



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Институт (Филиал): № 8 «Компьютерные науки и прикладная математика»

Кафедра: 806

Группа: М8О-408Б-20

Направление подготовки: 01.03.02 Прикладная математика и информатика

Профиль: Информатика

Квалификация: бакалавр

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему: «Разработка сервера CRM-системы с применением микросервисной архитектуры»

Автор ВКРБ: Попов Матвей Романович ()

Руководитель: Лукин Владимир Николаевич ()

Консультант: ()

Консультант: ()

Рецензент: ()

К защите допустить

Заведующий кафедрой № 806 «Вычислительная математика
и программирование» Крылов Сергей Сергеевич ()

Москва 2024

РЕФЕРАТ

Выпускная квалификационная работа бакалавра состоит из 47 страниц, 7 рисунков, 10 использованных источников, 2 приложений.

CRM, СЕРВИС, СИСТЕМА, ПРИЛОЖЕНИЕ, СЕРВЕР, КЛИЕНТ, КОМПАНИЯ, СОТРУДНИК, ОБЪЯВЛЕНИЕ, СДЕЛКА

Объектом разработки в данной работе является CRM-система, рассчитанная на использование в сфере B2B.

Цель работы – создание CRM-системы с удобным функционалом для взаимодействия компаний различных размеров.

Для достижения поставленной цели были проведены исследования функционала и предназначения существующих CRM-систем. Основное содержание работы состояло в разработке CRM-системы, рассчитанной на использование в B2B-сфере.

Основной результат работы — сервер CRM-системы, созданный по принципам микросервисной архитектуры, рассчитанный на горизонтальное масштабирование и высокую надежность.

Данные результаты разработки предназначены для использования компаниями крупного, среднего и малого бизнесов, которые поставляют свои товары и услуги другим компаниям.

Применение результатов данной работы позволяет компаниям удобно взаимодействовать с заказчиками и продавать свои товары и услуги.

СОДЕРЖАНИЕ

РЕФЕРАТ.....	2
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ	4
ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ	5
ВВЕДЕНИЕ	6
1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ	10
1.1 Компания	10
1.2 Сотрудник.....	11
1.3 Контакт	12
1.4 Объявление.....	12
1.5 Сделка	13
1.6 Уведомление	15
2 АРХИТЕКТУРА ПРОЕКТА	17
2.1 Монолитная и микросервисная архитектуры	17
2.2 Чистая архитектура.....	18
2.3 Используемые технологии.....	22
2.4 Взаимодействие между сервисами	27
2.5 Авторизация и аутентификация	30
2.6 Сервисы CRM-системы.....	32
3 ОПИСАНИЕ РАБОТЫ.....	40
3.1 Реализованная архитектура	40
3.2 Взаимодействие с клиентом	40
ЗАКЛЮЧЕНИЕ.....	44
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	45
ПРИЛОЖЕНИЕ А Схема архитектуры проекта.....	46
ПРИЛОЖЕНИЕ Б QR-код репозитория с исходным кодом	47

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

В настоящей выпускной квалификационной работе бакалавра применяются следующие термины с соответствующими определениями:

- Веб-сервис – программная система, имеющая стандартизированный интерфейс и веб-адрес
- Веб-сервер – программное обеспечение, принимающее запросы от клиентов и выдающее ответы на них
- Бэкенд – внутренняя часть сайта или приложения, скрытая от пользователя
- Фронтенд – публичная часть веб-сервисов, с которой пользователь взаимодействует напрямую
- Веб-сервис – программная система, имеющая стандартизированный интерфейс и веб-адрес
- Веб-сервер – программное обеспечение, принимающее запросы от клиентов и выдающее ответы на них
- Бэкенд – внутренняя часть сайта или приложения, скрытая от пользователя
- Фронтенд – публичная часть веб-сервисов, с которой пользователь взаимодействует напрямую

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей выпускной квалификационной работе бакалавра применяются следующие сокращения и обозначения:

CRM	– Customer Relationship Management, или управление взаимоотношениями с клиентами
B2B	– Business-to-business, или «бизнес для бизнеса»
СУБД	– система управления базами данных
HTTP	– HyperText Transfer Protocol, или протокол передачи гипертекста
RPC	– Remote Procedure Calls, или удаленный вызов процедур
BLOB	– Binary Large Object, или двоичный большой объект
ACID	– Atomicity, Consistency, Isolation, Durability, или атомарность, согласованность, изоляция и устойчивость
SQL	– Structured Query Language, или язык структурированных запросов
ОС	– операционная система
VM	– Virtual Machine, или виртуальная машина
JSON	– Javascript Object Notation, или текстовый формат обмена данными, основанный на JavaScript
REST	– Representational State Transfer, или передача состояния представления

ВВЕДЕНИЕ

С возникновением информационного общества у компаний появились новые стандарты ведения взаимоотношений с клиентами. Для максимизации прибыли необходимо поддерживать высокую лояльность своих клиентов, как новых, так и постоянных. С ростом клиентской базы эта задача становится сложной, решить ее помогают CRM-системы. Первые CRM-системы, возникшие в 90-х годах прошлого века, предоставляли базовый функционал управления различной информацией о клиентах. За годы своего существования CRM существенно преобразились: появились различные аналитические функции, например оценка результативности сотрудников компании, появилось такое понятие как сделка, и различные ее источники. Современные CRM-системы являются инструментом не только сбора, хранения и анализа информации о клиентах, но и непосредственно связи с клиентами. На сегодняшний день способов для клиента установить контакт с компанией может быть очень много.

Для любого пользователя сегодня существует множество различных социальных сетей, мессенджеров, маркетплейсов и прочих площадок различного предназначения. У каждой такой платформы своя аудитория, и если компания хочет продавать свой продукт на широкую аудиторию, она должна быть представлена на каждой из наиболее популярных платформ. Но такое распространение значительно увеличивает работу сотрудников по связи с клиентами, нужно отслеживать поступающие от клиентов сообщения сразу из нескольких платформ. Им на помощь приходят современные CRM-системы, каждая из которых способна отслеживать сообщения, поступающие в аккаунты компании на разных площадках, и перенаправлять их в одно место, собственно, в саму CRM. Затем на обращение клиента в CRM создается сделка. Сотрудник, обслуживающий клиента, может оставлять различные заметки в сделке, переводить ее на определенный этап, в зависимости от степени завершенности. Это позволяет безболезненно сменить ответственного за сделку на другого сотрудника, который легко извлечет из нее всю необходимую информацию.

Такая модель позволяет не только эффективно работать с клиентом, но и грамотно оценить результативность работы сотрудников.

Таким образом, благодаря системе сделок в компании остается некоторая информация о каждом клиенте, которая при повторном его обращении поможет сотрудникам обеспечить высокий уровень персонализации, из которого следует высокая удовлетворенность клиента услугами компании.

Такая модель уже доказала свою жизнеспособность. Подобными CRM-системами давно пользуются тысячи компаний по всему миру. Их внедрение автоматизировано и интуитивно понятно. Но у всех них есть весьма неочевидный недостаток — источник сделки. Когда клиент отправляет сообщение кому-либо в социальных сетях или в мессенджерах, это сообщение должно пройти немалый путь, прежде чем попасть к адресату. В случае использования CRM-системы этот путь увеличивается на несколько этапов. Транспорт сообщений внутри CRM зачастую занимает не меньше времени и ресурсов, чем транспорт сообщений внутри мессенджеров. В связи с этим неизбежно возникает угроза потери некоторой доли сообщений. Критической эта угроза становится, когда известной компании ежедневно поступает несколько тысяч сообщений от клиентов. В таком случае каждый день компания будет терять лояльность как минимум нескольких клиентов, чьи сообщения останутся без ответа. Особенно страшно, если будет утерян постоянный клиент, о котором в компании уже собрана информация. Необходимо отметить, что вне зависимости от CRM существует множество факторов, из-за которых часть обращений клиентов может остаться без ответа, но крайне нежелательно, чтобы использование CRM увеличивало эту часть. Сложностей разработчикам CRM-систем добавляет и то, что все подобные платформы регулярно получают обновления, а значит нужно своевременно обновлять и CRM-систему, чтобы она работала с актуальными версиями всех социальных сетей и мессенджеров.

CRM-системы наиболее актуальны в том случае, когда клиенты компании — физические лица, закрывающие свои собственные потребности. Такие

компании могут быть разными по размеру (большой, малый бизнес), отрасли (развлечения, туризм, медицина, сфера красоты) и так далее. Однако существуют компании, которые выпускают товар, удовлетворяющий потребностям других компаний, и неприменимый для физических лиц. Эти товары могут относиться к самым разным отраслям и покрывать нужды различных заведений, магазинов и даже целых городов. Зачастую объявления с такими товарами нельзя найти в социальных сетях и на других распространенных площадках, поэтому для производящих их компаний не подойдут CRM, основная задача которых — доставлять сообщения из внешних ресурсов.

Цель работы — создать CRM-систему, подходящую для компаний, которые производят специфические товары. Эта CRM-система должна содержать площадку объявлений для таких товаров и не быть подвержена потерям сообщений.

В работе решены следующие задачи:

- реализован базовый функционал CRM-системы (отношения сущностей компании, сотрудника и клиента);
- разработана площадка для объявлений;
- разработана система сделок между компанией и клиентами;
- разработана система рейтинга компаний;
- разработана система уведомлений.

Основные результаты:

- создан бэкенд CRM-системы, включающей в себя рейтинг компании;
- создана площадка для объявлений, где будут продаваться товары, необходимые для деятельности компаний из разных сегментов бизнеса;
- CRM и площадка объявлений объединены в одну систему, что исключает какие-либо конфликты между ними;
- решение реализовано с применением микросервисной архитектуры.

Реализованную систему было решено назвать BRM, что означает «Business Relationship Management», или управление взаимоотношениями бизнесов.

Использование результатов работы позволит компаниям более эффективно находить и обслуживать компании, которым необходимы оптовые поставки либо поставки специфических товаров.

1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Необходимо разработать CRM-систему, предлагающую функционал площадки объявлений. Основными сущностями приложения будут:

- компания;
- сотрудник;
- контакт;
- объявление;
- сделка.

В компании может быть несколько сотрудников. Каждый сотрудник может размещать объявления от лица своей компании, просматривать объявления других компаний и откликаться на них. При отклике на объявление создается сделка, к ней имеют доступ сотрудники компании, на чье объявление откликнулись. Сделка представляет собой процесс согласования деталей предоставления услуги между потребителем и поставщиком. Успешное закрытие сделки означает, что обе компании пришли к соглашению, и поставщик обеспечил потребителя услугой в соответствии со всеми договоренностями.

1.1 Компания

Любая компания имеет следующие характеристики:

- название;
- описание;
- отрасль;
- рейтинг;
- дата регистрации.

Название и описание задается владельцем компании, только владелец компании может их изменить. Отрасль владелец выбирает из списка доступных.

Рейтинг компании определяется из того, сколько сделок и на какую сумму закрыла компания. У только что созданной компании рейтинг равен 4. Максимально возможный рейтинг равен 5. Рейтинг компании могут просматривать все пользователи BRM. Таким образом, если пользователь найдет на площадке привлекательное объявление, он может понять по рейтингу компании, стоит ли ему откликаться на такое предложение. Профиль любой компании доступен всем зарегистрированным пользователям BRM, даже тем, кто к этой компании не принадлежит.

Новые компании создаются при регистрации их владельцев в BRM. Регистрация компании происходит одновременно с регистрацией владельца, это исключает вероятность того, что будет зарегистрирована компания без владельца либо сотрудник без компании.

1.2 Сотрудник

Сотрудник — олицетворение пользователя BRM. Имеет следующие характеристики:

- компания;
- фамилия;
- имя;
- почтовый адрес;
- занимаемая должность;
- отдел компании;
- дата регистрации.

Сотрудник компании может создавать объявления от лица своей компании, откликаться на объявления других компаний и создавать сделки. Сотрудник, назначенный ответственным за какие-либо сущности (объявления или сделки), может изменять информацию, хранящуюся в этих сущностях, а также передавать ответственность другому сотруднику своей компании. При этом сотрудник не может изменить информацию о себе, например фамилию

или имя. Сделать это может только владелец компании, в которой работает этот сотрудник.

Владелец компании — сотрудник, выполняющий функции ответственного за компанию и за всех ее сотрудников. Он точно так же может редактировать информацию о компании и сотрудниках, а может передать владение компанией другому своему сотруднику. Только владелец может добавлять новых сотрудников в свою компанию. Рядовой сотрудник не проходит ту же процедуру регистрации, что и владелец компании. Вместо этого владелец регистрирует сотрудника в уже созданной, находящейся под его ответственностью компанией, и передает логин (в качестве него используется корпоративная почта) и пароль физическому сотруднику.

1.3 Контакт

Контакты есть у каждого сотрудника, зарегистрированного в BRM. Контакт имеет следующие характеристики:

- владелец контакта;
- сотрудник, которого добавили в список контактов;
- заметки;
- дата создания.

Сотрудник может добавлять в контакты как сотрудников своей компании, так и сотрудников, которые, например, откликнулись на его объявление или наоборот, указаны в чужом объявлении. В заметках сотрудник может указать любую нужную ему информацию о контакте. Разумеется, список контактов, в том числе и заметки, недоступны другим сотрудникам, даже владельцу компании.

1.4 Объявление

В разрабатываемой CRM-системе присутствует полноценная площадка для размещения объявлений. Каждое объявление имеет следующие характеристики:

- компания;
- название;
- текст;
- отрасль;
- стоимость;
- сотрудник, создавший объявление;
- сотрудник, ответственный за объявление;
- дата создания.

Каждый пользователь BRM имеет доступ к площадке объявлений. Можно найти объявление по названию или тексту, можно фильтровать объявления из поисковой выдачи (по компании, разместившей объявление или по отрасли), можно отсортировать список объявлений по возрастанию/убыванию стоимости или даты создания. Ответственный за объявление, как и владелец компании, может изменить название, текст, стоимость и отрасль, причем отрасль объявления может не совпадать с отраслью компании, от имени которой размещено объявление. Ответственный за объявление и владелец компании могут передать ответственность другому сотруднику своей компании. У компании сохраняются отклики на объявления, на которые откликались сотрудники этой компании.

1.5 Сделка

Сделка — одна из ключевых сущностей любой современной CRM. Имеет следующие характеристики:

- объявление, на основе которого создана;
- название;
- описание;
- стоимость;
- статус;
- ответственный;

- компания-поставщик;
- сотрудник-клиент;
- компания сотрудника-клиента;
- дата создания.

Когда сотрудник откликается на объявление, в его компании появляется отклик на это объявление, а в компании, которая разместила это объявление, появляется сделка. Таким образом, отклик на объявления — источник сделки в BRM. Детали сделки не видны компании-клиенту, ими управляет компания-поставщик. Создание сделки — автоматический процесс, ответственным за новую сделку назначается тот же сотрудник, который является ответственным за объявление, на основе которого появилась сделка. В дальнейшем этот сотрудник может либо передать ответственность, либо самостоятельно работать над этой сделкой, в том числе изменить название (по умолчанию подставляется название компании, откликнувшейся на объявление), описание, стоимость и статус. Существует несколько статусов сделки:

- Новая сделка — начальный статус сделки, говорит о том, что сотрудники еще не рассматривали созданную сделку;
- Установка контакта — ответственный за сделку связывается с клиентом;
- Обсуждение деталей — контакт установлен, обе стороны договариваются о различных ситуациях, возникновение которых возможно в процессе выполнения сделки;
- Заключительные детали — большая часть сделки со стороны поставщика выполнена, стороны обсуждают завершающие нюансы, например сроки оплаты и поставки;
- Завершено — заключительный этап, означает, что обе стороны выполнили свои обязательства и сделка закрыта;
- Отклонено — заключительный этап, на который попадают сделки, в которых на каком-либо этапе стороны не смогли прийти к соглашению.

Сотрудники могут просматривать сделки своей компании. Доступна группировка сделок по статусу и по ответственному. Примечательно, что созданную сделку, в отличие от объявления, нельзя удалить, а можно только перевести в один из заключительных этапов — «Завершено» или «Отклонено». При закрытии сделки, то есть при переводе ее в этап «Завершено», и при подтверждении закрытия сделки со стороны клиента, компания получает увеличение рейтинга.

1.6 Уведомление

В любом приложении возникает необходимость уведомлять пользователя о каких-либо событиях. В BRM существует два типа уведомлений:

- уведомление о новой сделке;
- уведомление о закрытой сделке.

Вне зависимости от типа, у каждого уведомления есть следующие характеристики:

- компания;
- дата;
- состояние (просмотрено/не просмотрено).

В BRM все уведомления общие на всю компанию. Их может просматривать любой сотрудник, свежие уведомления имеют статус «Не просмотрено», при просмотре уведомления сотрудником статус меняется на противоположный.

Уведомление о новой сделке приходит в компанию вместе с новой сделкой, когда на объявление компании откликается клиент. Задача этого уведомления — обратить внимание сотрудников компании на то, что появилась новая сделка. Помимо общей информации, это уведомление содержит новую сделку и компанию, откликнувшуюся на объявление.

Уведомление о закрытой сделке имеет более важное значение. Как было отмечено, рейтинг компании растет, если она закрывает сделки со своими клиентами. Но тогда компания может сразу же переводить все новые сделки по своим объявлениям в статус «Завершено», даже не связываясь с клиентом, тем самым искусственно увеличивать свой рейтинг. Уведомление о закрытой сделке существует, чтобы предотвратить такие случаи. Оно приходит компании-клиенту, когда компания-поставщик переводит сделку с этим клиентом в состояние «Завершено». В меню уведомления компания-клиент может либо подтвердить закрытие сделки, то есть поставщик честно отработал и заслуживает повышения рейтинга, либо не подтвердить, и тогда поставщика ждет понижение рейтинга за мошеннические действия.

2 АРХИТЕКТУРА ПРОЕКТА

2.1 Монолитная и микросервисная архитектуры

Во время зарождения веб-сервисов многие из них строились по принципу монолитной архитектуры. Она подразумевает, что приложение не разделено на части, существует единственная точка входа. У монолитной архитектуры есть неоспоримые преимущества:

- простота разработки и проектирования — не нужно заботиться о том, что приложение состоит из нескольких частей, все сущности находятся в одном месте и тесно связаны между собой;
- легче отслеживать взаимодействия сущностей между собой, так как они существуют в пределах одной программы;
- сведение сетевых взаимодействий к минимуму, а значит отсутствие задержек.

В течение 2000-х годов этот подход хорошо себя зарекомендовал при разработке различных веб-приложений, как в России, так и за рубежом. Однако шло время, популярность сервисов, работающих на монолитах, росла, а вместе с ней росла и нагрузка на сервисы. Появились сложности с тем, чтобы справляться с возросшей нагрузкой. Популярным и наиболее очевидным выходом из ситуации стало вертикальное масштабирование — увеличение вычислительной мощности одной отдельно взятой машины, на которой запущен веб-сервер. Действительно, чем мощнее сервер, тем больше запросов он может обработать в единицу времени.

Однако бесконечно растить мощность одного сервера невозможно. Чтобы качественно решить проблему, необходимо увеличить количество вычислительных узлов — применить горизонтальное масштабирование. Тут на помощь приходит микросервисная архитектура — подход, при котором приложение разделяется на несколько независимых программ, которые работают отдельно друг от друга и взаимодействуют друг с другом через различные сетевые протоколы. Многие компании столкнулись с

необходимостью разделять монолиты на отдельные микросервисы, чтобы иметь возможность запускать приложение с аналогичным функционалом на нескольких машинах. Помимо возросшей в несколько раз производительности, у микросервисной архитектуры обнаружили ряд других неочевидных преимуществ:

- меньшая чувствительность к сбоям — при отказе одного из сервисов остальная часть приложения не утратит работоспособности;
- улучшенная масштабируемость команды разработки.

Стоит отметить, что микросервисная архитектура — не панацея, ее уместно применять в приложениях с высокой нагрузкой, но монолиты все еще имеют право на существование, особенно в маленьких проектах с низкой нагрузкой. С этим связаны недостатки микросервисов:

- сложность проектирования — зачастую бывает проблематично разделить одно приложение на несколько частей;
- сложность отладки и тестирования, так как при нескольких сервисах сложнее отслеживать взаимодействие сущностей между собой;
- возросшие накладные расходы на сетевое взаимодействие между сервисами.

Таким образом, микросервисная архитектура хорошо подходит для приложений, которые в перспективе должны выдерживать большое число запросов от клиентов, в таком случае трудности в проектировании и тестировании будут оправданы.

2.2 Чистая архитектура

В микросервисной архитектуре каждый сервис является самостоятельной программой, выполняющей определенный спектр задач. Для их работы могут использоваться различные внешние зависимости: базы данных, очереди сообщений, другие сервисы, общение с которыми может происходить с помощью различных протоколов. В процессе эксплуатации

приложения может возникнуть необходимость заменить один или несколько его компонентов, например поменять СУБД с MySQL на PostgreSQL. Чтобы эти изменения происходили максимально безболезненно, то есть затрагивали как можно меньше частей сервиса, принято использовать чистую архитектуру.

Чистая архитектура — подход, решающий некоторые проблемы, которые возникают при длительном процессе разработки приложения. Основная идея чистой архитектуры — разделение приложения на слои:

- слой сущностей (entities);
- слой бизнес-логики (usecases);
- слой интерфейсов (interfaces);
- слой инфраструктуры (infrastructure).

На рисунке 1 продемонстрирована схема чистой архитектуры в виде концентрических окружностей [1].



Рисунок 1 — Схема чистой архитектуры

Сущности — наиболее важные компоненты приложения. Именно они представляют реальные объекты в бизнес-процессах и отвечают за основные операции с данными. Слой сущностей находится в самом центре приложения и не имеет никакой информации о более внешних слоях.

Слой бизнес-логики описывает конкретные сценарии использования приложения, именно в нем определяется, какие операции выполняются над сущностями и как они выполняются. С сущностями этот слой взаимодействует напрямую, а с внешней инфраструктурой — через интерфейсы.

Слой интерфейсов содержит описания бизнес-логики и инфраструктуры в виде непосредственно интерфейсов. Это промежуточный слой между бизнес-логикой и внешними зависимостями, он обеспечивает их взаимодействие и независимость друг от друга. Благодаря ему слой бизнес-логики не имеет информации об инфраструктуре, а слой инфраструктуры ничего не знает о бизнес-логике. Все, что требуется от бизнес-логики и инфраструктуры — удовлетворять своим интерфейсам и взаимодействовать друг с другом исключительно через эти интерфейсы. Таким образом, использование интерфейсов является ключевым в разработке приложения по принципам чистой архитектуры.

Слой инфраструктуры отвечает за взаимодействие приложения с внешним миром, включая базы данных и различные сетевые сервисы. Благодаря слою интерфейсов любой компонент внешней инфраструктуры можно заменить на другой, главное, чтобы новый компонент удовлетворял уже существующему интерфейсу. Это условие гарантирует то, что замена одного или нескольких компонентов никак не повлияет на работу бизнес-логики, так как ее слой взаимодействует не с инфраструктурой напрямую, а через интерфейсы. Инфраструктура оперирует с бизнес-логикой через интерфейсы, что обеспечивает изолированность этих слоев друг от друга.

Изображение чистой архитектуры в виде круговой диаграммы имеет очень важный смысл. Чтобы представить, как выполняется клиентский запрос к сервису, нужно провести диаметр на диаграмме — это и будет путь запроса. Сначала из инфраструктуры приходит запрос, затем обработчик запроса вызывает метод из интерфейса, который реализует бизнес-логика. В этом методе выполняются процессы бизнес-логики и используются сущности.

Вполне возможно, что для выполнения метода бизнес-логике потребуется взаимодействовать с инфраструктурой, например с базой данных. Это взаимодействие происходит через интерфейс клиента базы данных. Таким образом, чистая архитектура пройдена насквозь, причем слой инфраструктуры и слой бизнес-логики ни разу не взаимодействовали напрямую.

В разрабатываемой CRM все микросервисы реализованы по правилам чистой архитектуры, все они состоят из определенных частей:

- `model` — содержит описания сущностей, соответствует слою сущностей;
- `app` — содержит интерфейс (соответствует слою интерфейсов) и реализацию приложения (относится к слою бизнес-логики);
- `ports` — содержит описание серверов приложений (HTTP либо gRPC), с помощью которых другие сервисы вызывают методы этого приложения, относится к слою инфраструктуры;
- `adapters` — содержит интерфейс (соответствует слою интерфейсов) и реализацию клиента других сервисов для взаимодействия с ними (относится к слою инфраструктуры);
- `hero` — содержит интерфейс клиента базы данных (соответствует слою интерфейса) и его реализацию (относится к слою инфраструктуры).

Использование чистой архитектуры позволило отделить бизнес-логику приложения от внешних зависимостей, что позволит в будущем тратить меньше времени при замене одних компонентов приложения на другие. Помимо этого, разбиение приложения на слои значительно упрощает тестирование и поиск ошибок в программе, так как теперь можно отдельно протестировать каждый слой.

Таким образом, чистая архитектура обеспечивает простоту изменения и поддержки продукта. Чистая архитектура подразумевает независимость приложения от каких-либо фреймворков или различных внешних

интерфейсов. Она не зависит от языка программирования и может быть применима к любым технологиям.

2.3 Используемые технологии

Весь бэкенд реализован на языке программирования Golang с применением микросервисной архитектуры. В качестве хранилища данных используется PostgreSQL, в качестве инструмента для кеширования используется Redis. Во время разработки для оперативного развертывания различных компонентов приложения использовалась система контейнеризации Docker.

2.3.1 Golang

Golang — высокоуровневый компилируемый язык программирования. Разрабатывается компанией Google. Был представлен в 2009 году, первая официальная версия вышла в 2012 году. Синтаксически язык похож на C, однако имеет ряд существенных отличий:

- наличие сборщика мусора;
- отсутствие неявного преобразования типов;
- отсутствие арифметики указателей;
- функции могут иметь несколько возвращаемых значений.

Помимо этого, Go имеет ряд существенных особенностей, которые его сильно выделяют на фоне других языков:

- работа с многопоточностью;
- обработка ошибок;
- работа в ООП-парадигме.

На каждой из этих особенностей стоит остановиться поподробнее.

Работа с многопоточностью: в отличие от C и C++, в которых разработчик вынужден напрямую взаимодействовать с системными потоками, Go имеет собственный планировщик и горутины — легковесные потоки с

меньшим размером стека, переключение между которыми занимает значительно меньше времени, чем у системных потоков. Помимо них в языке есть множество средств синхронизации, которые хоть и основаны на системных сущностях, но имеют достаточно высокоуровневое описание, благодаря чему с ними удобно работать. Стоит отметить, что Go — кроссплатформенный язык, а значит разработка многопоточных приложений не отличается для Linux и Windows, несмотря на то что многопоточность в этих системах устроена по-разному.

Обработка ошибок: в Golang нет механизма обработки исключений по типу try-catch. Вместо него в Go используется тип error. Переменную этого типа может возвращать функция, если во время ее выполнения произошла ошибка. Затем разработчик может в зависимости от типа ошибки определить дальнейшее выполнение программы. Ошибки можно объединять в цепочки, которые потом можно разворачивать и подробно просматривать проблемные места кода.

Работа в ООП-парадигме: объектно-ориентированное программирование реализовано совсем не так, как в C++ и других подобных языках. В Go есть структуры, у структуры могут быть поля и методы. Можно создавать экземпляры структур, но в языке нет таких понятий, как конструктор и деструктор, нет перегрузки операторов, по-другому устроены наследование и полиморфизм. В качестве аналога наследования используется встраивание, при котором одна структура становится как бы подмножеством другой, и можно неявно обращаться к ее публичным полям и методам из структуры-наследника. В Go есть интерфейсы, но не нужно явно указывать, что конкретная структура реализует конкретный интерфейс. Чтобы структура реализовывала интерфейс, необходимо, чтобы структура имела все методы, характерные для интерфейса — так называемая «утиная типизация».

Все эти особенности дают понять, что Golang — язык, в который намеренно не стали добавлять большое число средств, повторяющих функциональность друг друга. Хотя на Go и существуют проекты, имеющие огромную сложную инфраструктуру, например Docker и Kubernetes, основное применение язык получил в бэкенд-разработке, в частности в разработке микросервисов. С теми же целями этот язык использовался при создании CRM-системы.

2.3.2 PostgreSQL

Сохранение данных пользователя — наиболее ответственная задача любого сервиса, для которой существует огромное количество СУБД самого разнообразного предназначения. Зачастую используются реляционные СУБД, где информация хранится в виде связанных таблиц. PostgreSQL — одна из наиболее распространенных реляционных СУБД — была выпущена в 1996 году, она кроссплатформенная, с открытым исходным кодом. За годы своей эксплуатации PostgreSQL зарекомендовала себя как надежное и прекрасно отлаженное решение. Важно то, что она поддерживает все необходимые для проекта типы данных:

- целые числа;
- вещественные числа;
- символьные типы произвольной длины;
- булев тип;
- дата и время;
- BLOB.

Немаловажно и то, что PostgreSQL удовлетворяет всем принципам ACID:

- атомарность — любая транзакция будет либо закончена полностью, либо не выполнена совсем, не существует промежуточного состояния;

- согласованность — никакие транзакции не допустят противоречия данных;
- изоляция — несколько параллельных транзакций не оказывают влияния друг на друга;
- устойчивость — любые законченные изменения сохраняются даже при возникновении аварии на аппаратном уровне.

Соблюдение этих свойств — гарантия того, что СУБД способна сохранять и корректно обрабатывать сохраненную информацию. Это крайне важно, ведь информация пользователей — наиболее ценное, что может быть в CRM-системе.

2.3.3 Redis

Без постоянного хранилища данных не обойтись, однако в современных высоконагруженных приложениях обращение к базе данных с точки зрения быстродействия программы — узкое место. Если на один HTTP-запрос со стороны клиента приходится 4-5 SQL-запросов к базе данных, временные задержки могут стать существенными, что плохо скажется на быстродействии приложения. Необходимо уметь сохранять полученную единожды информацию из основной СУБД во временное хранилище, чтобы новый запрос этих же данных не занимал столько же времени. Для этих целей принято использовать Redis — NoSQL СУБД, сохраняющую данные в формате «ключ-значение», которая была выпущена в 2009 году. Redis располагает свои хранилища в оперативной памяти, которая на несколько порядков быстрее, чем жесткие диски и даже SSD. В них и располагает свои таблицы PostgreSQL. Примечательной особенностью Redis является еще и то, что любой добавляемой записи можно задать время существования, по истечении которого запись будет удалена из хранилища. Этот функционал идеально подходит для создания временного хранилища данных.

Справедливости ради стоит отметить, что на организации временных хранилищ возможности Redis далеко не исчерпываются. Эта СУБД обладает

механизмами журналирования, что позволяет сохранять данные из оперативной памяти в постоянную, а значит использовать ее в качестве постоянного хранилища. Она поддерживает master-slave репликацию, с помощью которой можно создавать распределенные системы хранения данных. Redis даже предоставляет операции для реализации обмена сообщениями в формате «издатель – подписчик», что позволяет использовать его как очередь сообщений.

2.3.4 Docker

Любое современное сколь угодно сложное приложение состоит из нескольких компонентов. Особенно это актуально для микросервисной архитектуры, так как проект состоит из нескольких изолированных частей. Это вносит некоторые сложности при разработке приложения, например для фронтенд-разработчика крайне важно иметь у себя на компьютере запущенное приложение, чтобы отслеживать внесенные изменения.

Воссоздание на локальном компьютере того же окружения, которое задумывал бэкенд-разработчик — самое очевидное решение проблемы. Необходимо создать все базы данных, создать таблицы в них, установить все библиотеки, используемые в сервисной части приложения. У этого подхода есть ряд существенных недостатков. Фронтендеру придется каждый раз по отдельности запускать каждый из микросервисов, а их может быть несколько десятков. Критическим является то, что некоторые компоненты могут вообще не существовать для операционной системы, используемой фронтенд-разработчиком. Например, Redis не имеет поддержки ОС Windows.

Решить последний недостаток призвана VM — виртуальная машина. Это программа, которая запускает внутри себя отличную от основной операционную систему, которая использует виртуальные ресурсы. VM строго ставит в соответствие виртуальным ресурсам физические. Этот подход решает проблему несовместимости приложения для разных ОС, однако возникает

новая проблема — ухудшение производительности. Запуск двух полноценных ОС на одной машине требует существенного потребления ресурсов, что негативно сказывается на быстродействии приложения, к тому же запуск приложения внутри изолированной VM ограничивает фронтенд-разработчику сетевое взаимодействие с ним.

Docker — программное обеспечение для контейнеризации приложений, выпущено в 2013 году, поддерживается и обновляется до сих пор. Контейнеризация существенно отличается от виртуализации: у контейнеров нет строгого сопоставления виртуальных ресурсов физическим, они не требуют наличия собственной полноценной ОС. поэтому они потребляют значительно меньше ресурсов и запускаются на порядок быстрее, нежели VM. Контейнеризация идеально подходит для микросервисной архитектуры. Каждый компонент приложения можно запустить в отдельном контейнере, будь то СУБД, очередь сообщений или сервис. Ключевая особенность — возможность сконфигурировать запуск нескольких контейнеров одновременно. Это позволит разработчику не запускать каждый сервис вручную, а одной командой запустить сразу все необходимые контейнеры. Таким образом, Docker значительно облегчает разработку микросервисного приложения, как серверной его части, так и клиентской.

2.4 Взаимодействие между сервисами

При разделении на микросервисы неизбежно возникает задача обеспечить их коммуникацию по сети. Существует множество сетевых протоколов, способных решить эту задачу.

2.4.1 HTTP и REST API

HTTP — протокол прикладного уровня, за время своего существования ставший стандартом во взаимодействии между различными узлами Всемирной паутины. HTTP подразумевает общение в формате «запрос-ответ». HTTP-запрос состоит из следующих частей:

- версия протокола;
- тело запроса — в нем передается основная информация, необходимая для выполнения запроса, чаще всего в формате JSON;
- метод — HTTP поддерживает множество методов, наиболее используемые из них это GET, PUT, POST, DELETE;
- заголовки — содержат в основном дополнительную информацию, например токен авторизации или формат тела запроса;
- адрес, на который совершается запрос. при этом в адресе также может содержаться необходимая для запроса информация, например query-параметры.

Ответ на HTTP-запрос состоит из следующих частей:

- версия протокола;
- заголовки — аналогично с запросом;
- тело ответа — содержит основную часть информации, запрашиваемой клиентом;
- код ответа.

Код ответа сигнализирует клиенту о результате выполнения его запроса. Если запрос выполнен корректно клиент получит код ответа 200. Если запрос невозможно выполнить из-за ошибки со стороны клиента, например, поиск несуществующей информации, некорректные входные данные, отсутствие авторизации, будет получен один из 400-х кодов. Если ошибка произошла на стороне сервера, клиент получит код 500.

В разрабатываемой CRM HTTP используется для коммуникации между бэкендом и фронтендом, которая происходит в стиле REST API.

REST (Representational State Transfer) — архитектурный подход, применяемый при разработке приложений, работающих с HTTP. Накладывает ряд обязательных условий, которым приложение должно соответствовать:

- клиент-серверное взаимодействие — существует HTTP-сервер, который принимает запросы от множества клиентов, и отвечает на них;

- отсутствие состояний — клиентские запросы не влияют на внутреннее состояние сервера, они могут изменять информацию, хранящуюся в базах данных, но не поведение сервера;
- многоуровневая система — от клиента скрыто, взаимодействует ли он с самим сервером, или с каким-либо промежуточным узлом, например прокси-сервером или балансировщиком;
- кэширование — клиенты имеют возможность кэшировать ответы на некоторые запросы;
- единообразие интерфейса — сервер предоставляет своим клиентам строгую схему взаимодействия, которая содержит форматы поддерживаемых запросов и ответов на них, а клиенты обязаны соблюдать эту схему.

Стоит отметить, что принятый в HTTP формат запросов, содержащий адрес и метод, существует не просто так. В адресе должно содержаться название сущности, с которой взаимодействует клиент, а метод должен обозначать то, какую операцию стоит проделать с этой сущностью. Соблюдение этого правила, а также всех требований REST, способствует созданию удобного и надежного интерфейса.

2.4.2 gRPC

gRPC (Remote Procedure Calls) — технология удаленного вызова процедур, разработана Google в 2015 году. Представляет собой способ сетевого межпроцессорного взаимодействия, который существенно отличается от REST API. Использует протокол HTTP/2, имеющий несколько существенных отличий от HTTP/1:

- бинарный протокол, а не текстовый;
- сжатие заголовков;
- мультиплексирование — позволяет клиенту и серверу разбивать запросы на несколько кадров, чередовать, а затем собирать в правильном порядке;

- `server push` — возможность сервера отправлять несколько ответов на один клиентский запрос.

Использование gRPC на стороне клиента напоминает традиционный вызов функций, как если бы они выполнялись не удаленно, а на той же машине. С одной стороны это снимает некоторые ограничения REST-архитектуры, с другой возникает необходимость в кодогенерации. Поддерживаемые сущности и методы gRPC-сервера описаны в специальном файле на языке определения интерфейса Protobuf. Инструменты генерации кода для gRPC-серверов и клиентов существуют для всех распространенных языков программирования, разработчику лишь необходимо создать реализацию для сгенерированного интерфейса. Как и в случае с HTTP, клиент и сервер могут быть написаны на разных языках.

В разрабатываемой CRM-системе gRPC используется как средство взаимодействия сервисов друг с другом.

2.5 Авторизация и аутентификация

Когда перед приложением стоит задача обслуживания большого числа пользователей и предоставления им возможностей по управлению принадлежащей им информации, возникает необходимость в защите данных одного пользователя от посягательств других. Эту задачу решают авторизация и аутентификация.

Аутентификация — процесс проверки личности пользователя и выдача ему определенных прав доступа в зависимости от результатов проверки. Авторизация определяет, имеет ли пользователь с определенными правами доступ к каким-либо операциям над информацией. Аутентификация и авторизация работают вместе, существует несколько способов их реализации.

Наиболее удобным способом обеспечить авторизацию и аутентификацию является токенизация. Токен — ключ, содержащий данные для авторизации, выдается при успешном прохождении аутентификации, то

есть когда пользователь вводит верные логин и пароль. Токен может быть либо ключом, указывающим на информацию о пользователе в базе данных, либо внутри себя же содержать всю необходимую для авторизации информацию в зашифрованном виде.

JWT — стандарт создания токенов, содержащих информацию в зашифрованном виде. Имеет механизм подписи: если есть субъект, который выдает токены, он должен их подписывать определенным ключом. Субъект, расшифровывающий токены, обладает этим же ключом, и с помощью него он определяет, что токен имеет верную подпись, а затем извлекает из него зашифрованные данные. Если токен подписан неверно, запрос не выполняется. Этот механизм обеспечивает защиту от злоумышленников, так как без секретного ключа невозможно создать подписанный токен, а ключ хранится только на стороне сервера и никуда не передается ни в каком виде.

Для обеспечения большей безопасности каждый токен обладает еще одной характеристикой — временем истечения. Это время, после которого токен считается невалидным и больше не принимается сервером. Чтобы обновлять истекший токен, используется пара токенов, состоящая из access- и refresh-токенов. Access-токен используется в качестве основного: именно в нем содержится зашифрованная информация о пользователе, и он передается в заголовке запроса, требующего авторизацию, но он обладает малым временем истечения. Refresh-токен используется, когда истекает access-токен: выполняется запрос в сервис аутентификации, в нем передается refresh-токен, в ответ клиент получает новую пару токенов. У refresh-токена есть время истечения, обычно от нескольких дней до месяца, в то время как у access-токена это время составляет несколько часов. Если истекли оба токена из пары, пользователю придется повторно ввести логин и пароль, но это произойдет только в том случае, если пользователь не воспользуется CRM-системой весьма продолжительное время.

2.6 Сервисы CRM-системы

Серверная часть BRM состоит из 9 сервисов. 4 из них нужны для управления основными сущностями, еще 5 выполняют задачи, связанные с транспортом, авторизацией и сбором статистики.

2.6.1 core-сервис

Ключевой сервис BRM. Управляет базой данных компаний, сотрудников и контактов. Предоставляет gRPC-сервер, поддерживающий следующие операции:

- получение компании по ID;
- регистрация компании и ее владельца;
- редактирование информации о компании;
- удаление компании;
- получение списка доступных отраслей;
- регистрация сотрудника;
- редактирование информации о сотруднике;
- удаление сотрудника;
- получение списка сотрудников компании с фильтрацией и пагинацией;
- поиск сотрудника по фрагменту фамилии/имени;
- получение сотрудника по ID;
- создание контакта;
- редактирование информации о контакте;
- удаление контакта;
- получение списка контактов с пагинацией;
- получение контакта по ID.

База данных сервиса, схема которой продемонстрирована на рисунке 2, состоит из 4-х таблиц. Каждая сущность, представленная сервисом, имеет свой ID.

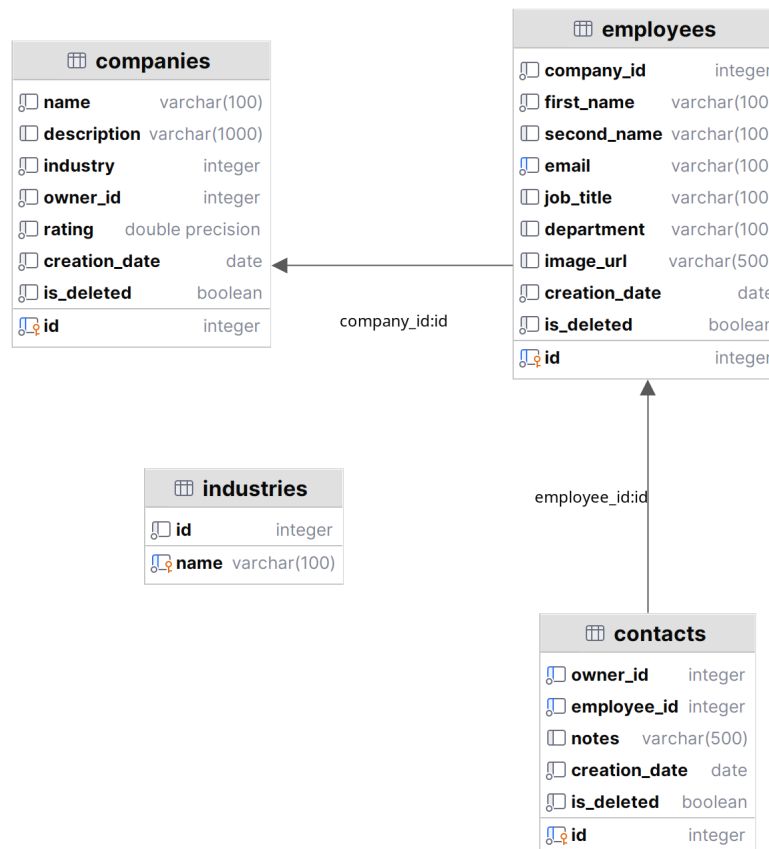


Рисунок 2 — Схема базы данных core-сервиса

2.6.2 Сервис объявлений

Сервис, управляющий базой данных объявлений. Предоставляет gRPC-сервер, поддерживающий следующие операции:

- получение объявления по ID;
- получение списка объявлений с фильтрацией, пагинацией, сортировкой, возможностью поиска по названию и тексту объявлений;
- создание объявления;
- редактирование информации об объявлении;
- удаление объявления;
- создание отклика на объявление;
- просмотр откликов на объявления;
- получение списка поддерживаемых отраслей.

При отклике на объявления сервис взаимодействует с сервисом сделок, чтобы создать новую сделку. База данных объявлений, схема которой

продемонстрирована на рисунке 3, состоит из 3-х таблиц и ссылается на объекты из базы данных core-сервиса.

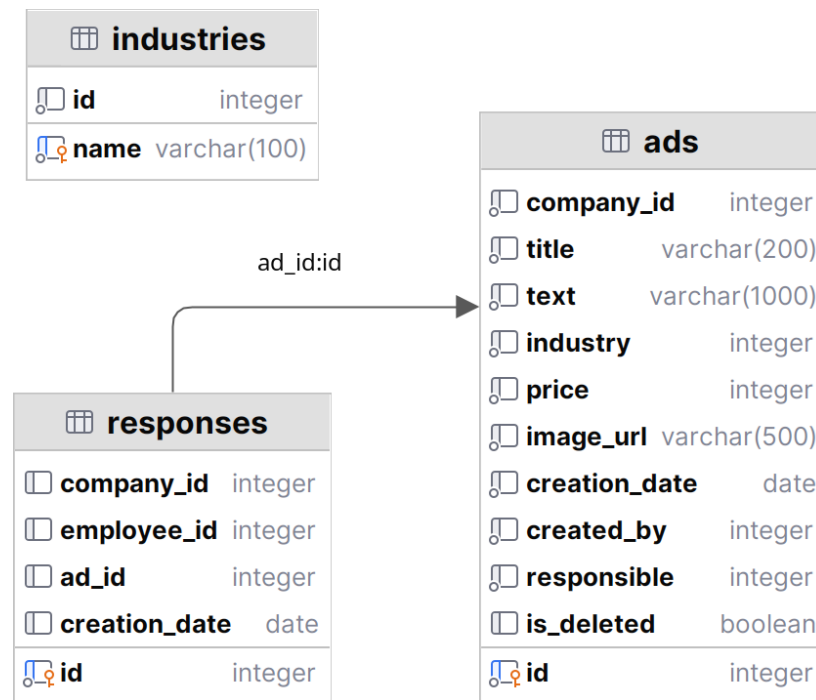


Рисунок 3 — Схема базы данных сервиса объявлений

2.6.3 Сервис сделок

Сервис, управляющий базой данных сделок. Предоставляет gRPC-сервер, поддерживающий следующие операции:

- создание сделки;
- редактирование информации о сделке;
- получение списка сделок с фильтрацией и пагинацией;
- получение сделки по ID;
- получение списка этапов сделки.

Этот сервис использует методы сервиса объявлений, чтобы получить информацию об объявлении для создания новой сделки, а также методы сервиса уведомлений, чтобы создавать уведомления о новых и закрытых сделках.

База данных сервиса, схема которой продемонстрирована на рисунке 4, состоит из 2-х таблиц и ссылается на объекты баз данных core-сервиса и сервиса объявлений.

leads	
ad_id	integer
title	varchar(200)
description	varchar(1000)
price	integer
status	integer
responsible	integer
company_id	integer
client_company	integer
client_employee	integer
creation_date	date
is_deleted	boolean
id	integer

statuses	
id	integer
name	varchar(100)

Рисунок 4 — Схема базы данных сервиса сделок

2.6.4 Сервис уведомлений

Сервис, управляющий уведомлениями в CRM, в том числе реализующий логику подтверждения закрытых сделок. Предоставляет gRPC-сервер, поддерживающий следующие операции:

- создание уведомления;
- получение уведомления по ID;
- получение списка уведомлений с фильтрацией и пагинацией;
- подтверждение закрытия сделки.

Метод создания уведомления использует сервис сделок, при подтверждении закрытия сделки сервис уведомлений использует метод сервиса статистики, чтобы изменить рейтинг компании.

База данных уведомлений, схема которой продемонстрирована на рисунке 5, состоит из 3-х таблиц: 2 таблицы для каждого из типов уведомлений и 1 таблица для общих полей.

notifications	
company_id	integer
date	date
viewed	boolean
type	varchar(100)
id	integer

new_lead_notifications	
id	integer
lead_id	integer
client_company	integer

closed_lead_notifications	
id	integer
ad_id	integer
producer_company	integer
answered	boolean

Рисунок 5 — Схема базы данных сервиса уведомлений

2.6.5 Сервис аутентификации

Сервис, обеспечивающий процесс аутентификации пользователей. Использует базу данных паролей пользователей (все пароли хранятся в зашифрованном виде) и хранилище для refresh-токенов. В этом же сервисе происходит создание и обновление токенов. Предоставляет HTTP-сервер, поддерживающий следующие операции:

- обновление пары токенов;
- получение токенов с помощью логина и пароля;
- удаление пары токенов (выход пользователя из аккаунта).

Пароли всех новых сотрудников попадают в этот сервис. Для этого есть gRPC-сервер, поддерживающий следующие операции:

- регистрация сотрудника;

- удаление сотрудника.

На рисунке 6 продемонстрирована схема базы данных паролей.







 passwords	
 password	varchar(500)
 employee_id	integer
 company_id	integer
  email	varchar(100)

Рисунок 6 — Схема базы данных паролей

2.6.6 API-сервис

Сервис, с которым непосредственно взаимодействует фронтенд по протоколу HTTP. Взаимодействует с core-сервисом, сервисами сделок, объявлений, уведомлений и статистики с помощью gRPC. Предоставляет HTTP-сервер, поддерживающий следующие операции:

- получение информации о компании;
- получение статистики о компании;
- обновление информации о компании;
- удаление компании;
- получение информации о сотруднике;
- получение списка сотрудников;
- обновление информации о сотруднике;
- удаление сотрудника;
- добавление сотрудника в контакты;
- получение контакта;
- получение списка контактов;
- обновление информации о контакте;

- удаление контакта;
- добавление объявления;
- получение объявления;
- получение списка объявлений;
- обновление объявления;
- удаление объявления;
- создание отклика на объявление;
- получение списка откликов;
- получение сделки;
- получение списка сделок;
- обновление сделки;
- получение уведомления;
- получение списка уведомлений;
- подтверждение закрытия сделки по уведомлению.

Для совершения всех этих операций необходима авторизация, то есть наличие access-токена в заголовке к запросу. Сервис корректно обрабатывает все ошибки, связанные с авторизацией, получением несуществующих данных, некорректным вводом и в случае чего возвращает соответствующий код ошибки и текстовое описание. Все запросы, связанные с получением списка каких-либо сущностей, поддерживают пагинацию и фильтрацию по различным признакам.

2.6.7 Прочие сервисы

Помимо перечисленных, BRM состоит из еще нескольких более мелких сервисов:

- сервис регистрации, позволяющий клиентам создавать новые компании, предоставляет HTTP-сервер, поддерживающий основную операцию — создание компании с владельцем;

- сервис статистики, отвечающий за сбор статистики о компании, а также обновление рейтинга компаний, использует базы данных других сервисов, в частности core-сервиса;
- сервис изображений, обеспечивающий хранение и получение изображений объявлений и сотрудников.

3 ОПИСАНИЕ РАБОТЫ

3.1 Реализованная архитектура

По окончании разработки серверной части получили систему из 9 микросервисов. Для запуска сервера используется Docker, вся серверная часть приложения занимает 17 контейнеров — 9 контейнеров отведены под сами сервисы, еще 8 — под их зависимости (базы данных различного типа). Четыре микросервиса из девяти отведены под операции над основными сущностями: core-сервис, сервис объявлений, сервис сделок, сервис уведомлений. У каждого из них есть отдельная база данных. Это необходимо, чтобы отказ одной базы данных не повлек за собой аварию более чем на одном сервисе, например, при аварии на сервисе сделок пользователь все еще смог бы просматривать объявления. Таким образом, приложение разделено на микросервисы с целью минимизации потери функционала при отказе одного сервиса, например, из-за неудачного релиза или из-за высокой нагрузки.

Схема архитектуры приложения приведена в приложении А, ссылка на репозиторий с исходным кодом приведена в приложении Б.

3.2 Взаимодействие с клиентом

Из 9 микросервисов 4 имеют собственные HTTP-серверы для непосредственного взаимодействия с фронтендом:

- сервис регистрации;
- сервис аутентификации;
- API-сервис;
- сервис изображений.

Для удобства фронтенд-разработчика у каждого из этих сервисов есть Swagger-документация, которая позволяет взаимодействовать с API в удобном формате. Swagger-документация полезна бэкенд-разработчику при тестировании разработанного API и фронтенд-разработчику при ознакомлении с функционалом API, так как содержит описания всех

возможных запросов, в том числе их адреса, методы, query- и path-параметры, форматы тел запроса и ответа. На рисунке 7 продемонстрирован интерфейс Swagger-документации.

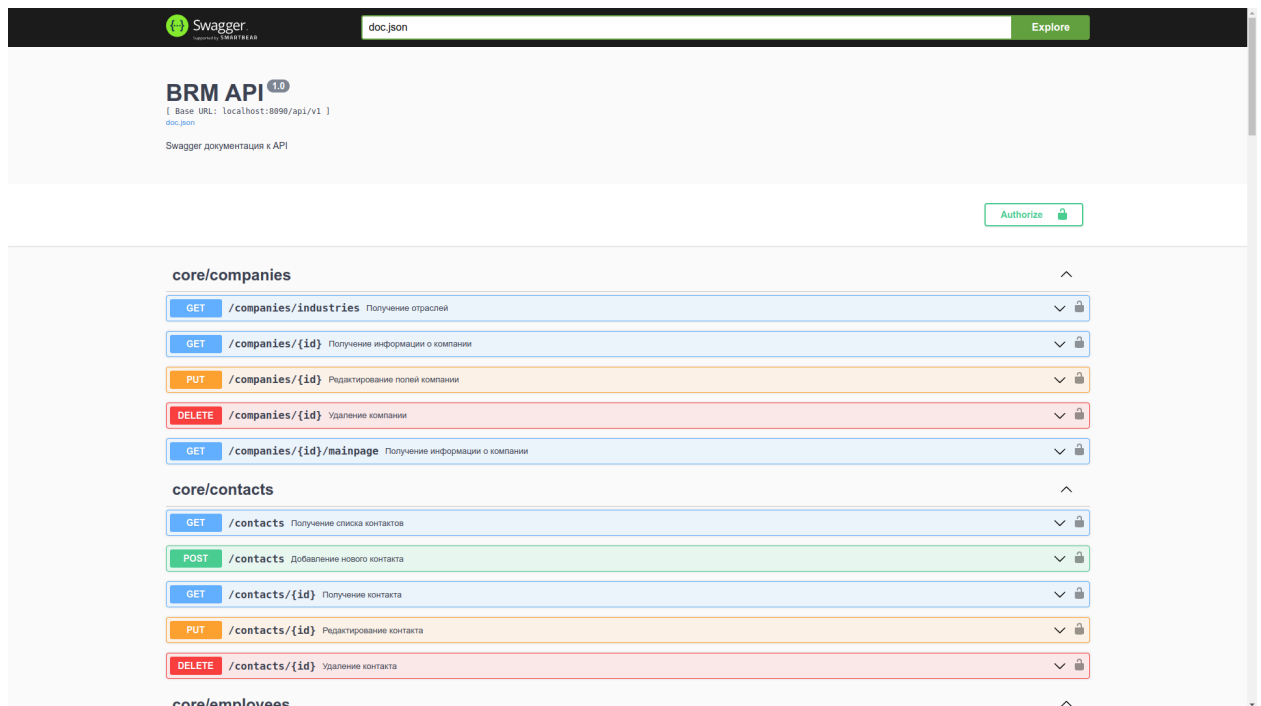


Рисунок 7 — Интерфейс Swagger-документации

3.2.1 Сервис регистрации

Процедуру регистрации проходят все новые клиенты BRM. Для прохождения регистрации не нужно иметь токен доступа, так как он выдается только существующим пользователям. Сервис регистрации поддерживает два типа запросов:

- добавление новой компании и владельца;
- получение списка отраслей, которые могут быть у компании.

3.2.2 Сервис аутентификации

С сервисом аутентификации фронтенд взаимодействует, когда пользователь вводит свои логин и пароль для входа в систему, или когда необходимо обновить access-токен пользователя. В этом случае взаимодействие с сервисом происходит незаметно для пользователя.

HTTP-сервер сервиса аутентификации поддерживает следующие запросы:

- получение пары токенов;
- удаление refresh-токена;
- обновление токенов.

3.2.3 API-сервис

Этот сервис предоставляет функционал для управления всеми компонентами CRM-системы. Для его использования пользователю необходим токен доступа, который можно получить в сервисе аутентификации. Токен необходимо передавать с каждым запросом в заголовке «Authorization» и с префиксом «Bearer ».

Запросы к API-сервису можно разделить на 6 логических частей:

- управление компаниями;
- управление контактами;
- управление сотрудниками;
- управление сделками;
- управление объявлениями;
- управление уведомлениями.

Для контактов, сотрудников и объявлений доступны все CRUDL-операции. Для компаний доступны операции получения, редактирования и удаления информации, создать компанию с помощью API-сервиса нельзя, для этого есть сервис регистрации. Для сделок доступны операции получения и редактирования, создать сделку с помощью API нельзя, так как сделка создается автоматически при отклике на объявление. Для управления уведомлениями в API доступны операции просмотра в виде списка или по отдельности и операция подтверждения закрытия сделки.

3.2.4 Сервис изображений

Взаимодействие фронтенда с этим сервисом происходит незаметно для пользователя каждый раз, когда пользователь запрашивает информацию, содержащую изображения.

HTTP-сервер сервиса изображений поддерживает следующие запросы:

- добавление изображения;
- получение изображения.

ЗАКЛЮЧЕНИЕ

В рамках выполнения выпускной квалификационной работы бакалавра была разработана CRM-система, обеспечивающая взаимодействие компаний друг с другом. Компании различных размеров получили площадку для размещения объявлений и торговли друг с другом, что позволяет отнести разработанное приложение в сферу B2B. Тем не менее, приложение сохраняет историю всех сделок компании, предлагает некоторые аналитические функции и обеспечивает коммуникацию компании и ее клиентов, поэтому его можно классифицировать как CRM-систему.

Приложение разработано в соответствии с принципами микросервисной архитектуры. Это означает, что поддерживается горизонтальное масштабирование сервисов и обеспечена высокая отказоустойчивость, так как отказ одного из сервисов не влияет на работоспособность всех остальных. Благодаря средству контейнеризации Docker, можно легко запустить все сервисы и зависимости приложения.

Разработку такой системы будет легко поддерживать в дальнейшем, так как каждый сервис разработан по правилам чистой архитектуры. Это значит, что внутренние компоненты сервисов могут быть легко заменены на другие в случае необходимости.

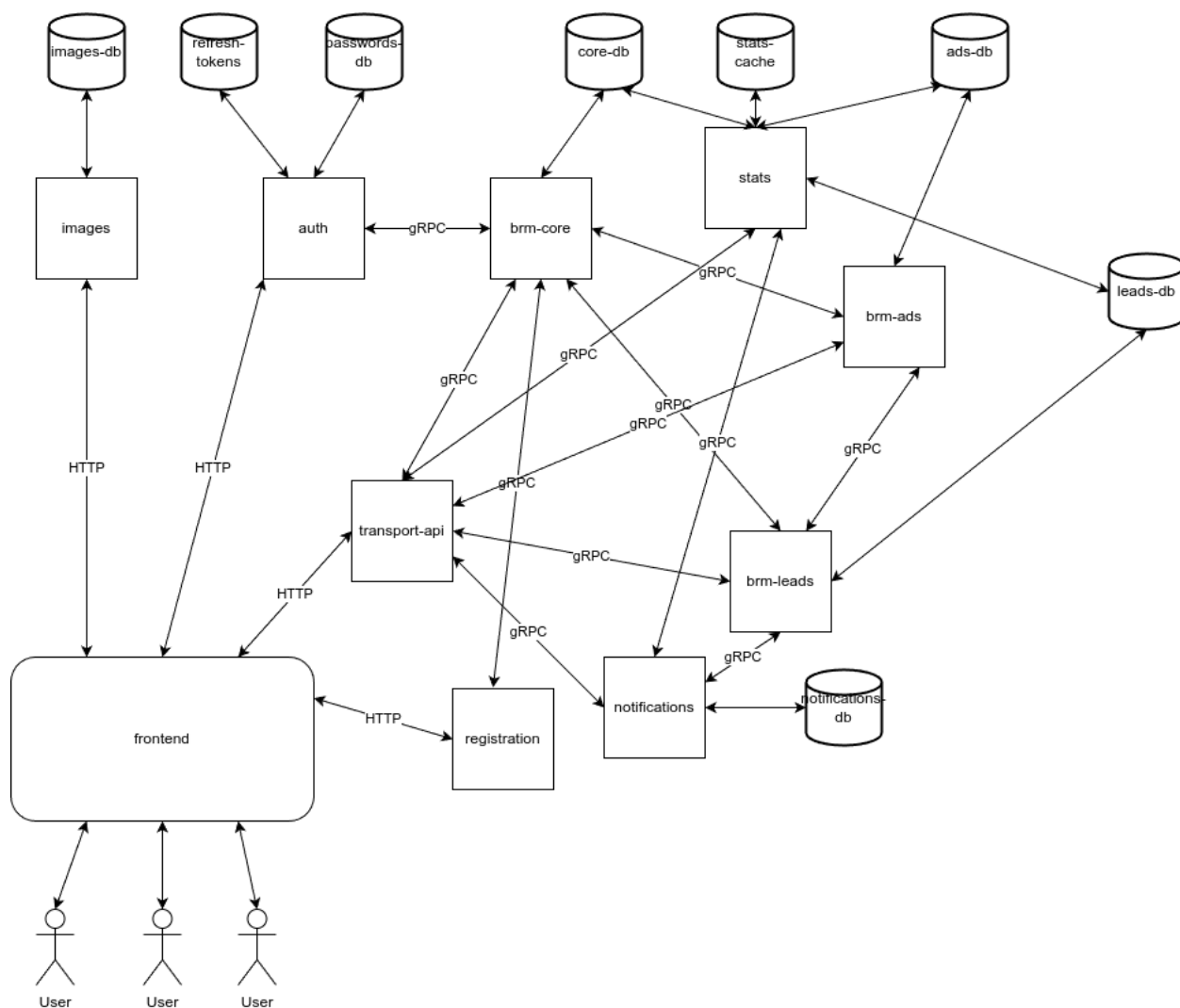
В процессе разработки был реализован весь планируемый функционал приложения. Предметная область включает в себя компанию, сотрудников, объявления и сделки. Благодаря микросервисной архитектуре, функционал приложения может быть легко расширен при необходимости.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Роберт Мартин. Чистая архитектура. Искусство разработки программного обеспечения. // Питер, 2021. – 352 с.
2. Полное руководство по CRM-системам [Электронный ресурс]. — URL: <https://www.oracle.com/cis/cx/what-is-crm/> (дата обращения 09.01.2024).
3. CRM [Электронный ресурс]. — URL: <https://sendpulse.com/support/glossary/crm> (дата обращения 09.01.2024).
4. Что такое CRM? [Электронный ресурс]. — URL: <https://www.sap.com/central-asia-caucasus/products/crm/what-is-crm.html> (дата обращения 09.01.2024).
5. CRM-система — что это и как работает [Электронный ресурс]. — URL: <https://megaplan.ru/news/articles/chto-takoe-crm-sistema/> (дата обращения 09.01.2024).
6. The Best CRM Software for 2024 [Электронный ресурс]. — URL: <https://www.pcmag.com/picks/the-best-crm-software> (дата обращения 09.01.2024).
7. Microservices. Как правильно делать и когда применять [Электронный ресурс]. — URL: <https://habr.com/ru/companies/dataart/articles/280083/> (дата обращения 02.02.2024).
8. 26 основных паттернов микросервисной разработки [Электронный ресурс]. — URL: <https://cloud.vk.com/blog/26-osnovnyh-patternov-mikroservisnoj-razrabotki> (дата обращения 02.02.2024).
9. A guide to Golang microservices [Электронный ресурс]. — URL: <https://www.cortex.io/post/golang-microservices> (дата обращения 02.02.2024).
10. Golang, Microservices, and Monorepo [Электронный ресурс]. — URL: <https://dev.to/bastianrob/golang-microservices-and-monorepo-gp3> (дата обращения 02.02.2024).

ПРИЛОЖЕНИЕ А

Схема архитектуры проекта



ПРИЛОЖЕНИЕ Б
QR-код репозитория с исходным кодом

