Санкт-Петербургский государственный университет

НЕУМОИНА Елизавета Петровна

Выпускная квалификационная работа

Разработка серверной части ПО для задач найма и обучения сотрудников с использованием многомодульной архитектуры

Уровень образования: бакалавриат

Направление 02.03.02 «Фундаментальная информатика и информационные технологии»

Основная образовательная программа СВ.5003.2018 «Программирование и информационные технологии»

Научный руководитель:
кандидат физ.-мат. наук, доцент
кафедры компьютерного моделирования
и многопроцессорных систем:
Корхов Владимир Владиславович

Рецензент:
генеральный директор,
Risen Sun Agency
Главанарь Макар Георгиевич

Санкт-Петербург

2022 г.

Оглавление

Введение

Успех любой компании зависит от того, насколько профессиональны ее сотрудники. Программы обучения не только предоставляют возможность сотрудникам улучшить свои профессиональные навыки, но и позволяют компаниям повысить производительность труда и, как следствие, увеличить прибыль.

Также они помогают снизить текучесть кадров, что, согласно исследованию Work Institute 2020 года, важно для чистой прибыли компании. Согласно отчету, добровольная текучесть кадров обходится американским предприятиям более чем в 630 миллиардов долларов в год. Сотрудники, у которых есть регулярные возможности учиться, развиваться и продвигаться по службе, с большей вероятностью останутся в компании [1].

Правильная организация обучения сотрудников в итоге позволяет получить экономическую выгоду для предприятия. Для сотрудников же повышение квалификации дает уверенность в своей компетентности, сохранении рабочего места и высокий профессиональный статус, который также положительно влияет на развитие предприятия.

Постановка задачи

Целью данной работы является разработка серверной части программного обеспечения (далее по тексту ПО) для задач найма и обучения сотрудников, предназначенной для применения компаниями с целью обеспечения эффективного роста навыков сотрудников и удобства работы HR-специалистов (human resources или человеческие ресурсы – далее по тексту HR).

Рост компетенций сотрудника реализуется с помощью построения индивидуального плана развития, содержащего навыки, необходимые для изучения на определенном уровне. В связи с этим необходимо спроектировать гибкую систему с набором материалов для обучения определенным навыкам. Под материалом понимается документ, содержащий в себе источники информации.

Так как помимо пользователей, их индивидуальных планов развития и системы навыков в сервисе подразумевается ряд другой функциональности (выдача мероприятий, вакансий и кандидатов на должность компании), не имеющей сильной зависимости от уже описанной, необходимо выбрать такую архитектуру ПО, чтобы она была высоко масштабируемой и расширяемой.

Таким образом, приложение должно иметь многомодульную архитектуру, структура которой позволяла бы добиться возможности имплементации новых компонентов системы с уникальной бизнес-логикой, не вызывая изменений в других компонентах и имея возможность быстрой интеграции с ними. Вместе с этим, нефункциональные требования к системе подразумевают низкую трудозатратность в разработке и содержании сервиса, что в свою очередь предполагает необходимость в использовании монолитной архитектуры [2]. Однако в данной работе необходимо предусмотреть шаги для масштабирования сервиса в высоконагруженной среде с минимальными трудозатратами. В этом случае монолитная

архитектура не является подходящим решением, так как не подразумевает возможности горизонтального масштабирования, при использовании которого используется несколько баз данных, и данные распределяются между ними.

Обзор литературы

В силу отсутствия нефункциональных требований по количеству пользователей в системе необходимо предусмотреть поведение и набор необходимых действий системы в условиях высоконагруженной среды. Планируется спроектировать два возможных режима запуска разрабатываемого приложения: монолитный и распределенный. Для реализации подобного решения необходимо спроектировать многомодульную архитектуру, позволяющую запускать отдельно взятые модули в системе как независимый процесс без необходимости модификации остальных модулей системы.

В источнике [3] автор на примере развития сервиса по доставке еды показывает, как при небольших размерах приложения может быть полезна монолитная архитектура: легко вносить значимые изменения, развертывать, тестировать и масштабировать. Однако при увеличении размеров сервиса возникает огромное количество трудностей, среди которых автор описывает проблемы с масштабированием. С увеличением количества пользователей различные части монолита могут запросить разные требования к ресурсам. Некоторые модули стоит развертывать на серверах с большим объемом оперативной памяти (например, данные о ресторанах), а некоторые - на серверах с большими вычислительными ресурсами (например, модуль, в котором присутствует функциональность обработки изображений). Поскольку все монолитное приложение разворачивается на одном сервере, возникает проблема долгой работы системы.

При переходе к микросервисной архитектуре важно понимать, каким образом будет достигаться масштабирование. В источнике [4] автор описывает варианты масштабирования с помощью XYZ-куба. Ось X предполагает масштабирование с использованием нескольких экземпляров системы и распределением нагрузки между ними. Ось Y также представляет клонирование экземпляров, но разделение происходит по какому-либо

признаку: действиям или данным (например, по идентификатору пользователя). Распределение нагрузки по оси X и Y можно сделать, используя монолитную архитектуру приложения, однако исключается возможность совместного использования двух вариантов. Масштабирование по оси Z предполагает разделение приложения на сервисы, каждый из которых может быть масштабирован по оси X или Y. Такое разделение помогает в масштабировании как транзакций и данных, так и наборов команд и процессов.

Помимо преимуществ с масштабированием и балансировкой нагрузки, использование микросервисной архитектуры предполагает ряд других. Автор в источнике [5] подробно описывает особенности, которыми обладает такая архитектура:

- возможность использования различных технологий реализации с целью достижения более высокой производительности;
- устойчивость к отказам;
- простота развертывания, так как при внесении изменений в микросервис требуется перезапустить только его;
- более простой рефакторинг кода.

Как было сказано выше в данном разделе, для реализации перехода от монолитной к микросервисной архитектуре требуется использовать многомодульную, поскольку именно она обеспечивает наиболее простое переключение между двумя типами архитектур и сочетает в себе ряд преимуществ и монолитной, и микросервисной архитектур.

Разрабатываемый сервис относится к области по управлению человеческими ресурсами и имеет ряд аналогов. Подробнее об этом будет написано в разделе 1.2. главы 1.

Глава 1. Описание продукта

1.1. Описание сервиса

Разрабатываемое ПО представляет собой сервис, где сотрудники компании смогут изучать новые навыки и в последующем повышать свою квалификацию. Также приложение предполагает работу с различными артефактами компании: мероприятиями, вакансиями и кандидатами на должность. В сервисе есть несколько классов пользователей: администратор, HR-сотрудник, сотрудник, интервьюер (назначается временно любому из классов) и ревьюер (назначается временно любому из классов).

Прежде чем переходить к описанию возможностей каждого из классов пользователей, необходимо рассмотреть основные разделы сервиса:

- **Библиотека навыков** — файловая система с набором материалов для обучения определенным навыкам.
- **Роадмап** — индивидуальный план развития сотрудника, разделенный на этапы.
- **Мероприятия** — мастер-классы, собрания, проводимые компанией.
- **Список пользователей** — список сотрудников компании.
- **Вакансии** — свободные должности компании.
- **Кандидаты** — специалисты, претендующие на определенную вакансию.
- **Уведомления** — список извещений пользователя о каких-либо событиях.

Администратор имеет доступ ко всей функциональности сервиса, приведенной ниже.

- Работа со своим профилем, включая просмотр и редактирование данных;

- работа с библиотекой навыков, включая просмотр и редактирование файловой системы (загрузка файлов, объединение в папки, удаление содержимого);
- работа с «роадмап» сотрудников, включая просмотр и редактирование «роадмап» (изменение статуса, изменение и удаление содержимого);
- работа с мероприятиями компании, включая просмотр, создание и редактирование мероприятия, подтверждение участия в определенном мероприятии;
- работа с пользователями системы, включая создание, удаление, просмотр и редактирование данных о сотруднике;
- работа с вакансиями компании, включая создание, редактирование (в том числе изменение статуса вакансии), просмотр и удаление вакансии;
- работа с кандидатами на должность, включая создание, редактирование (в том числе изменение статуса рассмотрения), просмотр и удаление;
- работа с уведомлениями.

Сотрудник имеет ограниченный доступ к сервису:

- работа со своим профилем, включая просмотр и ограниченное редактирование данных (фотография профиля и приватная информация);
- работа с библиотекой навыков, включая просмотр файловой системы;
- работа со своей «роадмап», включая просмотр активной карты и выбор навыка для изучения, изменение статуса текущего навыка, отправка уведомления о завершении обучения по текущему плану, просмотр пройденных планов и изученных навыков;

- работа с мероприятиями, включая просмотр мероприятий и подтверждение участия в определенном мероприятии;
- работа со списком пользователей, включая просмотр профиля сотрудника;
- работа с уведомлениями.

HR-сотрудник имеет тот же доступ к сервису, что и класс пользователей Сотрудник. Также ему доступна функциональность класса Администратор по работе с пользователями, вакансиями и кандидатами на должность.

Ревьюер — временная роль, назначаемая определенному Сотруднику для проверки знаний другого Сотрудника. Доступ к сервису ограничивается классом Сотрудник и расширяется функциональностью работы со списком сотрудников, доступных для ревью (выбор сотрудника, просмотр, редактирование и удаление его «роадмап»).

Интервьюер — временная роль, назначаемая определенному Сотруднику для проведения собеседования с кандидатом на должность. Доступ к сервису ограничивается классом Сотрудник и расширяется функциональностью работы с кандидатами на должность (просмотр, редактирование, изменение статуса рассмотрения).

Далее будет описан наиболее часто используемый сценарий в сервисе. Авторизуясь в системе, сотрудник может перейти в свой индивидуальный план развития и посмотреть навыки для изучения. Навык, по которому изучены материалы, он отмечает как пройденный. В соответствии с пройденными навыками формируется текущий процент прохождения «роадмап».

Все материалы, из которых формируются навыки, хранятся в папках в библиотеке навыков, поэтому предполагается, что также часто сотрудники могут перемещаться между папками в файловой системе. Вернуться на главную страницу библиотеки навыков можно, используя навигационную

систему (breadcrumbs/хлебные крошки), показывающую путь от корня до того элемента, где сейчас находится пользователь.

В разделе 3.4. главы 3 будут описаны результаты объемного тестирования, в котором имитировалось поведение системы на получение данных большого числа пользователей. Параллельно с этим производился обход дерева навыков (перемещение по библиотеке навыков), и для каждого навыка запрашивался путь до корневого узла (получение хлебных крошек).

1.2. Преимущества и существующие аналоги

В настоящее время у разрабатываемого ПО нет публичных достаточно хороших аналогов, решающих проблему с персоналом.

Существует большое количество онлайн-платформ, которые обучают IT (Information Technology) профессиям и выпускают специалистов, не умеющих качественно решать реальные задачи. Хорошие специалисты переходят в другие, более высокие, должности, а новых такого же уровня не приходит. Крупные компании, такие как Сбер и VK, могут позволить себе набрать большое количество стажеров и обучить их до требуемого уровня, однако такая модель недоступна для малого бизнеса, который вынужден обучать новых сотрудников своими силами. Однако выстроить процесс обучения — сложная и дорогая для любого бизнеса задача.

Teachbase предлагает в качестве решения этой проблемы обучающую платформу [6], на которой можно создавать онлайн-курсы и выстраивать индивидуальный план обучения, состоящий из нескольких курсов с возможностью просмотра аналитики и отчетов. Также платформа предлагает интеграцию с другими сервисами и видоизменение решения в зависимости от нужд компаний-заказчиков. Несмотря на то, что на платформе существует возможность построения индивидуального плана, она больше ориентирована на онлайн-курсы, которые достаточно сложно разработать под определенного человека.

Более похожий на разрабатываемую систему аналог — платформа ispringlearn [7]. Пользователям предоставляется конструктор онлайн-курсов, которые впоследствии можно объединять в траектории, а также возможность создания своей базы знаний, проведения тестирования 360, написания статей внутри платформы, управления мероприятиями и разделения сотрудников по ролям.

В крупных компаниях уже существуют внутренние системы обучения, но, как правило, это специфические продукты, реализованные конкретно под процесс обучения этих компаний, поэтому их сложно использовать другим бизнесам.

Вышеперечисленные платформы не совсем подходят под решаемые задачи, потому что разрабатываемый сервис ориентирован именно на индивидуальную работу с сотрудниками и построение для каждого из них персонального плана развития в соответствии с текущими знаниями. Помимо этого сервис стремится поддержать HR-специалистов, которые могут отслеживать прогресс сотрудников компании, анализировать текущее состояние кадров компании, создавать вакансии и проводить отбор кандидатов. Такая функциональность отсутствует в вышеперечисленных аналогах.

Как было сказано выше, обучение — процесс непростой. Нужно разбираться в том, как мыслят люди, как с ними взаимодействовать и каким образом обучать. Создаваемое решение обладает высокой гибкостью, которая необходима в этой области, благодаря использованию многомодульной архитектуры. Проводя опросы, общаясь с компаниями и понимая их требования, можно быстро вносить изменения в систему.

Подводя итог, можно отметить, что разрабатываемое решение — полноценный HR-инструмент, который помогает не только выращивать специалиста за предсказуемое количество времени и выстраивать процесс с прогнозируемыми результатами как для обучаемого, так и для тех, кто

обучает, но и оперативно выстраивать взаимодействие с командой в виде проведения мероприятий и создавать базу кандидатов для быстрого решения проблем с кадрами.

Так как существующие аналоги не выкладывают кодовую базу в открытый доступ, нельзя будет сравнить с количественной точки зрения время отклика на большое количество запросов в систему. Однако ожидаемо можно оценить следующее:

- полное соответствие разрабатываемого решения нуждам компании;
- безопасность хранимых данных, поскольку они будут расположены на собственных серверах компании, а представленные выше аналоги не предоставляют возможности хранить данные на стороне заказчика;
- контроль хранимых на сервисе данных, а именно уверенность в том, что данные не используются третьими лицами в рамках маркетинговых кампаний, основанных на приобретенных данных;
- возможность добавления новой функциональности в систему за прогнозируемый срок и затраты, нежели чем при использовании продуктов других компаний.

Глава 2. Построение архитектуры

2.1. Требования к продукту

Начать проектирование сервиса следует с анализа требований к продукту, которые можно разделить на функциональные и нефункциональные.

Функциональные требования заключаются в использовании определенных технологий разработки, в числе которых: программная платформа NodeJS языка программирования JavaScript, язык программирования TypeScript, фреймворк Express, база данных MongoDB, фреймворк Pollux, прокси-сервер NGINX и библиотека Yup для валидации данных JavaScript. Каждая из перечисленных технологий будет подробно описана в главе 3.

Нефункциональные требования включают в себя: высокие показатели расширяемости и масштабируемости с минимальными трудозатратами, действия по работе системы в высоконагруженной среде.

Масштабируемость

Масштабируемость подразумевает способность сервиса справляться с растущим числом пользователей, одновременно взаимодействующих с приложением. Таким образом, масштабируемое приложение должно одинаково хорошо работать как с несколькими, так и с тысячами пользователей и своевременно реагировать на изменения входящего трафика.

Горизонтальное масштабирование позволяет увеличить количество вычислительных ресурсов, доступных системе, путем добавления новых виртуальных машин, которые могут обрабатывать пользовательские запросы, поэтому использование монолитной архитектуры, как было сказано выше, не подходит для разработки.

Вертикальное масштабирование позволяет увеличить эффективность использования вычислительных ресурсов путем увеличения доступных машине ресурсов или оптимизации ПО.

Расширяемость

Расширяемость приложения предполагает использование такой архитектуры, которая позволила бы легко добавлять в приложение новую функциональность без изменения ранее написанного кода, увеличивая возможность повторного использования.

Расширяемое приложение обладает такими характеристиками, как:

- гибкость, определяемая как способность быстро адаптироваться к растущим потребностям и изменяющимся требованиям через расширение существующей и добавление новой функциональности;
- настраиваемость, определяемая как автоматизация повторяющихся задач путем поддержания инструментами, которые могут легко использоваться не программистами;
- возможность простой интеграции, обеспечивающая более прозрачный обмен данными с другими важными для бизнеса внешними сервисами [8].

Действия по работе системы в высоконагруженной среде

Поскольку количество пользователей в системе неизвестно, а предполагаемую нагрузку невозможно оценить точно, необходимо предусмотреть работу сервиса в высоконагруженной среде. В связи с этим разрабатывается многомодульная архитектура с возможностью запуска как в монолитном, так и в распределенном режиме.

2.2. Модули приложения

При выделении модулей приложения необходимо выделить все возможные действия пользователей в разрабатываемом сервисе, объединить их в смысловые группы и выявить зависимости между ними. В результате вышеописанных действий приложение разделено на семь модулей:

1. **AuthorizationService**. Модуль, отвечающий за доступ к приложению (авторизация, аутентификация и прочие функции).
2. **UserService**. Модуль, отвечающий за работу с пользователем (создание, редактирование и поиск).
3. **HRService**. Модуль, отвечающий за функциональность, доступную HR-отделу. В частности, работа с анкетами кандидатов на должность и работа с вакансиями компаниями.
4. **PDPService**. Модуль, отвечающий за работу с персональным планом развития сотрудника компании (создание, редактирование и удаление плана, изменение состояния тех или иных этапов плана, проведение ревью и прочие функции).
5. **SkillService**. Модуль, отвечающий за работу с системой навыков (создание, редактирование и удаление навыков, группировка в папки, поиск).
6. **ActivityService**. Модуль, предоставляющий функциональность взаимодействия с мероприятиями компании (создание, редактирование и удаление, добавление пользователей к мероприятию и прочие функции).
7. **NotificationService**. Модуль, отвечающий за работу с уведомлениями на сервисе (отправление пользователю и прочая функциональность).

Подробнее функциональность приложения и зависимости между модулями показаны на рисунке 1.

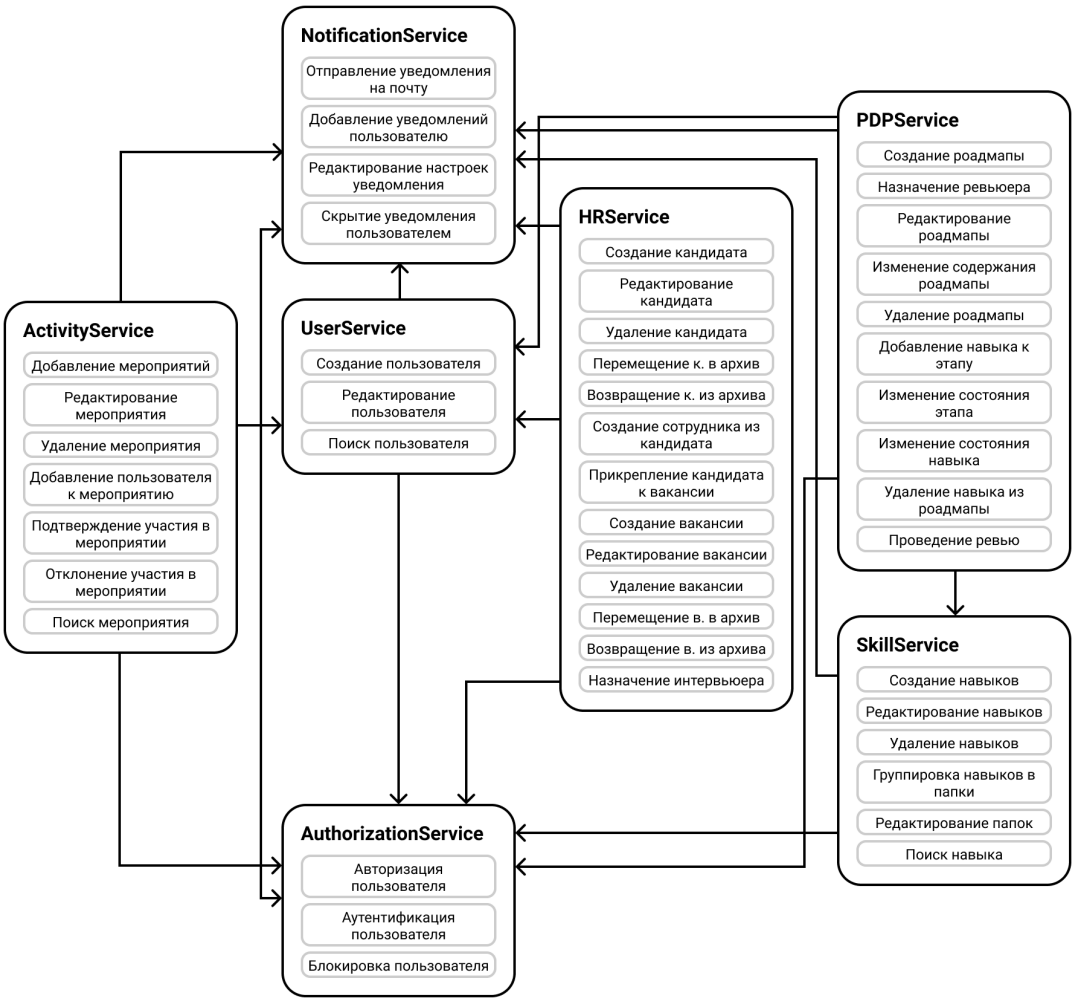

Рис. 1: Архитектура приложения

2.3. Зависимости между модулями

Для того, чтобы быстро и корректно разработать каждый модуль, нужно уже на этапе проектирования проанализировать, от каких сторонних модулей зависит рассматриваемый. Для этого необходимо проанализировать все семь модулей и их функциональность, выявить все возможные зависимости модулей друг от друга и влияние, которое оказывает рассматриваемая функциональность на функциональность других модулей.

Доступ к определенной функциональности модуля есть только у ограниченных классов пользователей. Например, в процессе создания Сотрудника из Кандидата (HRService) необходимо:

1. Обратиться в AuthorizationService для верификации прав пользователя.

2. После создания сотрудника отправить событие в NotificationService.
3. Далее NotificationService обрабатывает событие и принимает решение об уведомлении пользователей системы.

Подробное взаимодействие между функциональностями модулей представлено на рисунке 3.

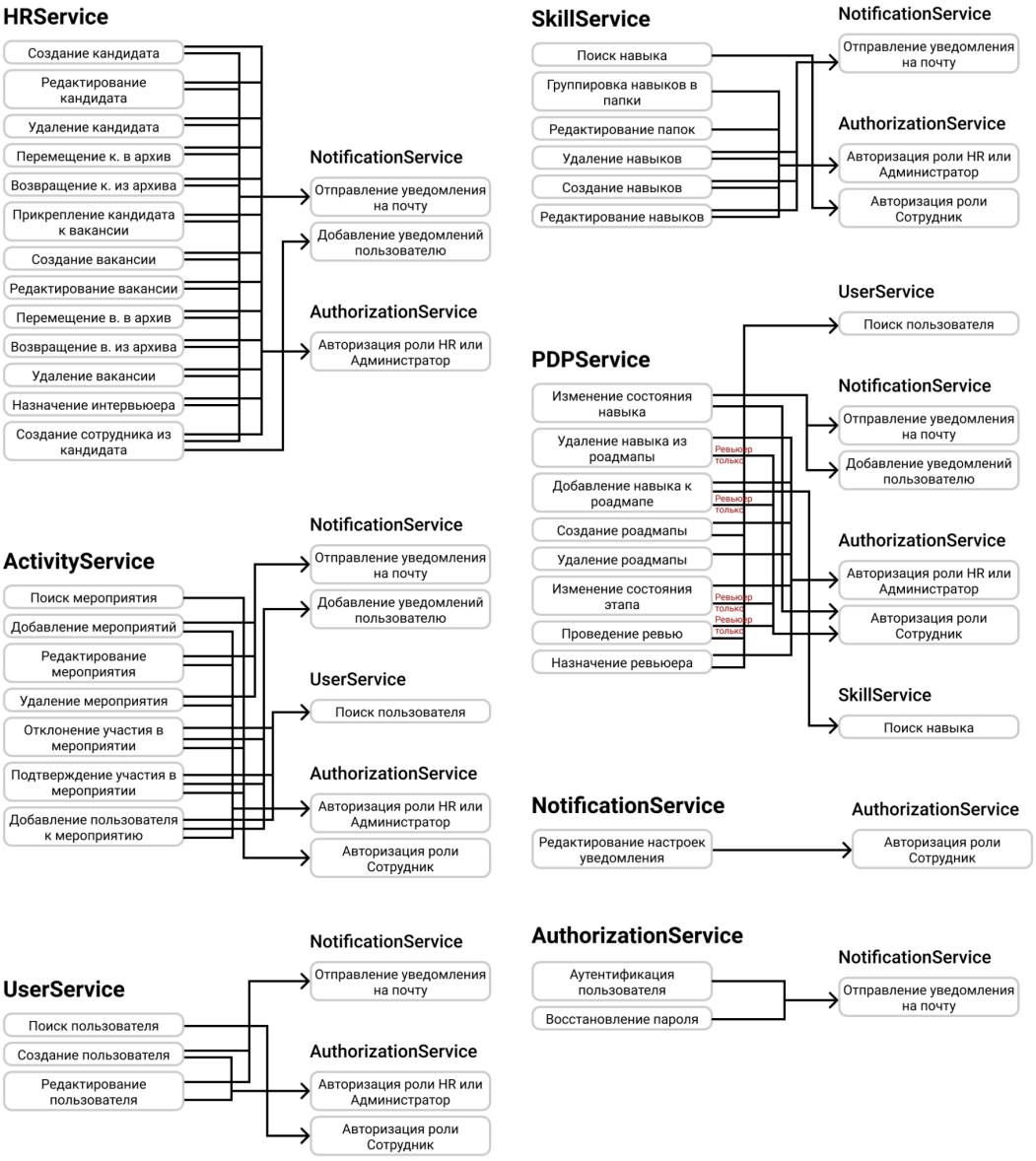

Рис. 2: Зависимости между модулями

2.4. Архитектура серверной части

В основе построения архитектуры серверной части приложения лежит многомодульный подход. Далее структура будет описана подробно.

DB

База данных. Представляет собой организованный набор структурированной в соответствии с указанной схемой информацией.

DAO

Data Access Object (далее по тексту DAO) является промежуточным слоем между базой данных и модулем, отвечающим за передачу запросов в DB и обработку полученных из нее данных [9]. DAO абстрагирует используемую реализацию доступа к данным для Model, обеспечивая прозрачный доступ к источнику данных.

DAO предоставляет относительно простое и строгое разделение двух важных частей приложения, которые могут, но не должны знать ничего друг о друге и которые, как ожидается, будут часто и независимо развиваться. Если же необходимо изменить базовый механизм доступа к данным, меняется только слой DAO, а не все места в логике предметной области, в которых используется DAO.

Model

Компонент, улучшающий показатель расширяемости приложения, так как появляется возможность имплементировать несколько компонентов Model, реализующих в себе разную бизнес-логику, при этом не вызывая изменений в остальных компонентах системы.

Model оперирует с данными с помощью компонента, реализующего интерфейс IDAO, но несет в себе бизнес-логику. Например, к Model можно отнести проверку корректности операции или данных.

Service

Один из основных компонентов, позволяющий реализовать быстрый и низкозатратный переход к микросервисной архитектуре посредством подмены модуля на компонент, реализующий общение с другими сервисами через такие протоколы передачи данных, как HTTP (HyperText Transfer Protocol, далее по тексту HTTP), gRPC (Remote Procedure Calls) и прочие.

Service представляет собой тонкий слой бизнес-логики, инкапсулирующий множество моделей в рамках одного сервиса для остальных компонентов системы. Таким образом, приложение взаимодействует с сервисом через программный код одного компонента и не получает доступа к Model сервиса.

Conductor

Оркестратор [10] верхнего уровня, отвечающий за полноценное взаимодействие всех сервисов: подключение всех сервисов в систему, их запуск и коммуникацию, то есть за внедрение зависимостей между сервисами.

На рисунке 3 продемонстрированы отношения между всеми частями архитектуры.

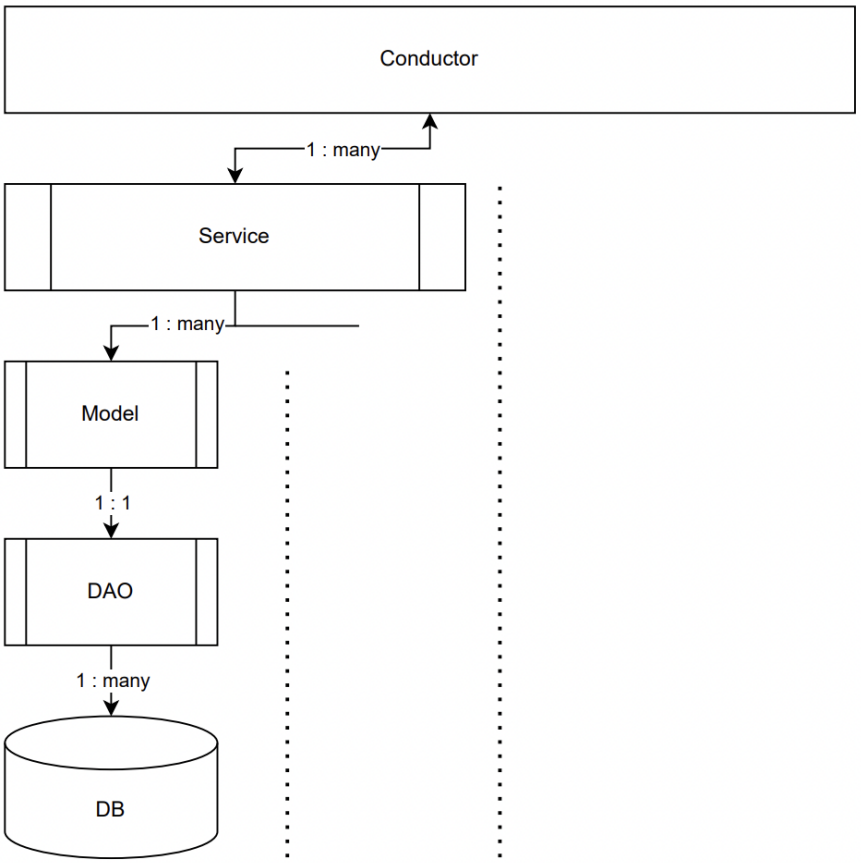

Рис. 3: Архитектура серверной части приложения

Глава 3. Реализация проекта

3.1. Инфраструктура сервиса

Инфраструктура сервиса представлена на рисунке 4. Далее она будет описана более подробно.

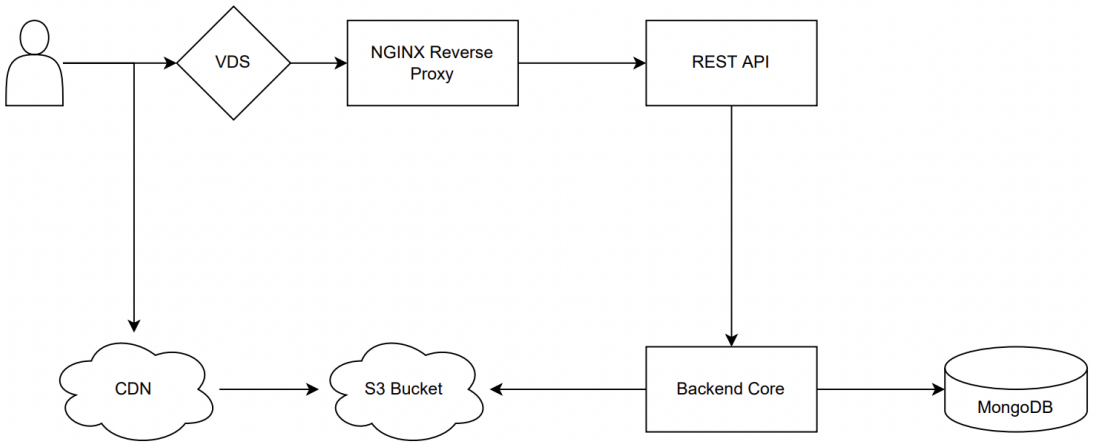

Рис. 4: Инфраструктура сервиса

Виртуальный выделенный сервер

Virtual dedicated server (далее по тексту VDS) относится к облачной технологии «инфраструктура как услуга» (IaaS) и позволяет пользователям арендовать вычислительные ресурсы провайдера услуги для разработки, развертывания и размещения ПО.

VDS доступен пользователю, как операционная система, к которой можно получить удаленный доступ через сеть Интернет. VDS не подразумевает возможность использования выделенных вычислительных ресурсов несколькими арендаторами, что гарантирует устойчивость ПО [11].

NGINX Reverse Proxy

NGINX Reverse Proxy [12] (обратный прокси-сервер) представляет собой тип прокси-сервера, который обычно находится за брандмауэром в частной сети и направляет клиентские запросы на соответствующий внутренний сервер. Обратный прокси-сервер предоставляет возможность дополнительно контролировать сетевой трафик между клиентами и серверами.

NGINX Reverse Proxy может использоваться в различных целях, среди которых: распределение нагрузки на ПО, защита сервера от нежелаемого внешнего воздействия, ускоренная обработка входящих запросов и обеспечение безопасности данных конечного пользователя.

Обратный прокси-сервер может действовать как «регулировщик», распределяя клиентские запросы по группе серверов таким образом, чтобы максимизировать скорость и использование емкости. При этом он гарантирует, что ни один сервер не будет перегружен и производительность останется на номинальном уровне. При выходе сервера из строя, балансировщик нагрузки перенаправляет трафик на менее нагруженные серверы.

Обратный прокси-сервер может сжимать входящие и исходящие данные и кэшировать часто запрашиваемый контент, ускоряя поток трафика между клиентами и серверами. Также он может выполнять дополнительные задачи, такие как шифрование SSL (Secure Sockets Layer), чтобы разгрузить веб-серверы, повышая их производительность [13].

Перехватывая запросы, направляемые на внутренние серверы, обратный прокси-сервер выступает в качестве дополнительной защиты от атак. Это гарантирует доступ к нескольким серверам с одного локатора записей или URL-адреса независимо от структуры локальной сети.

REST API

Representational State Transfer (далее по тексту REST) Application Programming Interface (далее по тексту API) [14] представляет собой компонент, реализующий архитектурный стиль интерфейса API, который использует HTTP-запросы для доступа и использования данных.

Запросы к REST API не имеют состояния, поэтому данный стандарт полезен в облачных приложениях. Компоненты без состояния могут быть повторно развернуты в случае сбоя, а также масштабированы в соответствии с изменениями нагрузки. Так как любой запрос может быть направлен к

любому экземпляру компонента, не может быть сохранено ничего, что должно быть запомнено следующей транзакцией.

REST API компонент подключает Backend Core, и вызывает функции-обработчики, на основе команды HTTP и универсального идентификатора ресурса (URI).

Content delivery network

Content delivery network (далее по тексту CDN) [15] представляет собой сеть серверов, связанных друг с другом с целью максимально быстрой, дешевой, надежной и безопасной доставки контента: HTML-страниц, файлов javascript, таблиц стилей, изображений и видео. Чтобы повысить скорость и возможности подключения, CDN размещает серверы в точках обмена между различными сетями. Также, благодаря своей распределенной структуре CDN может обрабатывать больше трафика и выдерживать сбои оборудования лучше, чем многие исходные серверы.

Благодаря кэшированию и другим оптимизациям CDN могут уменьшить объем данных, которые должен предоставлять исходный сервер, снижая затраты на хостинг.

S3 Bucket

Simple storage service (далее по тексту S3) Bucket [16] представляет собой службу хранения объектов, которая хранит данные в виде объектов в корзинах. Корзина — это контейнер для объектов, которыми могут быть файлы и любые метаданные, описывающие файл. У каждого объекта есть свой ключ, который является уникальным идентификатором объекта в корзине.

S3 предоставляет функции, которые можно настроить для поддержки конкретного варианта использования. Например, используя управление версиями, можно хранить несколько версий объекта в одном сегменте и восстанавливать случайно удаленные или перезаписанные объекты. Стоит отметить, что сегменты и объекты доступны только в том случае, если явно

предоставляется доступ, управление которым осуществляется точкой доступа S3.

Backend Core

Backend Core содержит в себе функции-обработчики, некоторые из которых являются промежуточным программным обеспечением.

В контексте маршрутизации промежуточным ПО является любой код, который выполняется между получением сервером запроса и отправкой ответа. Функции могут изменять объект запроса, запрашивать базу данных или другим способом обрабатывать входящий запрос. Функции промежуточного ПО обычно заканчиваются передачей управления следующей функции промежуточного ПО, а не отправкой ответа. В результате вызывается промежуточная функция, которая завершает цикл запрос-ответ, отправляя ответ HTTP обратно клиенту [17].

MongoDB

MongoDB [18] является документо-ориентированной базой данных, которая хранит данные в документах, подобных JSON (JavaScript Object Notation) (BSON (Binary JavaScript Object Notation)), поэтому поля могут варьироваться от документа к документу, а структура данных может меняться с течением времени. Специальные запросы, индексирование и агрегирование в реальном времени предоставляют эффективные способы доступа к данным и их анализу.

MongoDB по своей сути является распределенной базой данных, поэтому высокая доступность, горизонтальное масштабирование и географическое распределение встроены и просты в использовании.

3.2. Используемые технологии

В главе 2 были перечислены функциональные требования, заключающиеся в использовании определенных технологий при разработке сервиса. В предыдущем разделе были описаны документо-ориентированная

база данных MongoDB и обратный прокси-сервер NGINX. Далее будут описаны другие технологии.

NodeJS

Представляет собой программную платформу языка программирования JavaScript, расширяющую базовые возможности JavaScript. NodeJS [19] предоставляет множество преимуществ, которые будут перечислены далее.

- Расширенный API взаимодействия с операционной системой. NodeJS добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, написанный на C++, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода.
- Поддержка асинхронных операций. Несмотря на то, что JavaScript является однопоточным языком, с помощью механизмов Event Loop и Promise можно параллельно ожидать несколько операций, не блокируя при этом исполнение других.
- Продвинутая система пакетов. По умолчанию NodeJS использует для контроля зависимостей пакетный менеджер node package manager (далее по тексту npm). npm является открытым регистром пакетов с продвинутой системой версионирования и рекурсивной поддержкой зависимостей. Благодаря большому сообществу программистов, использующих JavaScript, npm содержит много пакетов, которые можно использовать в разработке.
- Возможность горизонтального масштабирования системы. NodeJS предоставляет возможность на программном уровне создавать дочерние процессы для исполнения отдельного JavaScript кода, что позволяет при необходимости реализовать механизмы балансировки внутри приложения без необходимости изменения инфраструктуры.

TypeScript

Является языком программирования, расширяющим возможности JavaScript и предоставляющим дополнительные возможности, которые отсутствуют в JavaScript:

- статическая типизация, позволяющая четко обозначить тип переменной и добавить конкретную структуру какому-либо элементу;
- расширенная поддержка принципов объектно-ориентированного программирования (далее по тексту ООП), заключающаяся в предоставлении синтаксических структур, схожих с ООП-языками (Interface, Generic Types, Type Inference, Enums, Namespaces) и позволяющими интегрировать такие принципы как DIP (Dependency Inversion Principle), а также более четко структурировать программный код, написанный на JavaScript;
- транспиляция кода в JavaScript на этапе запуска, позволяющая на этапе исполнения использовать преимущества JavaScript, а во время разработки избавиться от его недостатков;
- благодаря жесткой архитектуре TypeScript [20] ускоряется тестирование ПО.

Express

Фреймворк в Node.js для создания HTTP-серверов, предоставляющий возможности маршрутизации и промежуточных слоев обработки запросов. Express поддерживает все HTTP-методы [21].

Pollux

Pollux [22] — фреймворк, который является надстройкой над Express и предоставляет следующую функциональность:

- удобную валидацию данных,
- гибкую структуру формирования запросов,
- работу с базой данных,

- отслеживание состояния приложения,
- подключение конфигурационных данных окружения,
- реализацию паттернов pubsub, registry, config,
- возможности распределенного запуска сервисов.

Yup

Представляет собой библиотеку в npm, предназначенную для анализа и валидации данных. Использование Yup [23] предоставляет разработчику ряд возможностей, представленных далее.

- Интерфейс схемы предназначен для моделирования как простых, так и сложных моделей данных.
- Использование вместе с TypeScript позволяет убедиться, что схема правильно реализует тип.
- Yup поддерживает асинхронные проверки как на стороне сервера, так и на стороне клиента.
- Позволяет добавлять собственные методы и схемы валидации.
- Предоставляет подробные сведения об ошибках, благодаря которым отладка становится значительно легче.

3.3. Методы тестирования

Unit-тестирование

При тестировании приложения используется покрытие кода unit-тестами. Их задача — подтвердить, что поведение модулей (наименьших изолированных элементов) программы соответствует ожиданиям. Unit-тестами покрываются элементы программы, которые не несут в себе бизнес-логики, а только участвуют в более крупных и сложных элементах программы.

При грамотном построении unit-тестов, успешное прохождение их модулем позволяет предположить, что модуль с большой долей вероятности не будет причиной ошибки в других элементах программы. Данное суждение позволяет упростить процесс дальнейшей разработки и тестирования, так как

отсутствует необходимость в тестировании модуля в тех местах, где он используется.

Для реализации unit-тестов в проекте используется фреймворк Jest [24]. Jest обладает рядом преимуществ по сравнению со своими аналогами. Они приведены далее.

- Не требует первичной конфигурации.
- Изолированность и многопоточность. Каждый тест запускается в отдельном JavaScript-процессе, что значительно увеличивает скорость и уровень надежности тестирования.
- Мощная система обработки ошибок. При неуспешном прохождении теста модулем Jest четко описывает, по какой причине это произошло.
- Jest позволяет понять, какой процент кодовой базы протестирован. Данная метрика позволяет описать уровень надежности кода с системной точки зрения и понять, какие модули требует особого внимания при ручном тестировании.
- Расширенный API. Фреймворк предоставляет объемный интерфейс взаимодействия, что упрощает процесс миграции с аналогов на Jest.

Volume testing

Volume testing (объемное тестирование) входит в состав нагрузочного тестирования и проводится на больших объемах данных. Оно проводится с целью проверки производительности системы при увеличении количества данных в базе данных. Оно помогает узнать следующее о приложении:

- время отклика системы при определенном количестве данных и запросов;
- объем данных, соответствующих нарушению стабильности системы.

3.4. Полученные результаты

В результате разработан полноценный сервис, который можно запускать как в монолитном, так и в распределенном вариантах в зависимости от потребностей системы.

Проведено unit-тестирование наиболее значимых функций. Например, тест test(‘Collect endpoints in correct order’, () => {}) проверяет функцию collectEndpoints(), которая рекурсивно проходит по файловой структуре проекта, собирая все объявленные в ней запросы. При разработке проекта маршрутизация отображается через файловую структуру. Файлы в этом случае организованы так же, как и конечные точки в приложении. При тестировании нужно было убедиться, что конечные точки собираются корректно и правильно внедряются в систему. Например, если в структуре присутствуют файл users/new и файл users/:id, то необходимо, чтобы первый файл подключался перед следующим, так как он объявляет статический маршрут, а второй – динамический маршрут.

Далее представлена вышеописанная функция теста.

```
test('Collects endpoints in correct order', () => {
  const endpoints = collectEndpoints(path.join(__dirname, '..',
'__mocks__'));

  const routes = endpoints.map(({ route }) => route);

  expect(routes[0]).toBe('/');
  expect(routes[1]).toBe('/sample');
  expect(routes[2]).toBe('/:slug1');
  expect(routes[3]).toBe('/:slug1/sample');
  expect(routes[4]).toBe('/:slug1/:slug2');
  expect(routes[5]).toBe('/:slug1/:slug2/sample');
});
```

Объемное тестирование проводилось с целью получения среднего значения отклика системы под разной нагрузкой. Для этого:

- запускалось 1000 запросов на получение данных пользователя;

- в качестве критерия нагрузки системы использовалось определенное количество параллельно запущенных процессов, которые совершают обход дерева навыков в ширину, запрашивая для каждой вершины, которую они встречают пути к корню этого дерева;
- считалось среднее время выполнения одного запроса на получение данных пользователя.

На рисунке 6 показан график зависимости среднего времени отклика системы на получение данных пользователя от критерия нагрузки в монолитном и распределенном вариантах запуска.

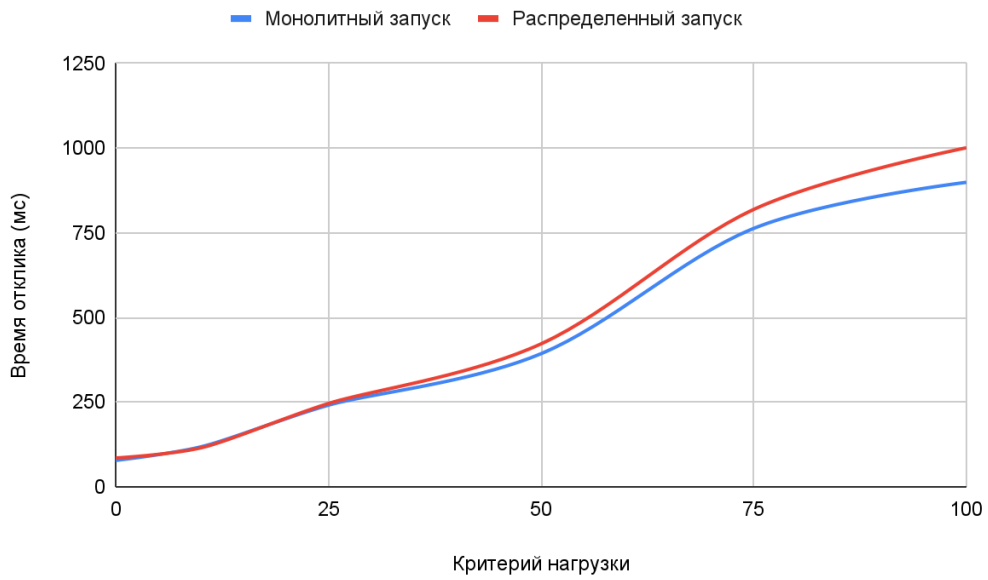

Рис. 6: График зависимости среднего времени отклика системы от критерия нагрузки

График показывает поведение системы на увеличенных показателях нагрузки. Как видно из графика, время отклика системы в монолитном запуске ожидаемо меньше, чем в распределенном. Это связано с тем, что в распределенном запуске имеется необходимость передавать данные по протоколу HTTP, что в свою очередь несет определенные издержки для каждого запроса.

На графике можно заметить зависимость роста, близкую к линейной. Отсюда можно сделать вывод, что поведение системы предсказуемо при средних и близких к высоким уровням трафика. Таким образом, при увеличении нагрузки в два раза можно прогнозировать среднее время отклика системы на получение какого-либо запроса. Это позволяет контролировать нагрузку на сервис с помощью дополнительного ПО, которое будет отслеживать уровень трафика и при необходимости распределять его для конечных точек, время отклика которых не соответствует требованиям. В случае экстремальных нагрузок можно масштабировать всю систему, что позволит привести время отклика к номинальным значениям.

Для того чтобы сравнить, какое количество запросов успевает обработать система за секунду при различной нагрузке, было подсчитано общее время выполнения запросов на получение данных пользователя одновременно с обходом дерева и получением пути от узла к вершине и их количество. Затем количество запросов делилось на время выполнения.

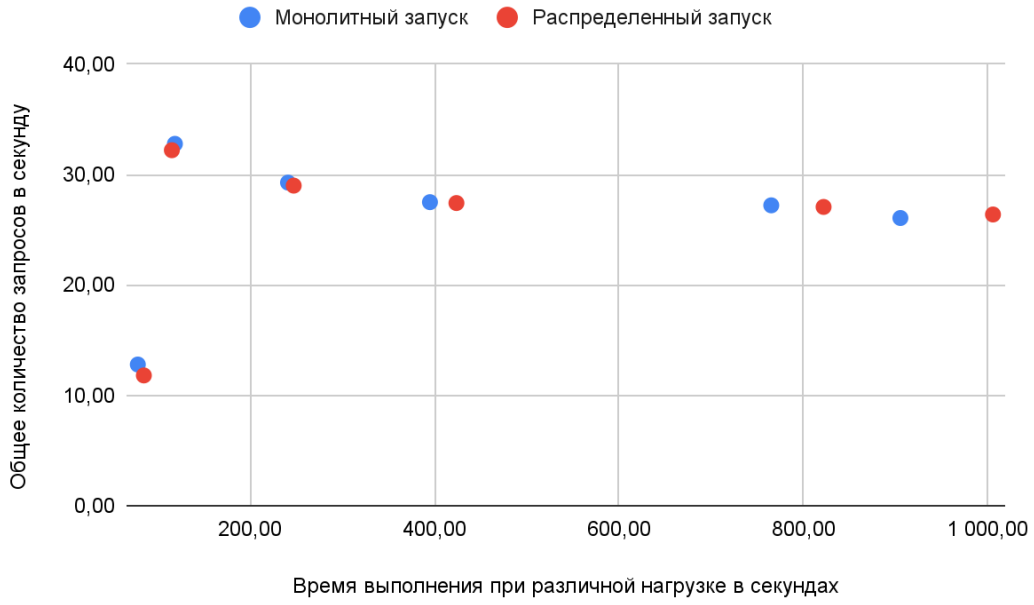

Рис. 7: Общее количество запросов в секунду при различном времени выполнения

На диаграмме, показанной на рисунке 7, видно, что при увеличении параллельно запущенных процессов время выполнения тестирования в целом

быстрее в монолитном запуске. Количество обработанных запросов в секунду в двух вариантах запуска отличается незначительно и сначала увеличивается, а затем плавно уменьшается, достигая предельного значения, равного 26 запросам. Это связано с тем, что в зависимости от ресурсов машины, на которой запускается система, есть предельное значение запросов, которое она может обработать. Ограничение можно устранить путем горизонтального масштабирования.

На диаграмме, показанной на рисунке 8, отображены целевые запросы на получение данных пользователя. Можно увидеть, что при 75 параллельно запущенных процессах (пятые точки диаграммы) система успевает обрабатывать не более одного полного запроса.

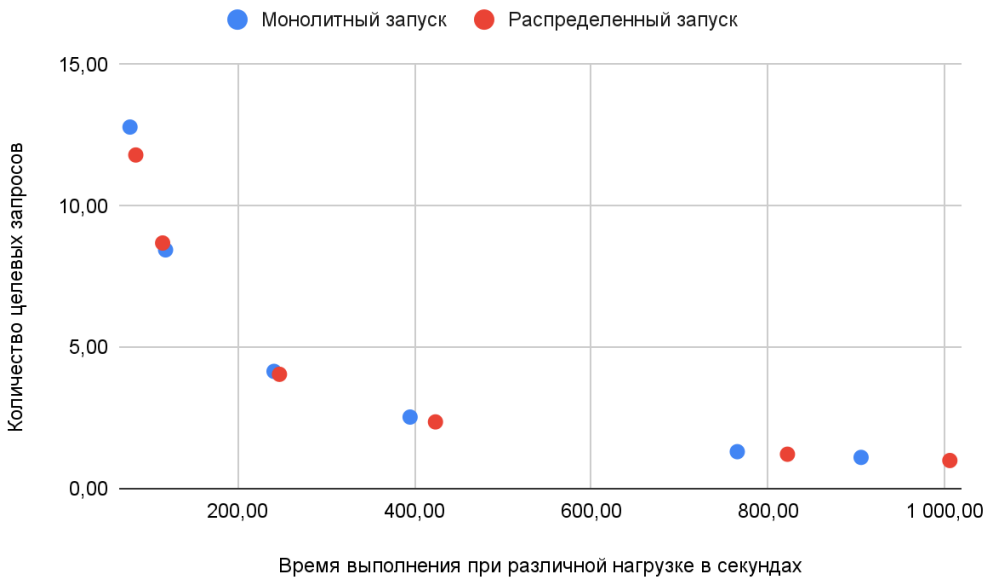

Рис. 8: Количество запросов на получение данных пользователя в секунду при различном времени выполнения

Выводы

Система разработана с использованием многомодульной архитектуры, поэтому добавление новой функциональности может быть получено за прогнозируемый срок и стоимость. Это значительно быстрее и дешевле, чем внедрение новых модулей в продукт третьей стороны, поскольку оно делается по запросу большого количества пользователей или по индивидуальному заказу.

Сервис разработан с возможностью распределенного запуска с учетом необходимых критериев масштабируемости. Он может быть масштабирован в любой его части. Если со временем нагрузка при использовании монолитного запуска будет увеличиваться, можно будет запустить приложение в распределенном режиме с учетом нефункциональных требований. В этом случае ПО будет масштабироваться контролируемо и предсказуемо.

Требования к безопасности данных были учтены и реализованы должным образом, так как все данные хранятся на собственных серверах компании, которые защищены в рамках инфраструктуры заказчика.

Заключение

В рамках работы разработан полноценный HR-инструмент, который может успешно использоваться для решения поставленных задач. Архитектура серверной части удовлетворяет требованиям к высоким показателям расширяемости приложения. Так как система может быть запущена с использованием как монолитного запуска, так и распределенного, она справится с растущим числом пользователей с помощью горизонтального и вертикального масштабирования.

В настоящий момент разработанное решение связано с HR нуждами, однако спроектированная архитектура позволяет добавлять новые модули, не связанные с HR, выращивая единую экосистему компании, чего в полной мере не могут дать конкуренты.

Список используемой литературы

[1] Work Institute. 2020 Retention Report: Insights on 2019 Turnover Trends, Reasons, Costs & Recommendations // Work Institute. – 2020. – с. 35. – URL: https://info.workinstitute.com/en/retention-report-2020 (дата обращения: 10.12.2021).

[2] Pattern: Monolithic Architecture // microservices.io – URL: https://microservices.io/patterns/monolithic.html (дата обращения: 18.04.2022).

[3] Крис Ричардсон. Микросервисы. Паттерны разработки и рефакторинга / Крис Ричардсон – СПб: Питер, 2019.

[4] Martin L. Abbott. The Art of scalability: scalable web architecture, processes, and organizations for the modern enterprise / Martin L. Abbott, Michael T. Fisher. – Boston: Pearson Education, Inc., 2009.

[5] С. Ньюмэн. Создание Микросервисов / С. Ньюмэн – СПб: Питер, 2016.

[6] Teachbase // teachbase.ru – URL: https://teachbase.ru/vozmozhnosti/ (дата обращения: 20.04.2022)

[7] ispringlearn // ispring.ru – URL: https://www.ispring.ru/ispring-learn/key-features (дата обращения: 20.04.2022)

[8] What is Extensibility? // propelplm.com – URL: https://www.propelplm.com/articles/what-is-extensibility (дата обращения: 20.04.2022)

[9] Data Access Object Pattern // geeksforgeeks.org – URL: https://www.geeksforgeeks.org/data-access-object-pattern/ (дата обращения: 27.04.2022)

[10] Orchestration Pattern // medium.org – URL: https://medium.com/gbtech/orchestration-pattern-3d8f5abc3be3 (дата обращения: 27.04.2022)

[11] Virtual Dedicated Server (VDS) // technopedia.com – URL: https://www.techopedia.com/definition/26808/virtual-dedicated-server-vds (дата обращения: 27.04.2022)

[12] What Is a Reverse Proxy Server? // nginx.com – URL: https://www.nginx.com/resources/glossary/reverse-proxy-server/ (дата обращения: 29.04.2022)
[13] Compression and Decompression // nginx.com – URL: https://docs.nginx.com/nginx/admin-guide/web-server/compression/ (дата обращения: 29.04.2022)
[14] What is a REST API? // redhat.com – URL: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата обращения: 27.04.2022)
[15] What is a CDN? | How do CDNs work? // cloudfare.com – URL: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/ (дата обращения: 30.04.2022)
[16] What is Amazon S3? // amazon.com – URL: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html (дата обращения: 30.04.2022)
[17] Back-End Web Architecture // codeacademy.com – URL: https://www.codecademy.com/article/back-end-architecture (дата обращения: 01.05.2022)
[18] MongoDB Documentation // mongodb.com – URL: https://www.mongodb.com/docs/manual/tutorial/getting-started/ (дата обращения: 03.05.2022)
[19] Node.js v18.2.0 documentation // nodejs.org – URL: https://nodejs.org/api/documentation.html (дата обращения: 25.03.2022)
[20] Compiler Options // typescriptlang.org – URL: https://www.typescriptlang.org/tsconfig (дата обращения: 10.04.2022)
[21] 4.x API // expressjs.com - URL: https://expressjs.com/en/4x/api.html (дата обращения: 15.04.2022)
[22] @glangeo/pollux // npmjs.com – URL: https://www.npmjs.com/package/@glangeo/pollux (дата обращения: 20.05.2022)

[23] yup // npmjs.com – URL: https://www.npmjs.com/package/yup (дата обращения: 15.04.2022)
[24] Expect // jestjs.io – URL: https://jestjs.io/docs/expect (дата обращения: 17.05.2022)