Разработка базы данных внутреннего портала для сотрудников предприятия

Студент: Княжев Алексей Викторович, ИУ7-52Б

Руководитель: Кострицкий Александр Сергеевич

Цель работы

Разработка базы данных внутреннего портала для сотрудников предприятия, а также приложения, позволяющего взаимодействовать с ней.

Задачи работы

— Формализовать задачу.

— Спроектировать базу данных.

— Выбрать СУБД.

— Реализовать базу данных, заполнить ее.

— Реализовать приложения для доступа к базе данных.

Обзор предметной области

С увеличением числа сотрудников компании часто возникают проблемы внутренней коммуникации сотрудников, получения информации о коллегах и автоматизации процессов рекрутмента. Для решения этих задач существуют внутренние порталы для сотрудников. Обычно они представляют функционал, схожий с соцсетями: отображение профилей пользователей, и др.

В основном, внутренние порталы для сотрудников представляют собой закрытые решения, которые разрабатываются в рамках одной компании, и не распространяются во внешний мир.

Сравнение предложенного решения с существующими

	Интранет VK	Avito People	Битрикс24	Предлагаемое решение
Возможность просматривать профили пользователей	+	+	+	+
Хранение информации об иерархии сотрудников	+	+	+	+
Возможность хранить информацию о команде сотрудника	-	+	-	+
Отсутствие платы за использование	+	+	-	+
Открытость решения для использования	-	-	+	+

Виды пользователей

— Сотрудник — сотрудник предприятия. Имеет базовые возможности по изменению части информации профиля, поиску среди пользователей системы.

— Рекрутер — менеджер по персоналу. Может создавать профили сотрудников, удалять их, изменять базовую информацию, такую как имя, должность.

— Администратор — главная роль в системе. Может делать то же, что может рекрутер. Кроме того, может редактировать базовую структуру организации, и управлять другими администраторами.

Сценарии использования

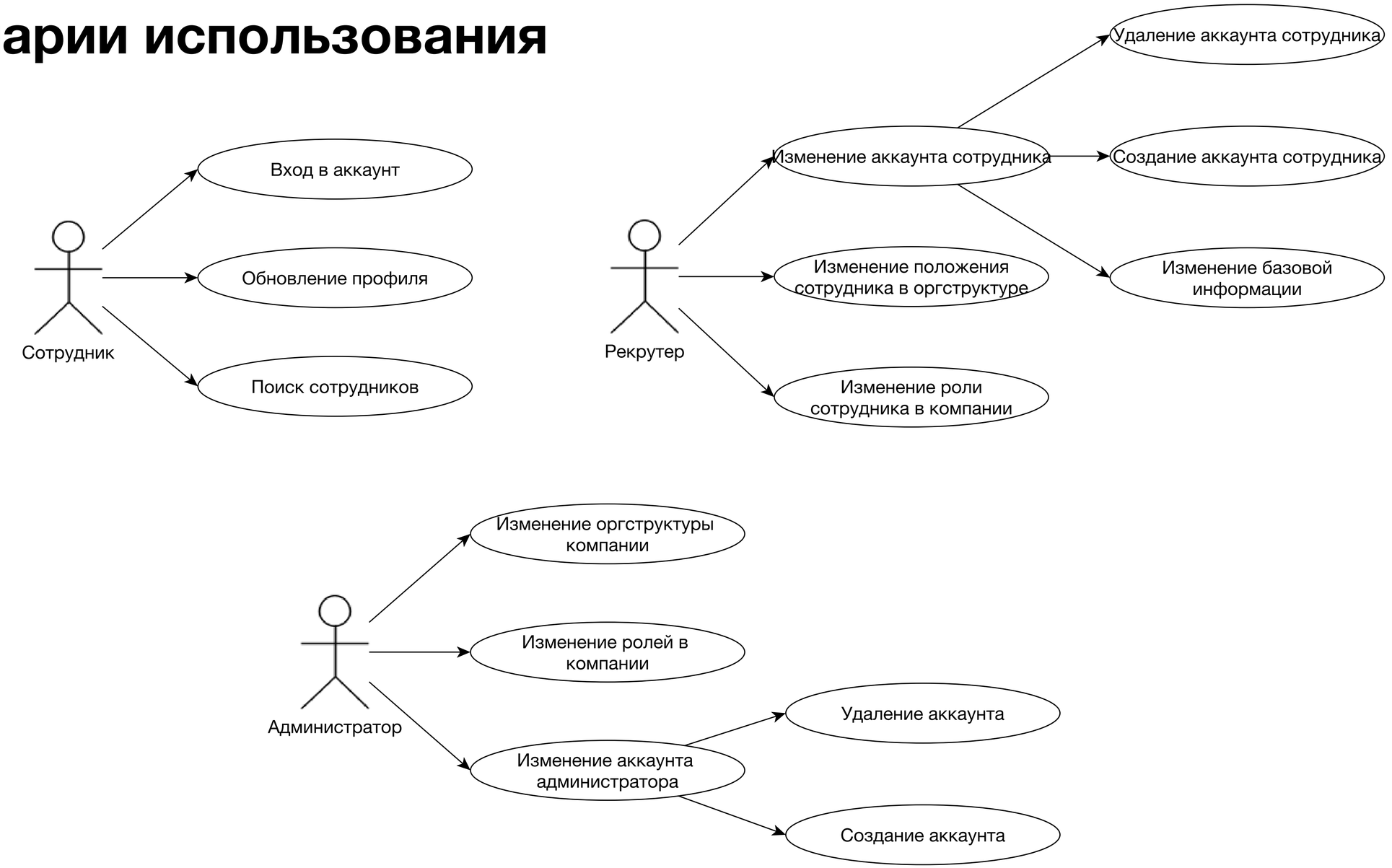
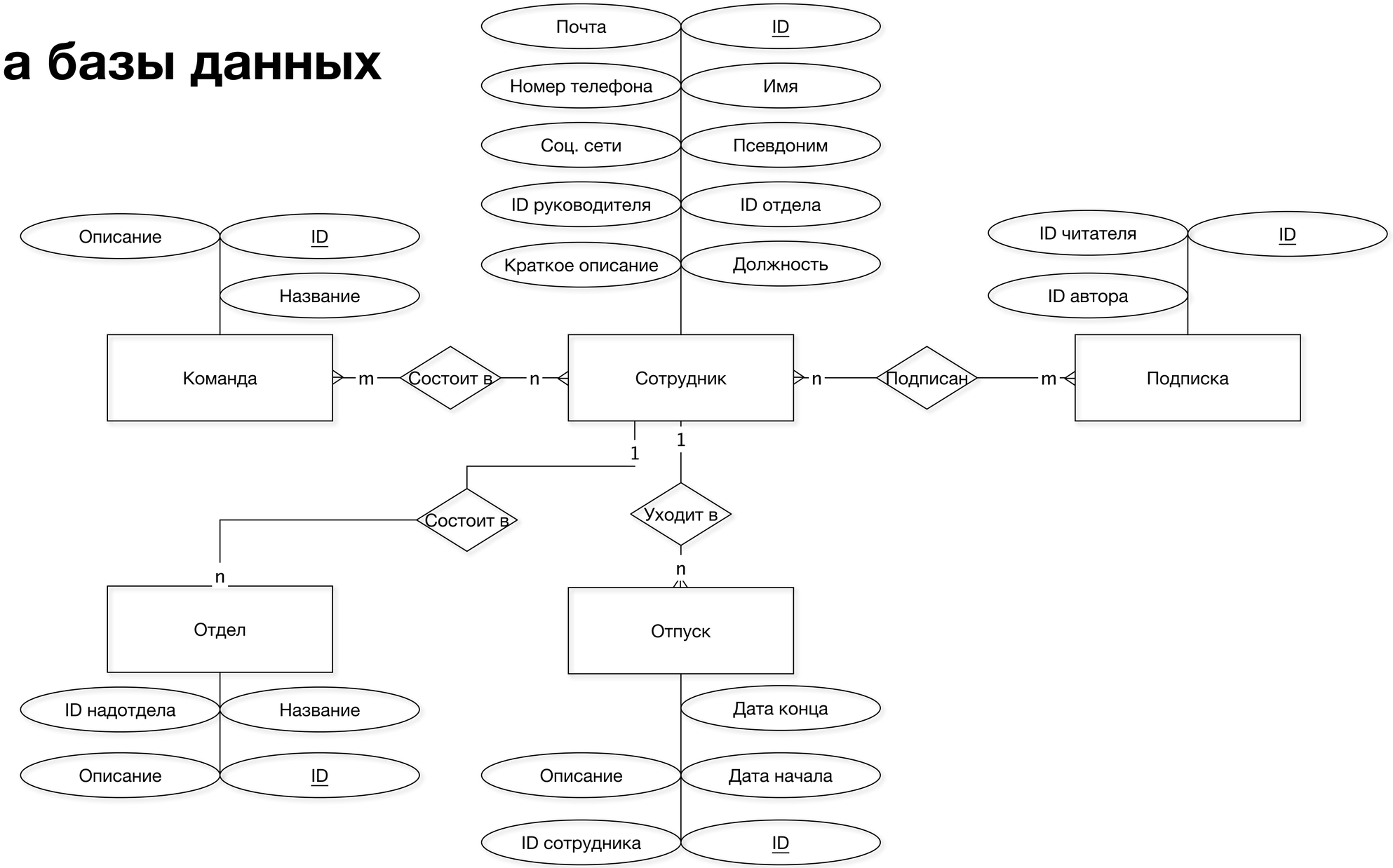


Схема базы данных



Ролевая модель

Сотрудник	
Subscription	SELECT, INSERT, DELETE
Team	SELECT
Employee	SELECT, UPDATE
Department	SELECT
Vacation	SELECT, INSERT

Рекрутер	
Subscription	SELECT, INSERT, DELETE
Team	SELECT
Employee	SELECT, UPDATE, INSERT
Department	SELECT
Vacation	SELECT, INSERT

Администратор	
Subscription	SELECT, INSERT, DELETE
Team	SELECT, UPDATE, INSERT, DELETE
Employee	SELECT, UPDATE, INSERT, DELETE
Department	SELECT, UPDATE, INSERT, DELETE
Vacation	SELECT, INSERT

Табличная функция

Была разработана табличная функция получения полного положения сотрудника в организационной структуре компании.

Алгоритм работы функции следующий.

1. Добавить текущий отдел к результату.

2. Если родительский отдел пуст, то возврат.

3. Если родительский отдел не пуст, то рекурсивно вызвать функцию поиска полного положения сотрудника в компании с аргументом — родительским отделом.

Средства реализации

Были выбраны следующие средства реализации:

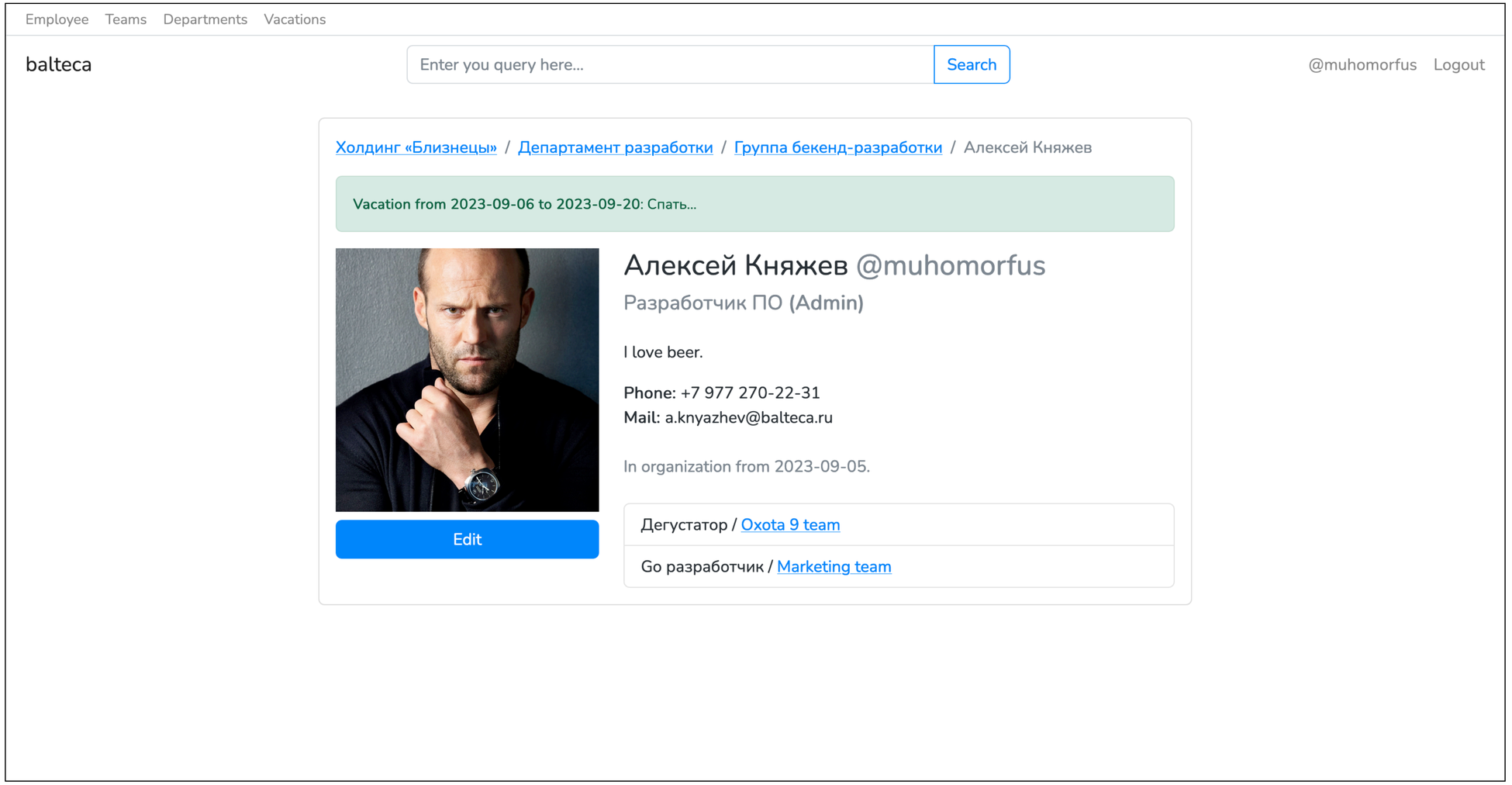
— СУБД: PostgreSQL.

— Язык программирования серверной части: Go.

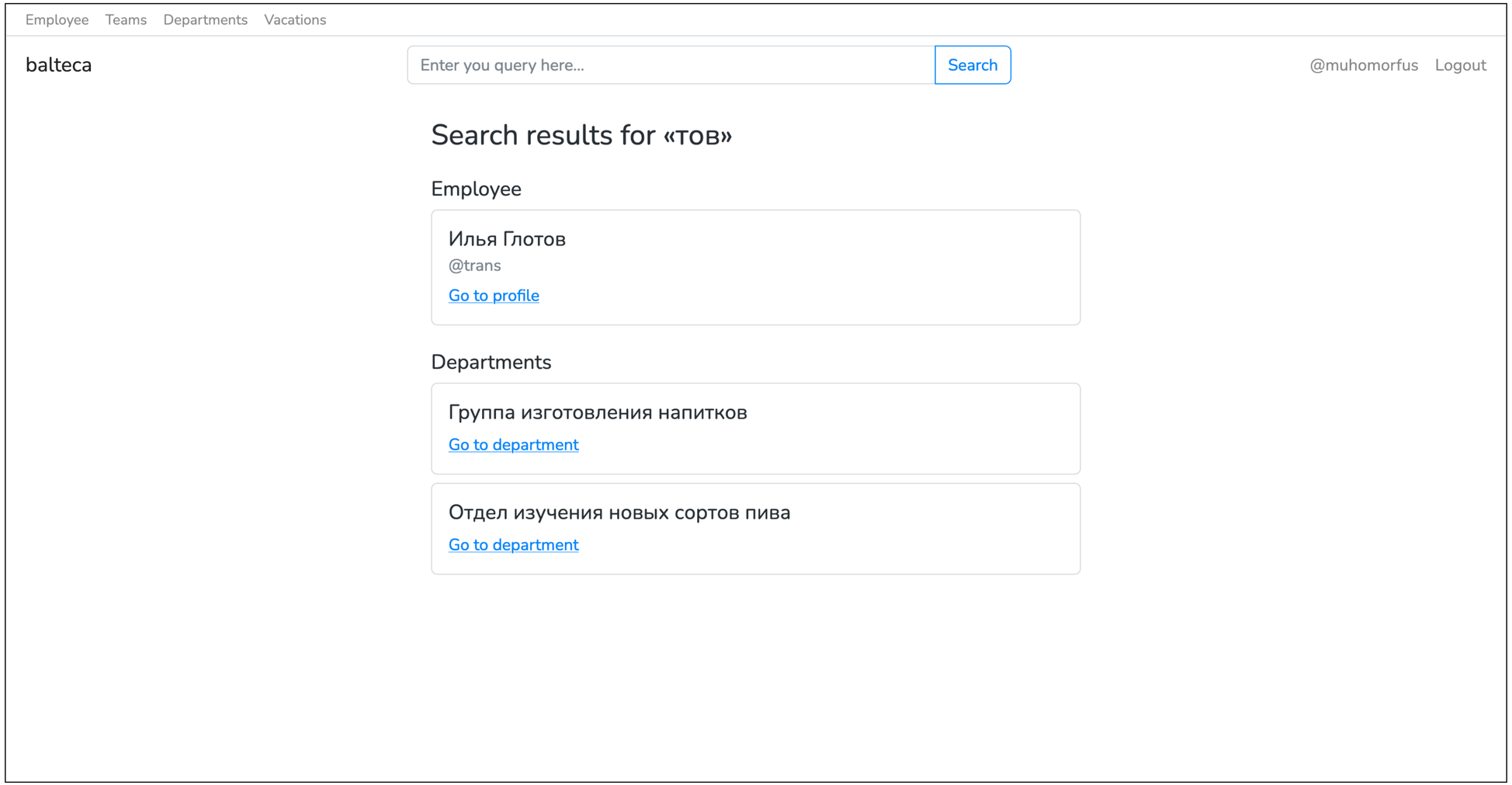
— Язык программирования клиентской части: TypeScript.

— Фреймворк для клиентской части: VueJS.

Примеры работы



Примеры работы

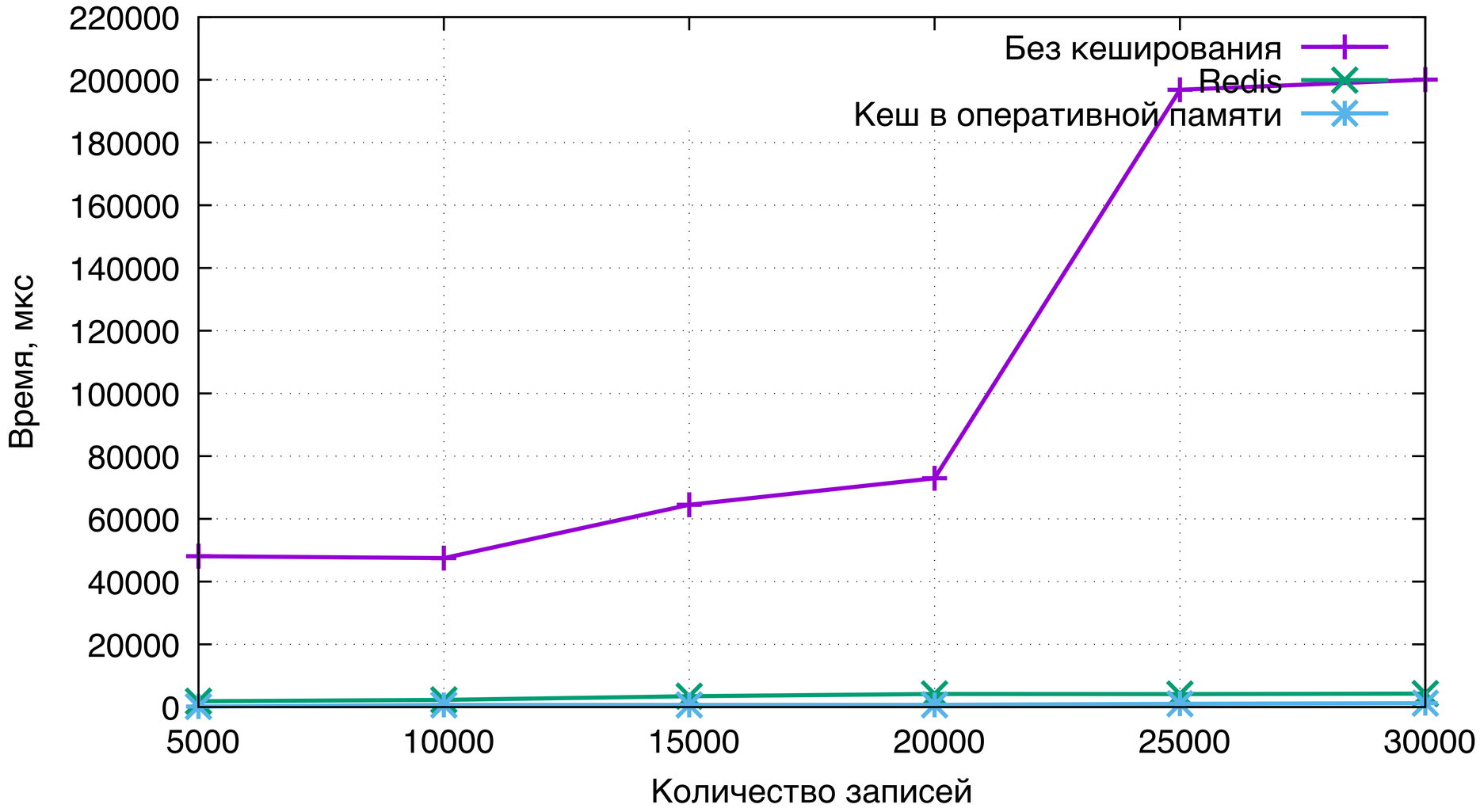


Исследование скорости работы поиска с кешированием и без

В рамках данного исследования проводилось сравнение скорости работы поиска без использования кеша и с использованием кеша двух видов: Redis и кеша в оперативной памяти. Результаты замеров времени работы поиска от количества сотрудников, отделов и команд, которых, в рамках данного исследования, одинаковое количество, представлены в таблице.

Количество записей	Время без кеша, мкс	Время с кешем в Redis, мкс	Время с кешем в ОЗУ, мкс
5000	48099	1855	175
10000	47488	2286	686
15000	64490	3455	693
20000	72954	4173	706
25000	196815	4132	1047
30000	200044	4290	1236

Исследование скорости работы поиска с кешированием и без



Исследование скорости работы поиска с кешированием и без

Исследуемая функция работает быстрее с кешированием. При этом, кеширование в оперативной памяти работает быстрее, чем кеширование в Redis на заданной выборке. Это связано в том числе с тем, что при использовании кеша в оперативной памяти, отсутствуют сетевые задержки при получении и отправке данных.

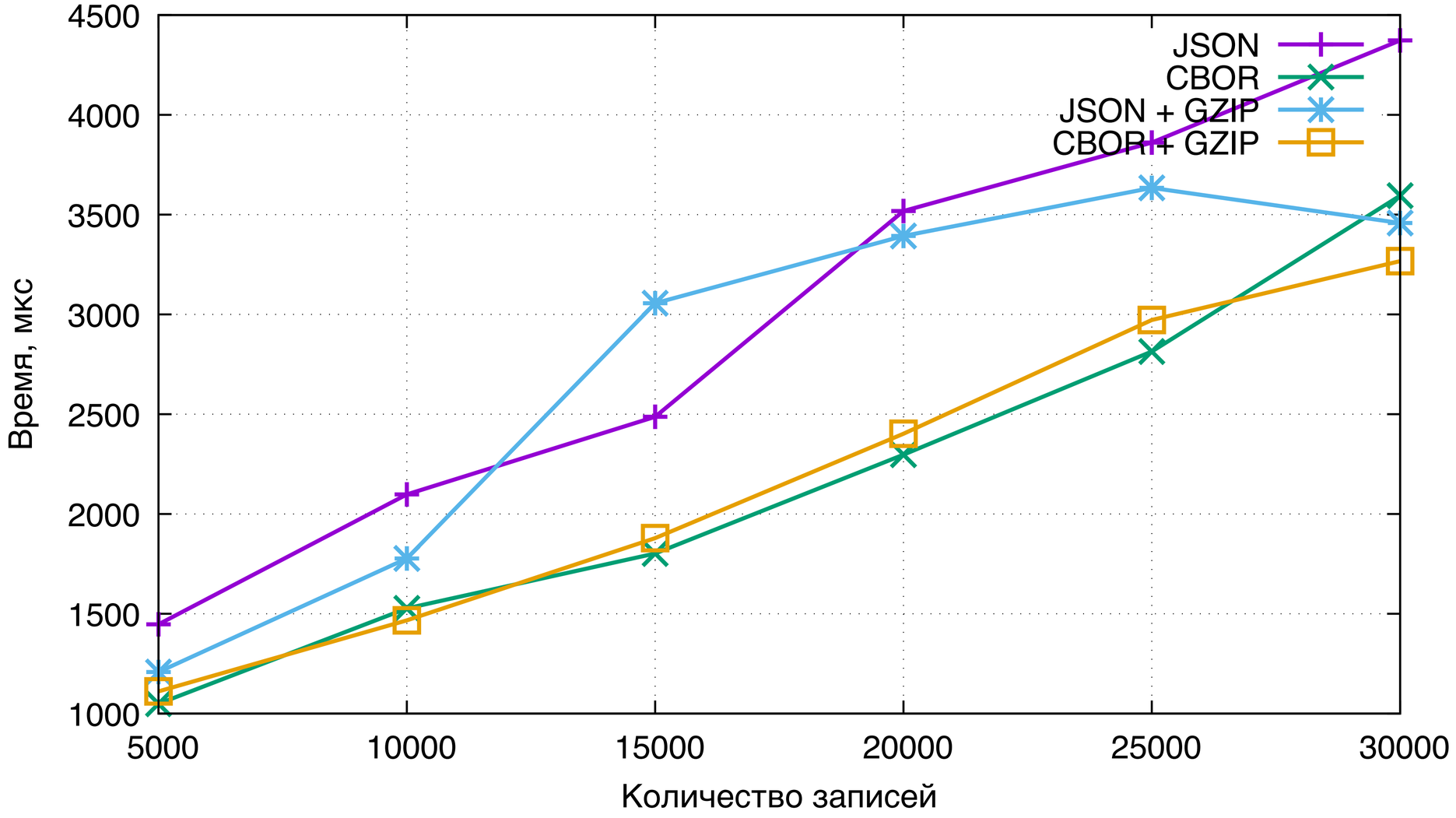
При объеме данных в 30 тысяч записей, запрос с использованием кеша в Redis работает примерно в 46 раз быстрее, чем без кеша. При этом, запрос с использованием кеша в оперативной памяти, работает быстрее реализации с кешем в Redis примерно в 3.4 раза.

Исследование скорости работы поиска с использованием различных методов сериализации при использовании кеширования в Redis

В рамках данного исследования проводилось сравнение скорости работы функции поиска с использованием различных алгоритмов сериализации при кешировании в Redis. В отличие от кеша в оперативной памяти, Redis хранит данные, в данном случае, в строковом формате, поэтому для хранения структуры необходимо сериализовать ее, затем преобразованную строку помещать в кеш.

Количество записей	JSON, мкс	CBOR, мкс	JSON+Gzip, мкс	CBOR+Gzip, мкс
5000	1447	1049	1208	1111
10000	2098	1527	1777	1467
15000	2487	1802	3056	1879
20000	3518	2297	3393	2402
25000	3861	2814	3634	2972
30000	4373	3595	3458	3267

Исследование скорости работы поиска с использованием различных методов сериализации при использовании кеширования в Redis



Исследование скорости работы поиска с использованием различных методов сериализации при использовании кеширования в Redis

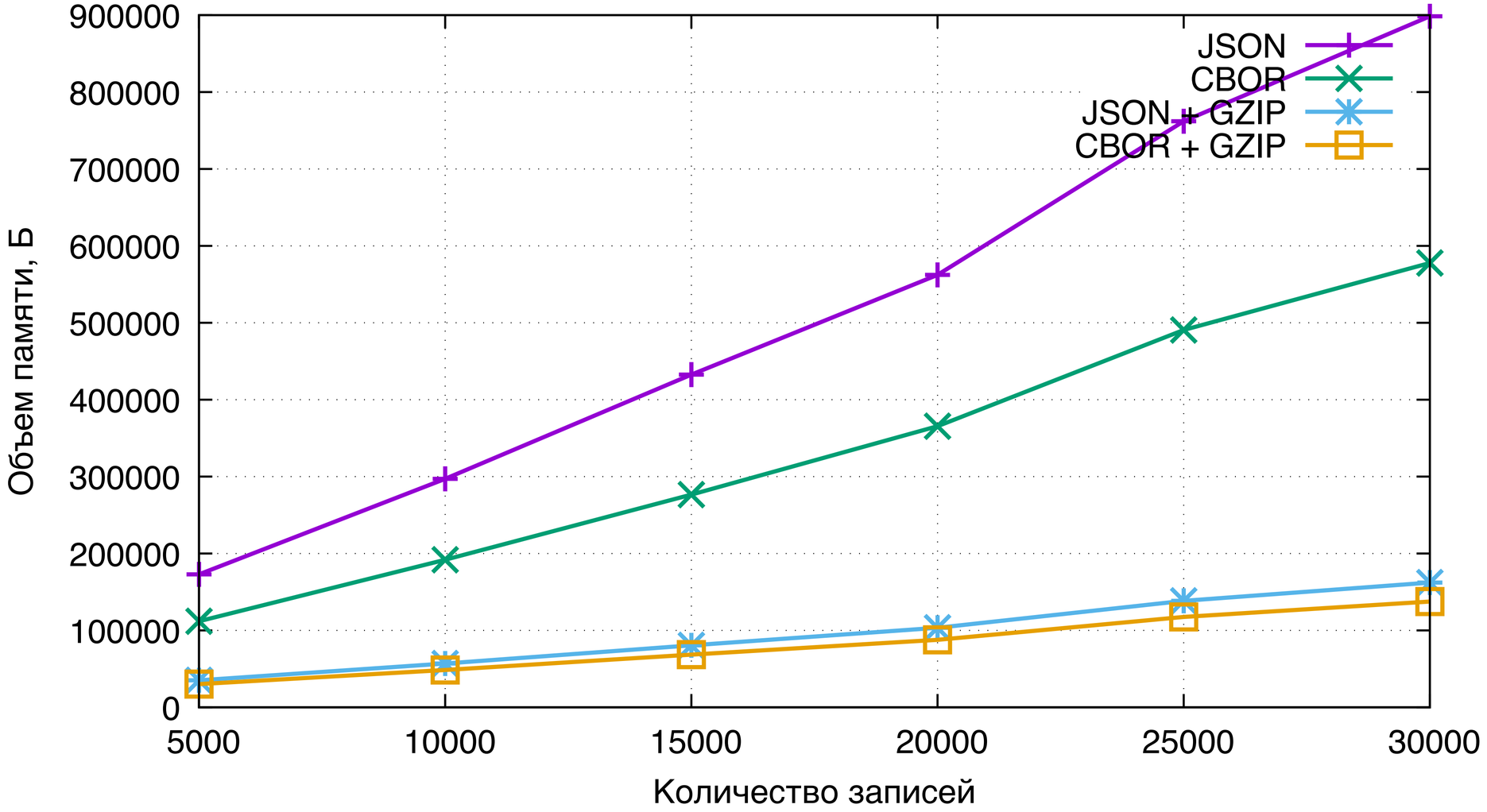
При объеме данных в 30 тысяч записей, самым долгим методом является сериализация в JSON, самым быстрым — сериализация в CBOR с использованием алгоритма сжатия Gzip. Разница во времени работы составляет примерно 1.3 раза. При этом, реализация с использованием комбинации JSON и Gzip работает быстрее реализации JSON без архивации, несмотря на большее количество операций. Такое поведение может быть связано с тем, что заархивированные данные занимают меньше места и быстрее передаются по сети.

Исследование объема занимаемой закешированными данными памяти при использовании различных алгоритмов сериализации

В рамках данного исследование производились замеры суммарного объема занимаемой ключами в Redis памяти в зависимости от количества записей в базе данных.

Количество записей	JSON, Б	CBOR, Б	JSON+Gzip, Б	CBOR+Gzip, Б
5000	172880	111920	35104	30176
10000	297088	191856	56928	48544
15000	432624	276576	80576	68512
20000	562256	365536	103440	87760
25000	762032	490624	138416	117504
30000	898544	577760	162192	137536

Исследование объема занимаемой закешированными данными памяти при использовании различных алгоритмов сериализации

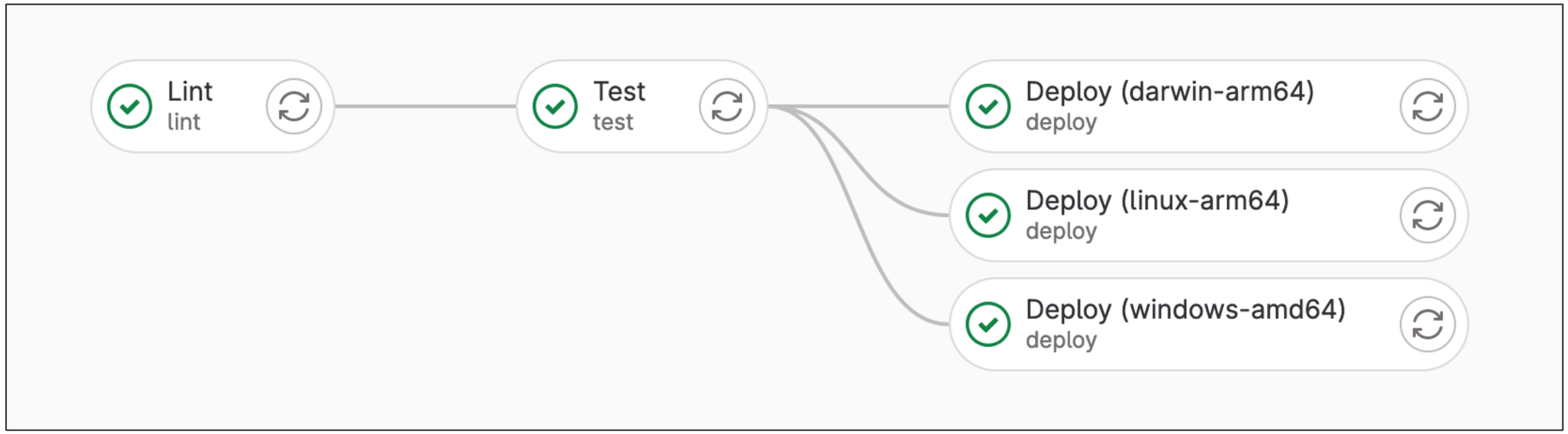


Исследование объема занимаемой закешированными данными памяти при использовании различных алгоритмов сериализации

При объеме данных в 30 тысяч записей, самым эффективным в плане потребляемой памяти является метод сериализации CBOR в комбинации с алгоритмом сжатия Gzip. Использование сжатия позволяет сократить потребление памяти примерно в 5.5 раз для JSON и примерно в 3.2 раза для CBOR. Такое значительное сокращение объема данных позволяет ускорить получение данных из кеша из-за меньшего объема данных, передаваемого по сети, что и показано в предыдущем исследовании.

CI/CD

Был реализован сценарий CI/CD, позволяющий выполнять тесты, проводить статический анализ кода и производить сборку приложения под разные платформы. Пример конвеера приведен на рисунке.



Вывод

В рамках выполнения работы была разработана база данных внутреннего портала для сотрудников организации. Все поставленные задачи были выполнены:

— формализована задача;
— спроектирована база данных;
— выбрана СУБД;
— реализована база данных;
— база данных заполнена данными;
— реализовано приложение для доступа к базе данных.

Цель работы — разработка базы данных внутренного портала для сотрудников предприятия, и приложения, позволяющего взаимодействовать с ней, была достигнута.

Дальнейшее развитие

В будущем в разработанную программу можно будет добавить такие функции, как:

— лента новостей;

— уведомления об отпусках и событиях;

— интеграции с внешними системами;

— награды за достижения сотрудников.