирование будущей системы;
- разработка дизайн-макетов;
- реализация следующих модулей системы: сборка нового релиза, проверка задач, создание релизной ветки, вывод diff коммитов и добавление новых задач в существующий релиз;
- интеграция front и back.

Актуальность проекта обусловлена несколькими ключевыми факторами. Во-первых, в условиях высокой конкуренции на рынке ИТ-решений скорость вывода новых функций и исправлений напрямую влияет на удовлетворенность клиентов и рыночные позиции компании.

Во-вторых, ручные процессы сборки и деплоя требуют значительных временных затрат и подвержены ошибкам, что может привести к критическим сбоям в работе ПО.

В-третьих, по мере роста компании и увеличения количества проектов масштабируемость процессов сборки и выпуска релизов становится все более сложной задачей, требующей стандартизации и автоматизации. Внедрение специализированного серверного приложения позволит компании СКБ Лаб не только ускорить выпуск, но и повысить качество выпускаемого ПО.

Разрабатываемый программный продукт предназначен для использования в ИТ-отделе компании СКБ Лаб, занимающейся разработкой различных типов программного обеспечения, включая веб-приложения, мобильные приложения и backend-сервисы.

Система будет интегрирована в существующую инфраструктуру и сможет масштабироваться в зависимости от потребностей конкретных

проектов. Это позволит применять ее как для небольших внутренних задач, так и для крупных коммерческих продуктов, требующих частых обновлений и высокой надежности процессов развертывания.

Уровни результата (минимальный / базовый / оптимальный):

- Минимальный: Реализация базового функционала для создания релизных веток и проверки статусов задач.
- Базовый: Добавление интеграции с Youtrack и GitLab, автоматизация Merge Request.
- Оптимальный: Полная автоматизация процесса сборки релизов с поддержкой адаптивного интерфейса и обеспечением безопасности.

1 Команда разработки

Савельева Александра Андреевна РИ-320940 – тимлид аналитик;
Бобина Анастасия Алексеевна РИ-320933 – дизайнер;
Жарков Семён Николаевич РИ-320935 – frontend-разработчик;
Лобанов Антон Александрович РИ-320949 – backend-разработчик;
Насибов Фариз РИ-320942 – backend-разработчик.

2 Определение проблемы

В ходе анализа существующих процессов сборки и выпуска релизов в компании СКБ Лаб были выявлены две ключевые проблемы, которые оказывают существенное влияние на эффективность разработки и качество конечного продукта.

Первая и наиболее критичная проблема – влияние человеческого фактора на возникновение ошибок в релизах. Несмотря на строгие регламенты и контроль, ручные операции на этапах сборки, тестирования и развертывания остаются источником рисков.

Даже опытные специалисты могут допустить ошибки при настройке окружения, ручном запуске тестов или развертывании сборки на production-серверах. Подобные ошибки часто приводят к критическим последствиям, включая отказы сервисов, потерю данных или необходимость экстренного отката версий.

Кроме того, ручные процессы требуют значительных временных затрат на проверку и согласование, что замедляет выпуск и снижает гибкость разработки. В условиях, когда компания выпускает множество релизов в разных проектах одновременно, масштаб проблемы только возрастает.

Вторая проблема – ограниченные возможности кастомизации в существующих аналогах. На рынке представлено множество инструментов для автоматизации CI/CD, однако большинство из них предлагают либо жестко заданный функционал, который не всегда соответствует специфике процессов компании, либо требуют сложной и трудоемкой адаптации.

Например, стандартные системы могут не поддерживать уникальные сценарии тестирования, применяемые в отдельных проектах компании, или не интегрироваться с узкоспециализированными инструментами, уже используемыми в инфраструктуре. Это вынуждает DevOps-команды искать обходные решения, что сводит на нет преимущества автоматизации и увеличивает нагрузку на сотрудников.

Кроме того, многие коммерческие решения обладают высокой стоимостью лицензирования и не всегда обеспечивают необходимый уровень гибкости даже за дополнительные платежи.

Эти проблемы напрямую влияют на скорость и качество разработки, увеличивая эксплуатационные расходы. Разрабатываемое серверное приложение призвано устранить данные недостатки за счет полной автоматизации критически важных этапов и предоставления широких возможностей для масштабирования функционала под конкретные нужды компании.

Это позволит не только минимизировать риски, связанные с человеческим фактором, но и обеспечить адаптивность системы к уникальным требованиям различных проектов внутри компании.

3 Анализ аналогов

В ходе анализа были выявлены два наиболее широко используемых аналога нашему продукту.

3.1 Jenkins

Основной функционал сервиса представляет из себя автоматический компилятор кода из репозитория, готовые тесты, деплой и планирование задач, а также проверка успешности сборки.

Сервис достаточно удобен в силу бесплатной основы, относительной гибкости и локальной работы – настроить его можно на собственном сервисе. Приведённые выше преимущества делают Jenkins достойным решением для малого бизнеса и индивидуальных предпринимателей в сфере разработки ПО, однако компания-заказчик СКБ Лаб не относится к перечисленной целевой аудитории. Именно поэтому были выделены основные недостатки системы, препятствующие её внедрению.

Сложность настройки: данный недостаток не представляет критической проблемы, однако в случае регулярных «падений» компания столкнётся с расстрчиванием финансовых и временных ресурсов на поддержание работоспособности.

С другой стороны, интуитивно сложный и относительно непонятный интерфейс может стать реальной проблемой для компании в силу траты огромного количества времени на обучение текущих сотрудников и удержание новых.

3.2 GitHub Actions

В данном продукте автоматические сборки запускаются при каждом пуше в репозиторий, так как система разработана на основе GitHub и является основным его дополнением. Тестирование в облаке поддерживает огромное количество языков, автоматический деплой и готовые шаблоны

позволяют максимально автоматизировать сборку. Параллельные задачи также ускоряют процесс.

Таким образом основными преимуществами GitHub Actions становятся скорость сборки, простота и максимально интуитивно-понятный интерфейс. Всё это дополняет недостатки предыдущего аналога, однако система не предназначена для кастомизации и не обладает необходимой гибкостью.

Также GitHub Actions привязан к GitHub, что абсолютно не подходит заказчику. Таким образом, использование данного аналога не просто замедлит сборку проектов, но потребует полного перехода на другую платформу, что представляется невозможным.

4 Подходы к решению проблемы

Исходя из проблем, которые были выделены в пункте 2, упор в будущей системе должен быть сделан на проверку каждого шага сборки релиза.

Таким образом, основным преимуществом будущего продукта станет проверка задач перед сборкой и проверка возможности автослияния. Это необходимо в целях максимального предотвращения ошибок, которые могут остаться незамеченными в ходе релиза.

Помимо всевозможных проверок, страница релиза будет содержать вывод diff коммитов, подсвеченных для повышения контроля над корректностью будущих релизов.

Минимальный набор функций, с другой стороны, позволит оптимизировать продукт специально для целей и нужд компании-заказчика. Данное преимущество системы позволит успешно и быстро внедрить её внутри компании и постепенно масштабировать с появлением необходимых функций и идей.

Также необходимо упомянуть то, что будущий сервис будет интегрирован с сервисами, используемыми СКБ Лаб. Это сделает систему максимально кастомизированной под текущие бизнес-процессы и инструменты компании.

5 Техническое задание

5.1 Функциональные требования

Страница сборки релиза

Поля для выбора:

- Выбор проекта (выпадающий список или поиск).
- Выбор релизной задачи (выпадающий список или поиск).
- Выбор ветки, из которой будет создана релизная ветка (выпадающий список).
- Чекбокс для пропуска pipeline (опционально).

Проверка задач

Проверка статусов задач:

- Все задачи, прикрепленные к релизной задаче, должны быть в статусе For Release или In Release.
- Если задачи находятся в других статусах, вывести список таких задач с указанием:
 - 1) Номер задачи.
 - 2) Заголовок.
 - 3) Статус.
 - 4) Теги.
 - 5) Автор.
 - 6) Разработчик.

Вывести предупреждение о наличии задач в неподходящих статусах.

Проверка связей задач:

- Если задача зависит от другой задачи, зависимая задача также должна быть включена в релиз.
- Если зависимая задача отсутствует в релизе, вывести список таких задач с предупреждением.
- Проверка зависимостей от других проектов: если задачи зависят от других проектов, вывести предупреждение о зависимости от другой платформы.

Создание релизной ветки

Если все проверки пройдены успешно: создать релизовую ветку из указанной ветки (если такая ветка еще не существует).

Если релиз собирается не из ветки develop:

- Взять задачи в статусе For Release.
- Создать Merge Request (MR) из ветки задачи в релизовую ветку.
- Если MR можно автоматически слить после прохождения pipeline, выполнить слияние.
- Если автоматическое слияние невозможно, назначить MR на релиз-менеджера.

Страница релиза

Отображение diff коммитов:

- Показать diff коммитов между новой версией и предыдущей.
- Из diff коммитов получить список задач, вошедших в релиз.

Сравнение списка задач:

- Сравнить список задач по коммитам со списком задач, прикрепленных к релизной задаче.
- Если обнаружены отличия, вывести предупреждение.

Проверка переноса коммитов:

- Проверить, что все коммиты из задачи перенесены в релиз.
- Если не все коммиты перенесены, вывести список недостающих коммитов.

Добавление новой задачи в релиз

- Найти существующую релизную ветку.
- Найти ветку задачи.
- Создать Merge Request (MR) из ветки задачи в релизную ветку.
- Если MR можно автоматически принять после прохождения тестов, выполнить слияние.

5.2 Нефункциональные требования

Интерфейс:

- Удобный и интуитивно понятный интерфейс.
- Поддержка адаптивного дизайна для работы на различных устройствах.

Производительность: быстрая обработка данных и отображение результатов проверок.

Безопасность:

- Обеспечение авторизации и аутентификации пользователей.

- Защита данных от несанкционированного доступа.

Интеграция:

- Интеграция с системами управления задачами (Youtrack).
- Поддержка работы с Git-репозиториями (GitLab).

6 Стек и архитектура разработки

6.1 Frontend-разработка

В качестве основного языка программирования был выбран TypeScript. Он лучше всего подходит для данного проекта, так как сильно схож с JavaScript, но уже имеет надстройку с жесткой типизацией, что позволяет получать минимум ошибок при разработке, а также в будущем минимизировать время на то, чтобы расширять проект и вникать в код, написанный ранее.

Библиотека React – одна из самых популярных библиотек для создания пользовательских интерфейсов. React предоставляет декларативный подход к разработке, что делает код более предсказуемым и удобным для отладки.

Компонентный подход позволяет разбивать интерфейс на независимые компоненты, что упрощает поддержку и масштабирование приложения. Наличие большого количества сторонних библиотек, инструментов и документации ускоряет процесс разработки и упрощает внедрение новых решений.

React идеально подходит для разработки масштабируемых и интерактивных веб-приложений, обеспечивая удобство в разработке, поддержку сообщества и гибкость в использовании.

Redux Toolkit (RTK Query) упрощает работу с глобальным хранилищем данных и предоставляет удобный способ для выполнения асинхронных запросов к API. RTK Query позволит сократить объем кода, связанного с обработкой состояний загрузки, кэшированием и повторными запросами, что делает разработку более удобной и эффективной.

6.2 Backend-разработка

В этом семестре для backend-части проекта было решено использовать следующий стек:

- Python – современный и самый популярный язык программирования, имеющий удобный синтаксис и много готовых библиотек для работы с API Gitlab и YouTrack.
- FastAPI – современный и производительный фреймворк для языка программирования Python, имеет встроенную документацию для RESTful API на основе Swagger и поддерживает асинхронные запросы.
- PostgreSQL – надежная и масштабируемая СУБД, хорошо совместимая с FastAPI, SQLAlchemy, и Alembic.
- SQLAlchemy – библиотека для языка программирования Python для работы с реляционными СУБД по технологии ORM. Имеет большую документацию.
- Alembic – библиотека для языка программирования Python, предназначенная для организации миграции баз данных в СУБД как автоматически, так и вручную.
- Docker – программное обеспечение для автоматизации развертывания готовых приложений в контейнере. В качестве оркестровщика будет использоваться Docker Compose.

Выбранный стек технологий позволяет эффективно реализовать все требуемые заказчиком функции. Наша команда имеет больший опыт работы с данными языками, фреймворками, и инструментами.

7 План работы на семестр

План работы на семестр был оформлен в качестве диаграммы Ганта (см. Приложение А). Основные его этапы представлены ниже:

- Анализ – проработка требований заказчика, анализ ЦА и аналогов, моделирование системы в нотации BPMN, разработка бэклога, ревью задач.
- Дизайн – разработка прототипов системы и создание дизайн-макетов.
- Разработка – вёрстка макетов, реализация основного функционала, интеграция с внешними системами, интеграция YouTrack и GitLab.
- Отчётность – сдача результатов по итерациям и подготовка итоговой отчётности по продукту.

Помимо диаграммы Ганта команда использовала доску задач в сервисе YouGile (рис. 1). Работа с доской задач имеет регулярный характер: в начале недели тимлид ставит задачу для каждого из членов команды в колонку «Новые задачи». Приступая к задаче, член команды переносит карточку в колонку «В работе», а по завершению – «Согласование».

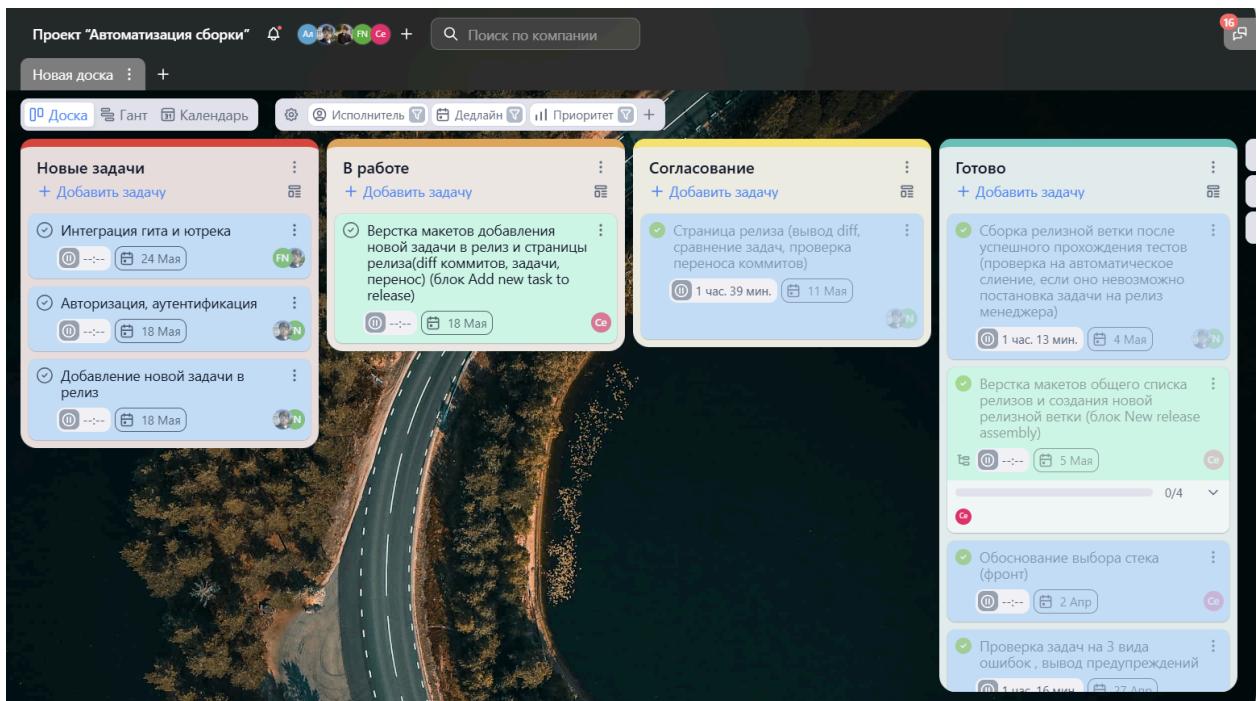


Рисунок 1 – Доска задач по проекту

Далее тимлид проверяет корректность задачи, даёт необходимые правки, а затем переносит в колонку «Готово». Данная схема работы была внедрена в целях усиления контроля над выполнением задач и их корректностью, что позволило ускорить работу и избежать критических несоответствий.

Помимо вышеперечисленных положений, в команде было введено правило фиксации времени, затрачиваемого на ту или иную задачу. Это позволило тимлиду наглядно отслеживать степень загруженности того или иного члена команды и эффективно корректировать нагрузку.

8.1 Отчёт о работе тимлида-аналитика

В течение данного семестра я исполняла обязанности тимлида и аналитика команды Novators в разработке серверного приложения для автоматизации сборки и выпуска релизов.

В качестве тимлида в начале семестра я договаривалась с заказчиком о сотрудничестве. В ходе 1 итерации изучала техническое задание заказчика, организовывала первые встречи и настраивала взаимодействие между командой разработки и куратором. Помимо этого, в мои обязанности входили планирование работ с помощью диаграммы Ганта и доски задач YouGile, а также согласование решений, изменений и прогресса с куратором.

В качестве аналитика в первую итерацию я проанализировала ТЗ заказчика и доработала его согласно видению команды. Также я проанализировала целевую аудиторию продукта и аналоги, благодаря чему составила подробное техническое задание.

На основе ТЗ я приступила к моделированию системы в нотации BPMN. По итогу работы были получены диаграммы основных модулей системы: создание нового релиза и добавление новой задачи в релиз (см. Приложение Б).

В начале 2 итерации я закончила аналитическую часть проекта и продолжала руководить процессом разработки как тимлид. Помимо этого, я занималась съёмками, написанием, оформлением и сдачей отчётности.

В середине второй итерации у команды возникли недопонимания касательно логики системы, однако все проблемы и вопросы улаживались достаточно быстро благодаря моим ответам на возникающие вопросы и дополнению технического задания и ревью дизайн-макетов и прототипов.

Также в ходе разработки несколько раз приходилось корректировать план работ для удобства интеграции модулей, но трудностей это не вызвало, а наоборот помогло развить навык быстрого реагирования.

В ходе 3 итерации я продолжала руководить процессом разработки и тестирования, а также приступила к написанию итогового отчёта и сбору итоговых артефактов. На данном этапе единственными проблемами были баги при тестировании, однако благодаря слаженному взаимодействию разработчики решали эти проблемы быстро и почти без моего участия.

По итогам своей работы в течение семестра могу сделать вывод, что, в отличии от прошлых проектов, где приобретаемые знания несли теоретический характер, в данном случае я получила множество практических навыков, таких как моделирование системы и управление требованиями.

Работа проходила не без сложностей, однако с большими кризисами ни я, ни моя команда не столкнулись. Приобретённые знания планирую и дальше применять в своей работе.

8.2 Отчёт о работе дизайнера

На протяжении семестра дизайнер выполняла задачу разработки интерфейса приложения. Для этого в первую очередь были разработаны прототипы, в которых был отражен базовый функционал приложения. После утверждения макетов она перешла к созданию финального дизайна. За основу внешнего вида интерфейса были взяты гайдлайны корпорации «Google LLC». Основными цветами стали тёмно-серый и лавандовый.

Во время первой итерации дизайнер приступила к созданию прототипов приложения. Основываясь на техническом задании, предоставленным компанией «СКБ-лаб», она составила макеты, отражающие основные функции, в частности: авторизацию, просмотр списка релизов, создание нового релиза, сборку проекта, экраны ошибок, создание новой

ветки и задачи и просмотр различий между задачами, коммитами и программным кодом. Макеты были созданы из простых фигур для того, чтобы наметить примерное расположение элементов и логику использования приложения.

Во время второй итерации дизайнер, получив обратную связь от тимлида и куратора, а также созданные аналитиком BPMN-модели, начала работать над итоговым дизайном. Компания предоставила гайдлайны и требования к оформлению приложения, в частности: предпочтение к тёмной теме и минималистичный стиль.

С учётом данных пожеланий дизайнер создала новые макеты на основе ранее созданных прототипов. Интерфейс стал более квадратным, уменьшив скругления, стиль перешел от залитых фигур к очерченному стилю. Акцентным цветом стал лавандовый. В этот момент началась разработка фронтенд стороны приложения, поэтому дизайнер подготавливала макеты к веб-разработке и вносила необходимые правки, которые требовались для корректной его работы.

Во время третьей итерации работа дизайнера была завершена, поскольку были выполнены все запланированные задачи на семестр.

Для выполнения своей работы дизайнер пользовалась онлайн-сервисом «Figma» как основным инструментом. Он предоставляет все необходимые функции для создания интерактивного дизайна интерфейсов, а также позволяет одновременно работать над проектом в команде. В частности, «Figma» имеет возможность работать с векторной графикой; настраивать автоматическую верстку элементов («Auto Layout»), что особенно удобно при работе над веб-сервисом; автоматизировать и систематизировать дизайн-токены, содержащие в себе HEX-коды цветов, размеры страниц, названия, размеры скруглений и так далее. Самым большим ее преимуществом является онлайн доступ к макетам, что облегчает работу разработчикам и аналитикам. Исходя из вышеописанного, выбор пал именно на этот инструмент работы.

Дизайнер применяла разные методики, что позволило упростить ее работу. Основной методикой было дизайн-мышление. Оно представляет из себя несколько этапов работы над новым продуктом или новой задачей. Первым этапом является эмпатия – понимание желаний и нужд будущего пользователя. Далее идет фокусировка – формирование определенных целей. За этим идет этап генерации идей – построение целей на основе идей. А дальше реализация – прототипирование и тестирование.

Все это позволяет создавать более полезные для пользователя интерфейсы, которые будут ему понятны и удобны. Данный метод чаще всего используется именно в UX/UI-дизайне и разработке, поскольку он очень гибкий (можно повторять необходимый этап столько раз, сколько нужно, чтобы прийти к желаемому результату) и очень индивидуальный.

Применив его, мы поняли, что нашим пользователям нужен сервис, который упростит задачу создания новых релизов и администрирования существующих.

Также мы поняли, что нашей целевой аудитории нужен сервис, в который интегрированы их любимые сервисы, такие как «GitLab» от компании «GitLab Inc.», который используется для хранения в нем кодовых файлов, и «YouTrack» от компании «JetBrains», используемый для организации работы команд. После этого мы разработали ряд целей, которые нужно было выполнить до окончания семестра.

Этап генерации идей был пропущен, поскольку заказчиком было предоставлено очень подробное техническое задание, и мы знали, какие требования были выставлены к работе этого приложения. Далее, как было описано выше, дизайнер создала прототипы, на их основе итоговый дизайн и позже он был отдан на тестирование работоспособности.

Применение данных инструментов и методик позволило дизайнеру создать макеты интерфейса, которые были позже использованы для создания фронтенд-стороны приложения. Они учитывают требования и пожелания заказчика, интерактивны и полностью работоспособны, а также

поддерживают вывод уведомлений об ошибках и успешно выполненных задачах.

Во время работы дизайнера не столкнулась с серьёзными трудностями, кроме выполнения задач к указанному сроку. Причиной этой проблемы являлась высокая нагрузка во время учебы, и она была решена путем перераспределения задач и времени, что позволило выполнить все задачи вовремя. Дизайнер довольна своей работой на протяжении этого семестра и намерена продолжать в том же духе далее.

К концу семестра дизайнер научилась применять новую методику решения конкретных задач «Дизайн-мышление», что развило ее навык справляться с нестандартными задачами. Также она узнала, как более эффективно разрабатывать библиотеки элементов («UI-kit») и систематизировать их, а также как правильно выстраивать логику перехода на экраны приложения.

8.3 Отчёт о работе frontend-разработчика

Первый спринт стал периодом активного обсуждения и планирования. Передо мной стояла задача разработки фронтенд-части проекта. После тщательного анализа доступных технологий было принято решение использовать TypeScript, React и RTK Query.

Этот стек был выбран для обеспечения структурированности API-запросов и удобства взаимодействия с ними. Однако на тот момент моих навыков оказалось недостаточно, поэтому значительная часть времени была посвящена изучению этих технологий. К концу спринта были определены цели на следующий этап: развернуть React-приложение, настроить роутинг и сверстать макеты страниц на основе дизайна из Figma.

Второй спринт начался с подготовки рабочего окружения. Было развёрнуто React-приложение, настроены необходимые инструменты для разработки, а также организована логичная и удобная структура папок и

файлов. Это позволило обеспечить чистоту кода и простоту его дальнейшей поддержки.

Далее была выполнена настройка роутинга с использованием библиотеки react-router-dom. Это обеспечило удобную навигацию между страницами и заложило основу для будущего масштабирования приложения. После этого я приступил к вёрстке макетов. На основе дизайна из Figma было создано семь страниц, каждая из которых тщательно прорабатывалась для достижения pixel-perfect соответствия. Дополнительно были добавлены модальные окна, которые отображали уведомления о успешных или ошибочных действиях пользователя.

В процессе работы выяснилось, что TypeScript, несмотря на свои преимущества, добавлял избыточную сложность для текущего проекта. После обсуждения с командой было принято решение перейти на JavaScript, который оказался более гибким и подходящим для наших задач. Все поставленные цели были достигнуты, и на следующий спринт была запланирована интеграция с бэкендом через API и тестирование системы.

Третий спринт был посвящён подключению фронтенда к бэкенду. Для организации API-запросов использовалась методология RTK Query, которая доказала свою эффективность. Все запросы были структурированы и вынесены в отдельные модули, что обеспечило их лёгкий доступ из любой части приложения. Это значительно упростило работу с данными и ускорило процесс разработки.

После успешной интеграции было проведено всестороннее тестирование системы. Проверялась корректность работы API-запросов, отображение данных на страницах, а также взаимодействие пользователя с интерфейсом. Все обнаруженные недочёты оперативно устранялись, и в итоге система продемонстрировала стабильную и предсказуемую работу.

По итогу все поставленные задачи были выполнены в срок. Проект прошёл путь от планирования и изучения технологий до полноценной реализации и тестирования. Переход с TypeScript на JavaScript оказался

оправданным, а использование RTK Query значительно упростило работу с API. В результате был создан функциональный и удобный в поддержке продукт.

8.4 Отчёт о работе backend-разработчиков

8.4.1 Отчёт Лобанова Антона

В рамках реализации данного проекта я выполнял обязанности разработчика серверной части веб-приложения, предназначенного для автоматизации процессов сборки, подготовки и выпуска релизов программного продукта. Основная цель заключалась в создании функциональной системы, способной взаимодействовать с внешними сервисами, обрабатывать задачи и коммиты, а также предоставлять удобные инструменты для управления релизами.

Мною была спроектирована и реализована логика приложения, обеспечивающая взаимодействие с такими инструментами, как GitLab и YouTrack, посредством их API. Это позволило наладить получение и обработку актуальных данных о задачах, связанных с ними коммитах и пользовательской активности, а также обеспечить отображение этой информации в структурированном виде.

В техническом плане проект был реализован с использованием современного и широко распространённого стека технологий. В качестве основного языка программирования был выбран Python версии 3.11, обладающий богатой экосистемой и поддержкой множества библиотек.

В качестве фреймворка для разработки REST API использовался FastAPI — быстрый, гибкий и масштабируемый инструмент, предоставляющий разработчику удобные инструменты для построения роутов, обработки ошибок и документирования кода. Для валидации данных мы использовали pydantic. Это значительно ускорило и упростило процесс создания API и сделало разработку более комфортной.

Дополнительно, я разработал и внедрил систему авторизации и аутентификации пользователей, что позволило организовать защищённый доступ к функционалу приложения, а также разграничение ролей в рамках системы. Это было важно для обеспечения безопасного взаимодействия между различными участниками процесса разработки, релиза и тестирования.

Для организации работы с базой данных использовалась связка SQLAlchemy и Alembic. SQLAlchemy обеспечивал объектно-реляционное отображение данных, позволяя работать с таблицами в виде Python-классов. Alembic, в свою очередь, отвечал за управление миграциями — создание, изменение и удаление таблиц и полей в базе данных. В качестве СУБД использовалась PostgreSQL, известная своей надёжностью, масштабируемостью и хорошей поддержкой со стороны сообщества.

Для упрощения развёртывания и изоляции окружения проекта была настроена контейнеризация с использованием Docker и Docker Compose. Эти инструменты позволили запускать все компоненты приложения в изолированных средах, обеспечивая единообразие среды разработки и продакшн-окружения. Это особенно важно в командной работе, где требуется, чтобы все члены команды работали в идентичных условиях.

Интеграция с внешними сервисами GitLab и YouTrack осуществлялась через их REST API, с применением токенов доступа, что обеспечивало безопасность и гибкость взаимодействия. Данные сервисы являются ключевыми в процессе управления задачами и релизами, поэтому стабильная и надёжная интеграция с ними имела первостепенное значение для успешного функционирования приложения.

На прикладном уровне мною была реализована возможность автоматического создания релизной ветки, формирования Merge Request по каждой задаче, а также проведение автоматической проверки возможности их автослияния.

Также была реализована страница релиза, где отображалась информация о задачах, коммитах и различиях между релизами, а также проверялась корректность переноса всех изменений.

Дополнительно в приложение была добавлена функция, позволяющая включать новые задачи в существующий релиз, что сопровождалось созданием соответствующего Merge Request и его отслеживанием в системе. Была предусмотрена и возможность просмотра всех релизов и анализа их состава.

Это дало возможность в любой момент времени получить полную картину состояния релизов, истории изменений и взаимодействия задач с кодовой базой.

Работа над проектом сопровождалась рядом технических сложностей. Наибольшие трудности вызвала интеграция с API GitLab и YouTrack. Их документация была не всегда полной и понятной, приходилось вручную анализировать ответы серверов, тестировать запросы и добиваться стабильной работы.

Также требовалась детальная проработка логики обработки задач, учёт различных состояний и возможных проблем с коммитами и ветками. Немалое внимание пришлось уделить обеспечению стабильности работы всех компонентов и проверке всех сценариев взаимодействия.

Тем не менее, в ходе реализации проекта мне удалось получить ценнейший практический опыт разработки решения для современной ИТ-компании.

Я приобрёл навыки интеграции сторонних сервисов, отладки взаимодействия с внешними API, организации серверной архитектуры и контейнеризации проекта. Я также углубил свои знания в области работы с базами данных, управления миграциями и развертыванием приложений на сервере.

Особо стоит отметить, что в процессе работы над этим проектом мне пришлось заниматься не только программированием, но и проектированием

архитектуры, анализом бизнес-логики, тестированием и отладкой. Это позволило мне развить взгляд на разработку программного обеспечения и лучше понять внутренние процессы, происходящие при подготовке релизов и их выпуске.

В итоге, благодаря проделанной работе, все задачи, поставленные в рамках проекта, были успешно решены. Были реализованы все требования, обозначенные заказчиком, и достигнуты все цели.

Разработанное приложение позволяет в автоматическом режиме управлять процессом релизов, начиная с анализа задач и заканчивая созданием и обработкой Merge Request. Это существенно сокращает объём ручной работы, снижает риск ошибок и делает процесс выпуска новых версий продукта более надёжным, прозрачным и контролируемым.

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
b2a2dadd5fcf	newneobuild-api	"uvicorn app.main:app"	6 hours ago	Up 6 hours	0.0.0.0:8000->8000/tcp	newneobuild-api-1
bf5b45dd4896b	jetbrains/youttrack:2024.3.78677	"/bin/bash /run.sh"	6 hours ago	Up 6 hours	0.0.0.0:8080->8088/tcp	newneobuild-youttrack-1
c9f4ab678eb42	postgres:15	"docker-entrypoint.s..."	6 hours ago	Up 6 hours	0.0.0.0:5432->5432/tcp	newneobuild-db-1
#28caab73e62	gitlab/gitlab-ce:16.9.1-ce.0	"/assets/wrapper"	6 hours ago	Up 6 hours (healthy)	0.0.0.0:22->22/tcp, 0.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp	newneobuild-gitlab-1

Рисунок 1 – информация о работающих Docker-контейнерах серверной части приложения

8.4.2 Отчёт Насибова Фариза

Проект Neobuild строился на стыке современных инструментов и проверенных практик. Мы выбрали Python и FastAPI не случайно: асинхронная природа этого фреймворка позволила писать высокопроизводительные эндпоинты, астроенная документация Swagger облегчила взаимодействие с фронтендом и быстрое подключение новых коллег.

В качестве базы данных остановились на PostgreSQL, используя SQLAlchemy для удобного маппинга и Alembic для управления миграциями: после первого опыта с «пропавшими» индексами мы включили их в скрипты явным образом и всегда проверяли целостность схемы перед каждым деплоем. Всё приложение было упаковано в Docker-контейнеры и описано

через Docker Compose, что дало неизменно одинаковое окружение для разработки и упрощённый CI/CD-процесс.

С точки зрения безопасности я реализовал аутентификацию на основе JWT, а для хеширования паролей применил библиотеку Passlib с алгоритмом bcrypt. Такой связки оказалось достаточно, чтобы на первом этапе защитить наши эндпоинты от несанкционированных запросов. К тому же добавление CORS-политики на уровне FastAPI позволило чётко контролировать, какие фронтенд-домены имеют право обращаться к сервису.

В ходе работы я существенно прокачал навыки асинхронного программирования в Python. Постоянно сталкиваясь с необходимостью выстраивать корректную последовательность зависимых запросов и не допускать гонок, я глубоко изучил принципы `async/await` и понял, как на практике разгружать систему при большом числе одновременных запросов.

Миграции в Alembic стали для меня отличным уроком аккуратности: ныне я всегда проверяю, что после каждого изменения схемы все внешние ключи и индексы на месте, и прописываю их явно в скриптах.

Работа в Docker-окружении дала мне понимание того, как на уровне сети Docker Compose связывает сервисы и как настроить порты так, чтобы разные контейнеры не конфликтовали. Благодаря этому я смог довести локальную сборку до такого же состояния, в котором она будет работать на тестовом сервере, без дополнительных ручных правок.

Интеграция с внешними API, таким как YouTrack и GitLab, научила меня правильно обрабатывать коды ошибок и учитывать ограничения по количеству запросов. Появившиеся в процессе задачи по реализации механизма пауз и обработки ответов с HTTP-кодом 429 помогли сделать взаимодействие более стабильным и предсказуемым.

В будущем я планирую вынести все секретные ключи и параметры конфигурации в файл `env` — это избавит команду от постоянного перекатывания настроек при смене окружений. Одним из первых шагов станет подключение автоматических тестов на базе `pytest`, чтобы

минимизировать ручное тестирование и отлавливать регрессии на ранних стадиях. Ещё одна задача — реализовать регистрацию пользователей и плавный переход к полноценному продакшн-развёртыванию: это позволит нам перейти от временной базы аккаунтов к реальной.

В результате проделанной работы я не только воплотил в коде все ключевые требования заказчика, но и получил бесценный опыт проектирования микросервисной архитектуры, настройки контейнеризации и надёжной интеграции с внешними системами.

Проект показал мне, как важно сочетать гибкие методики разработки с тщательным контролем качества на каждом этапе, и дал чёткое понимание дальнейших шагов по эволюции Neobuild.

9 Результаты

В результате командой было разработано серверное приложение для автоматизации сборки и выпуска релизов, которое на момент сдачи отчёта (27.05.2025) находится на этапе сборки и тестирования.

Основные блоки – создание нового релиза и добавление новой задачи в релиз – были разработаны в соответствии с требованиями заказчика. Страница релиза и список релизов были максимально оптимизированы под нужды компании. Результат соответствует видению и требованиям заказчика и куратора.

ЗАКЛЮЧЕНИЕ

В рамках семестра работа над серверным приложением для автоматизации сборки и выпуска релизов была полностью завершена. Командой разработки были выполнены следующие задачи:

- проанализированы целевая аудитория, аналоги и требования заказчика;
- разработано техническое задание, включающее функциональные требования системы и нефункциональные требования к стеку разработки, безопасности и архитектуры;
- разработаны функциональные прототипы системы;
- разработаны дизайн-макеты;
- свёрстаны дизайн макеты;
- разработаны основные модули и настроено взаимодействие между ними;
- настроена интеграция с YouTrack и Gitlab;
- настроена интеграция между frontend и API продукта;
- проведены итоговые тесты.

По завершению проекта командой было разработано серверное приложение для автоматизации сборки, состоящий из 2 основных модулей: создание нового релиза и добавление новой задачи в релиз.

Помимо основных модулей была реализована страница релиза, содержащая основную информацию о релизе и diff коммитов для повышения контроля.

В ходе работы были использованы такие инструменты как: Figma, React, TypeScript, Redux Toolkit, Python, PostgreSQL.

В заключении хотелось бы отметить, что несмотря на возникавшие трудности, каждый участник команды приобрёл несколько невероятно

важных навыков для будущей работы. Полученный результат удовлетворяет требования заказчика, куратора и самой команды.

Команда разработки уверена, что данный продукт станет началом автоматизации и цифровизации бизнес-процессов компании СКБ Лаб, а будущее масштабирование позволит повысить производительность и качество работы и вывести продукты компании на новый уровень.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хамбл Джез, Фарли Дэвид Непрерывное развёртывание ПО: автоматизация процессов сборки, тестирования и внедрения новых версий программ. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2017. – 432 с. : ил. – Парал. тит. англ.
2. Мелков И. А. Применение DevOps-практик для ускорения разработки и развёртывания программного обеспечения : материалы Междунар. науч.-практ. конф., 15 авг. 2024 г., Пенза / МЦНС «Наука и Просвещение». – 2024. – 160 с.
- 3.

Приложение А

№	Название	Ответственный	Длительность	Дата начала
Анализ				
	Анализ ЦА, изучение требований заказчика	Савельева	1 неделя	24.03.2025
	Моделирование работы системы	Савельева	1 неделя	24.03.2025
	Разработка плана и создание доски задач	Савельева	1 неделя	24.03.2025
	Ревью прототипов	Савельева	2 недели	31.03.2025
	Ревью дизайн-макетов	Савельева	2 недели	14.04.2025
Дизайн				
	Прототипы сборки релиза, проверки задач, создания релизной ветки	Бобина	2 недели	24.03.2025
	Прототипы страницы релиза и добавления новой задачи в релиз	Бобина	1 неделя	31.03.2025
	Согласование прототипов	Бобина	1 неделя	07.04.2025
	Дизайн-макеты сборки релиза, проверки задач, создания релизной ветки	Бобина	1 неделя	07.04.2025
	Дизайн-макеты страницы релиза и добавления новой задачи в релиз	Бобина	1 неделя	14.04.2025
	Согласование дизайна-макетов	Бобина	1 неделя	21.04.2025
Фронт				
	Проработка стека и применяемых технологий	Жарков	2 недели	24.03.2025
	Вёрстка макетов сборки релиза, проверки задач, создания релизной ветки	Жарков	1 неделя	21.04.2025
	Вёрстка макетов страницы релиза и добавления новой задачи в релиз	Жарков	1 неделя	05.05.2025
	Интеграция фронта и бэк	Жарков	1 неделя	02.06.2025
Бэк				
	Проработка стека и применяемых технологий	Лобанов, Насибов	2 недели	24.03.2025
	Сборка релиза	Лобанов, Насибов	1 неделя	14.04.2025
	Проверка задач, вывод предупреждений	Лобанов, Насибов	1 неделя	21.04.2025
	Создание релизной ветки	Лобанов, Насибов	2 недели	28.04.2025
	Страница релиза (вывод diff, сравнение задач, проверка переноса коммитов)	Лобанов, Насибов	2 недели	12.05.2025
	Добавление новой задачи в релиз	Лобанов, Насибов	2 недели	26.05.2025
	Авторизация, аутентификация	Лобанов, Насибов	1 неделя	02.06.2025
	Интеграция с Gitlab и YouTrack	Лобанов, Насибов	1 неделя	09.06.2025
Птоси				
	Подготовка к сдаче 1 итерации	Савельева	1 неделя	31.03.2025
	Подготовка к сдаче 2 итерации	Савельева	1 неделя	
	Подготовка к сдаче 3 итерации	Савельева	1 неделя	
	Оформление отчёта и презентации к защите	Савельева	2 недели	

Рисунок 1 – Структурная часть диаграммы

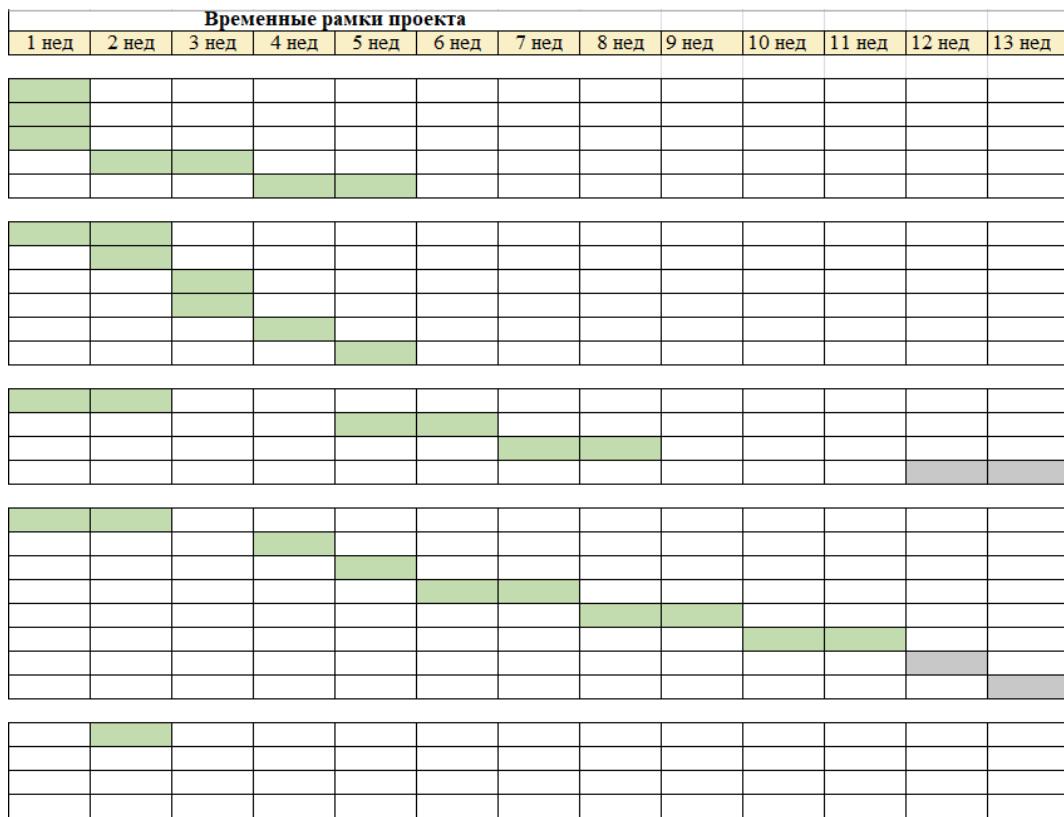


Рисунок 2 – Визуализация распределения задач на семестр

ПРИЛОЖЕНИЕ Б

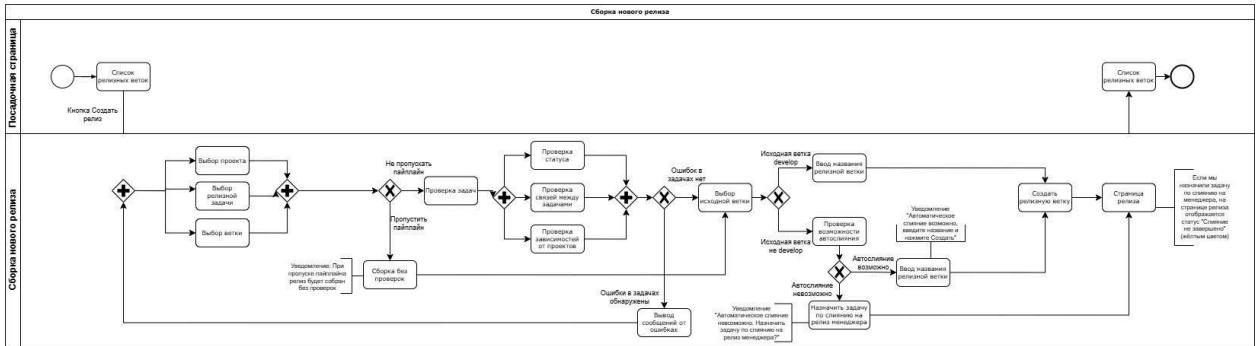


Рисунок 3 – Модель сборки нового релиза

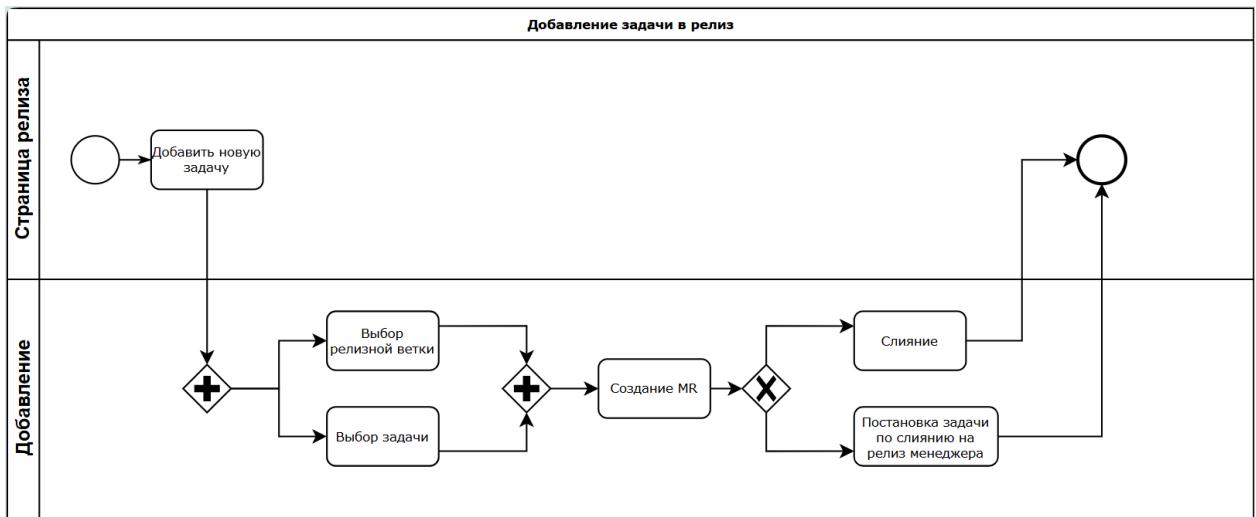


Рисунок 4 – Модель добавления новой задачи в релиз