Оглавление

Введение

Последние исследования показывают, что зимой 2022 года россияне летали на горнолыжные курорты на треть чаще. Эксперты оценивали число купленных билетов на горнолыжные курорты и выяснилось, что российские курорты стали популярнее на 32.6%, а заграничные — на 40% [1].

В связи с этим растет спрос на функциональные и удобные приложения для горнолыжных курортов, которые помогали бы туристам ориентироваться на трассах, планировать свои маршруты по склонам с учетом текущих погодных условий и загруженности подъемников.

Цель курсовой работы – разработать базу данных для онлайн-мониторинга состояния трасс и подъемников горнолыжного курорта.

Чтобы достигнуть поставленной цели, требуется решить следующие задачи:

- проанализировать существующие решения;
- формализовать задачу и данные;
- проанализировать способы хранения данных и системы управления базами данных, выбрать наиболее подходящие решения для поставленной задачи;
- спроектировать и разработать базу данных;
- реализовать программное обеспечение, которое позволит получить доступ к данным посредством REST API [2];
- провести исследование зависимости времени доступа к данным от способа хранения данных и их объема.

1 Аналитическая часть

В данном разделе приведена формализация задачи и данных, проанализированы существующие решения, рассмотрены типы пользователей и требуемый функционал. Представлен анализ способов хранения данных, а также произведен выбор оптимального для решения поставленной задачи подхода к хранению данных.

1.1 Формализация задачи

Необходимо спроектировать и реализовать базу данных для онлайн-мониторинга состояния трасс и подъемников горнолыжного курорта. Также необходимо разработать интерфейс, позволяющий работать с данной базой для получения и изменения хранящейся в ней информации и мониторинга очередей к подъемникам в онлайн-режиме. Требуется реализовать, как минимум, три вида ролей – пользователь, сотрудник лыжного патруля и администратор.

1.2 Формализация данных

База данных должна хранить информацию о:

- трассах;
- подъемниках;
- связях трасс и подъемников (на одном подъемнике можно добраться до нескольких трасс, и до одной трассы можно добраться на нескольких подъемниках);
- турникетах;
- проездных картах;
- считываниях карт на турникетах подъемников;

- сообщениях о происшествиях;
- пользователях;
- группах пользователей.

В таблице 1.1 приведены категории и сведения о данных.

Таблица 1.1: Категории и сведения о данных

Категория	**Сведения**
Трассы	ID трассы, название трассы, уровень сложности, открытость/закрытость.
Подъемники	ID подъемника, название подъемника, открытость/закрытость, количество мест, время подъема, время в очереди.
Связи трасс и подъемников	ID записи, ID подъемника, ID трассы.
Турникеты	ID турникета, ID подъемника, открытость/закрытость.
Проездные карты	ID карты, дата и время активации, тип.
Считывания карт на турникетах подъемников	ID записи, ID турникета, ID карты, дата и время считывания.
Сообщения о происшествиях	ID сообщения, ID отправителя, ID прочитавшего, текст сообщения.
Пользователи	ID пользователя, ID карты, email (логин), пароль, ID группы пользователей.
Группы пользователей	ID группы пользователей, права доступа.

На рисунке 1.1 отображена ER-диаграмма системы, основанная на приведенной выше таблице.

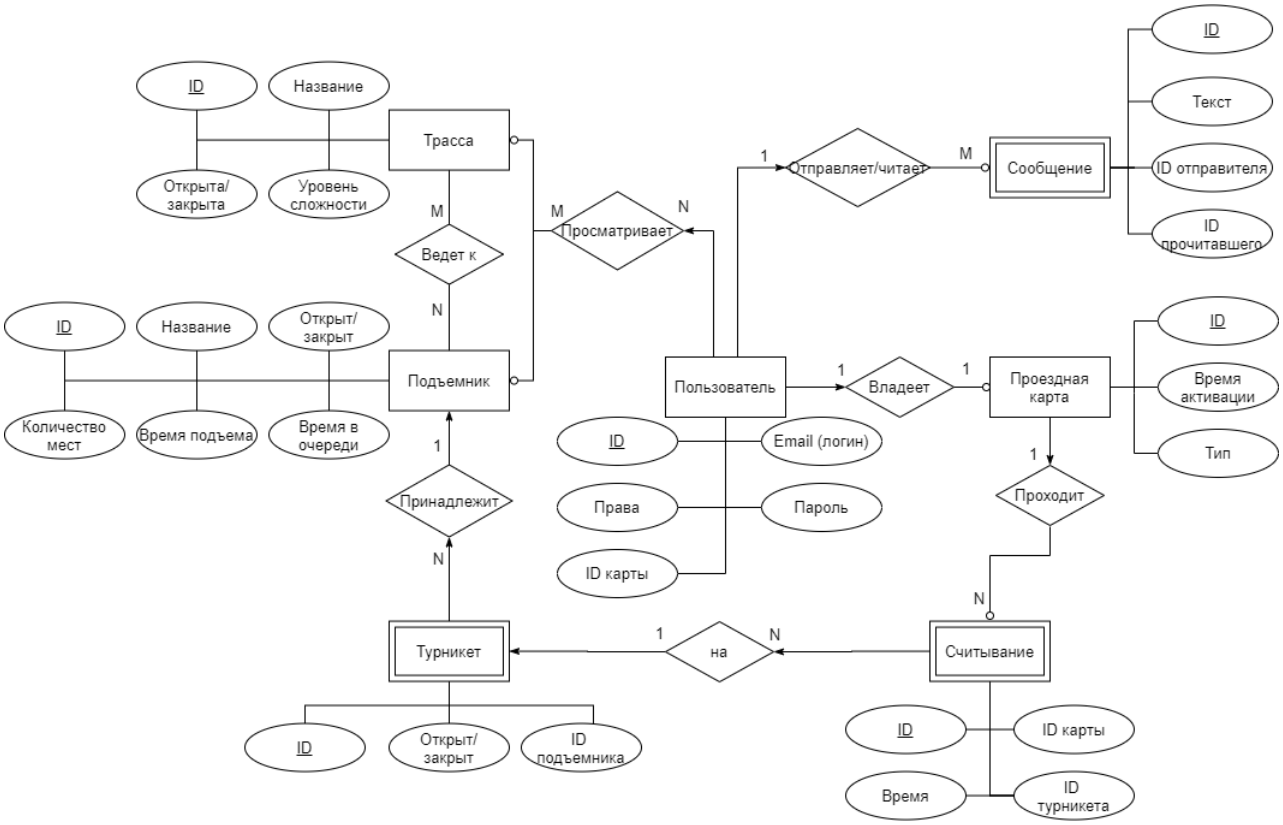


Рис. 1.1: ER-диаграмма

1.3 Типы пользователей

В соответствии с поставленной задачей необходимо разработать приложение с возможностью аутентификации пользователей, что делит их, прежде всего, на авторизованных и неавторизованных. Для управления приложением необходима ролевая модель: авторизованный (обычный) пользователь, сотрудник лыжного патруля и администратор.

Для каждого типа пользователя предусмотрен свой набор функций, который можно описать с помощью текста и Use Case Diagram (диаграммы прецедентов). Она состоит из графической диаграммы, описывающей действующие лица и прецеденты – конкретные действия, которые выполняет пользователь при работе с системой.

Набор функций неавторизованного пользователя:

- регистрация,
- аутентификация,
- просмотр информации о состоянии трасс и подъемников,
- просмотр информации о связях трасс и подъемников.

На рисунке 1.2 представлена диаграмма прецедентов для неавторизованного пользователя.

Рис. 1.2: Диаграмма прецедентов для неавторизованного пользователя

Набор функций авторизованного пользователя:

- выход,
- просмотр информации о состоянии трасс и подъемников,
- просмотр информации о связях трасс и подъемников,
- отправка сообщений о происшествиях.

На рисунке 1.3 представлена диаграмма прецедентов для неавторизованного пользователя.

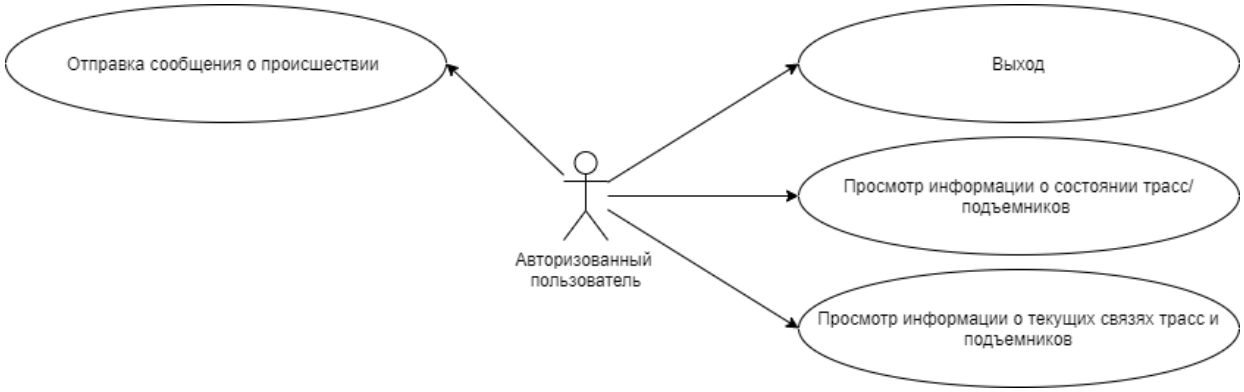


Рис. 1.3: Диаграмма прецедентов для авторизованного пользователя

Набор функций сотрудника лыжного патруля:

- выход,
- просмотр и изменение информации о состоянии трасс и подъемников,
- просмотр и изменение информации о связях трасс и подъемников,
- просмотр сообщений о происшествиях.

На рисунке 1.4 представлена диаграмма прецедентов для сотрудника лыжного патруля.

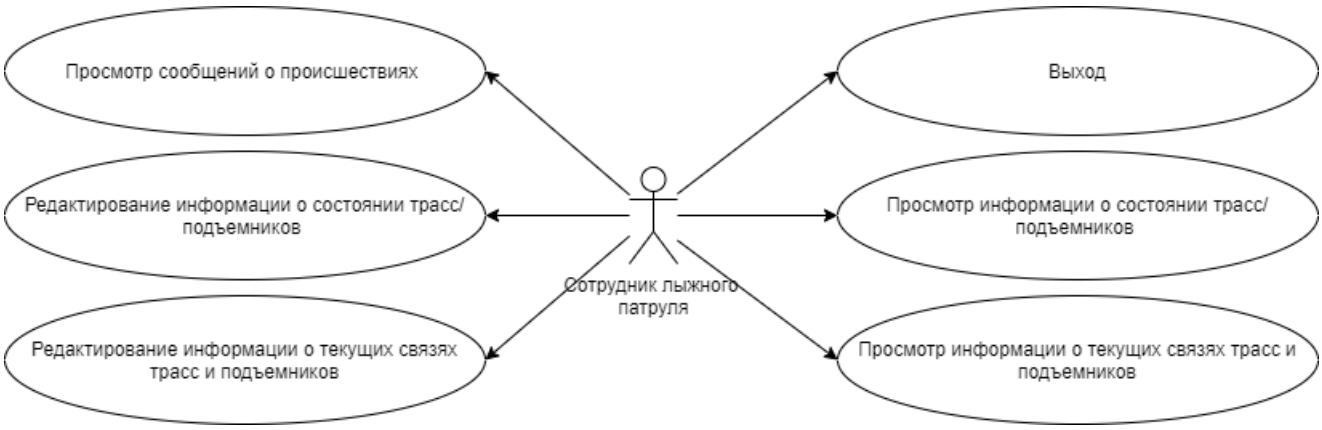


Рис. 1.4: Диаграмма прецедентов для сотрудника лыжного патруля

Набор функций администратора:

- выход,
- просмотр и изменение всей информации, доступной в базе данных, в том числе права доступа групп и отдельных пользователей.

На рисунке 1.5 представлена диаграмма прецедентов для администратора.

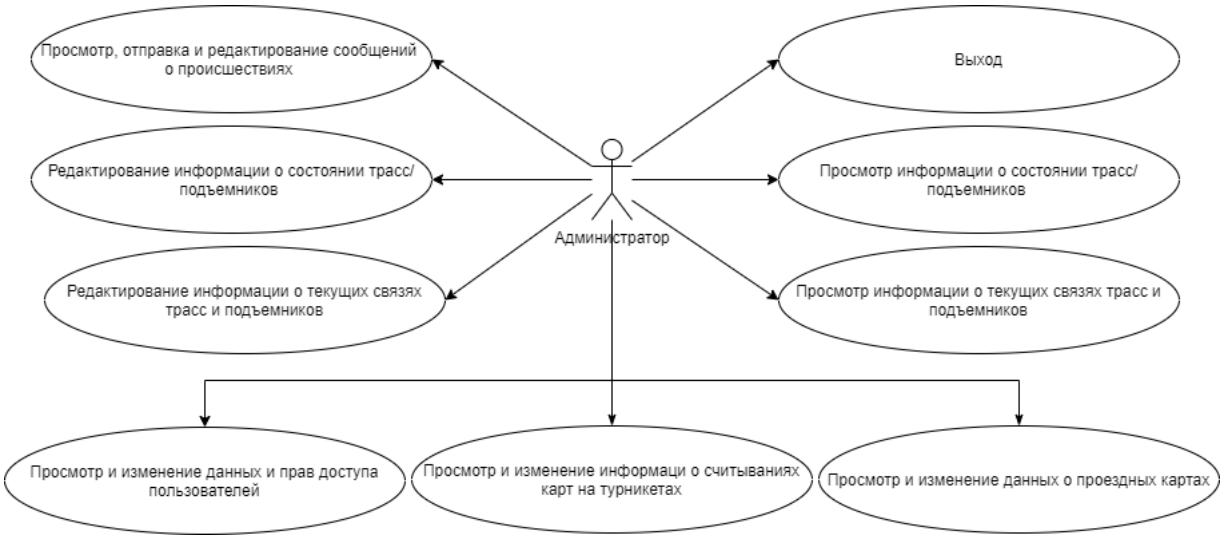


Рис. 1.5: Диаграмма прецедентов для администратора

1.4 Анализ существующих решений

1.4.1 Критерии

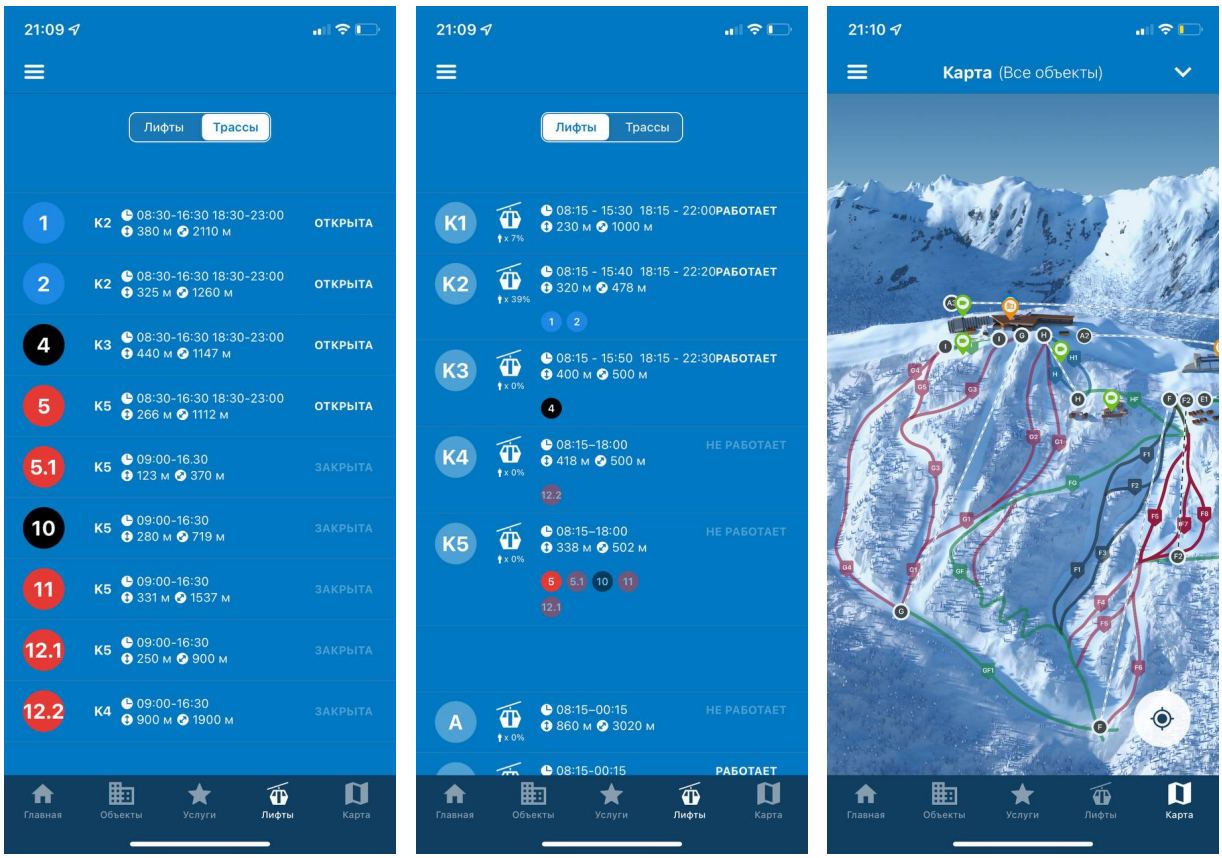
Проведем анализ аналогичных решений по следующим критериям:

1. наличие информации об открытости/закрытости трасс и подъемников;
2. наличие информации об очередях на подъемниках;
3. наличие сводной информации о связях трасс и подъемников;

4. возможность получить информацию о том, до каких трасс можно добраться на конкретном подъемнике/на каких подъемниках можно добраться до конкретной трассы.

1.4.2 Мобильное приложение горнолыжного курорта «Газпром»

В приложении горнолыжного курорта «Газпром» есть возможность просмотреть информацию об открытости/закрытости трасс (рисунок 1.6, a) и подъемников (рисунок 1.6, b), а также сводную информацию о связях трасс и подъемников с помощью карты курорта (рисунок 1.6, c):



(a) (b) (c)

Рис. 1.6: Просмотр информации в мобильном приложении «Газпром»

1.4.3 Мобильное приложение горнолыжного курорта «Courchevel»

В приложении горнолыжного курорта «Courchevel» на единой карте можно просмотреть информацию об открытости/закрытости трасс и подъемников, а также сводную информацию о связях трасс и подъемников (рисунок 1.7):

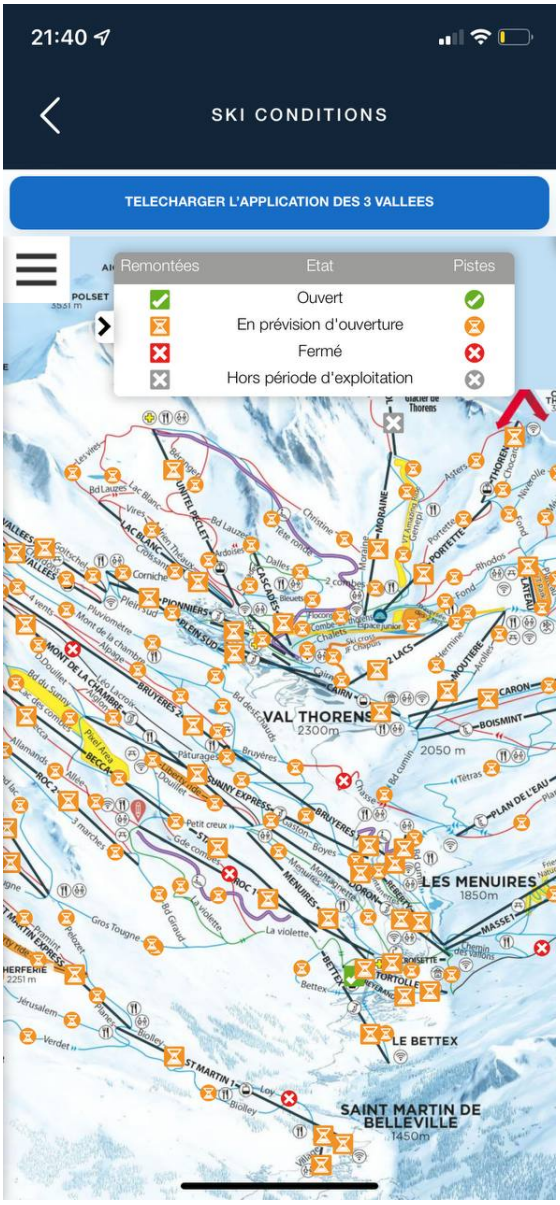
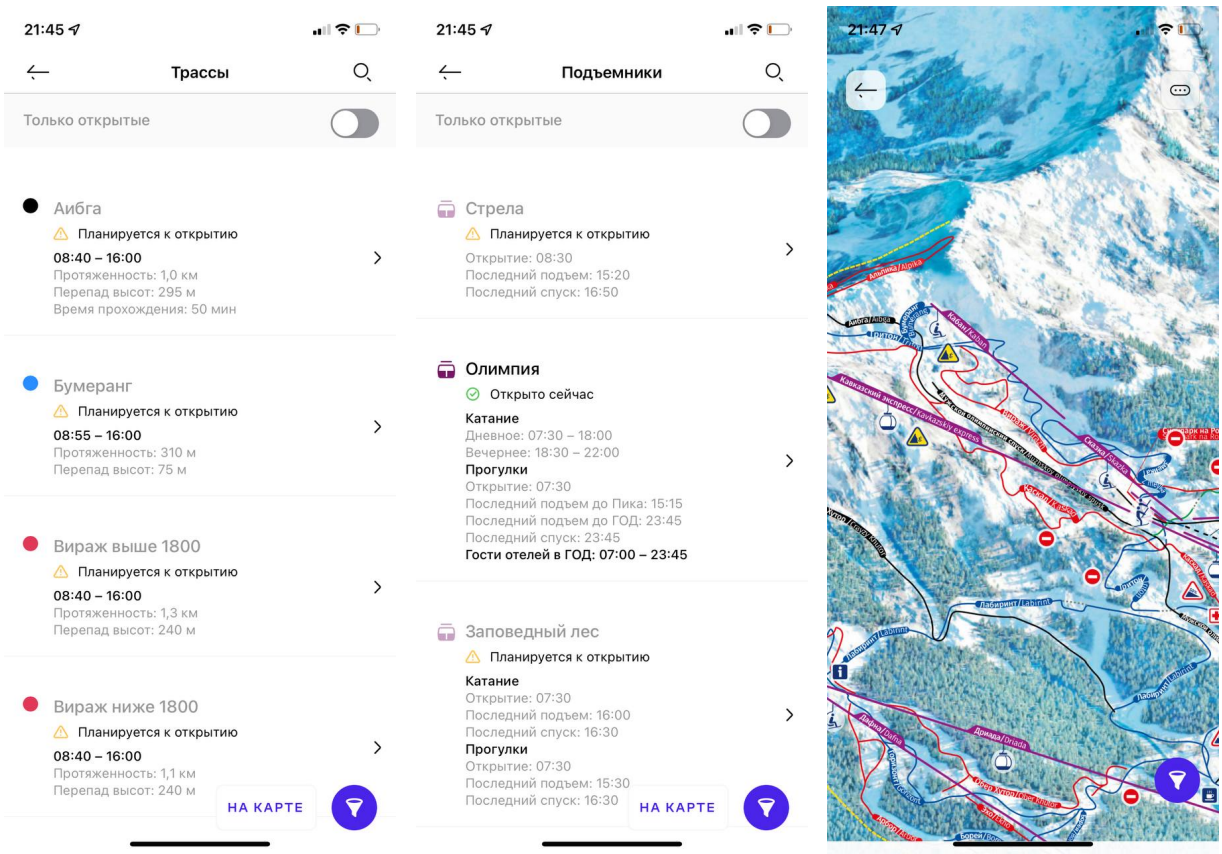


Рис. 1.7: Просмотр информации в мобильном приложении «Courchevel»

1.4.4 Мобильное приложение горнолыжного курорта «Роза Хутор»

В приложении горнолыжного курорта «Газпром» есть возможность просмотреть информацию об открытости/закрытости трасс (рисунок 1.8, a) и подъемников (рисунок 1.8, b), а также сводную информацию о связях трасс и подъемников с помощью карты курорта (рисунок 1.8, c):



(a) (b) (c)

Рис. 1.8: Просмотр информации в мобильном приложении «Роза Хутор»

Также есть возможность в онлайн-режиме просматривать камеры, расположенные на различных объектах курорта (пример показан на рисунке 1.9). Этим можно воспользоваться для мониторинга очередей на подъемниках.

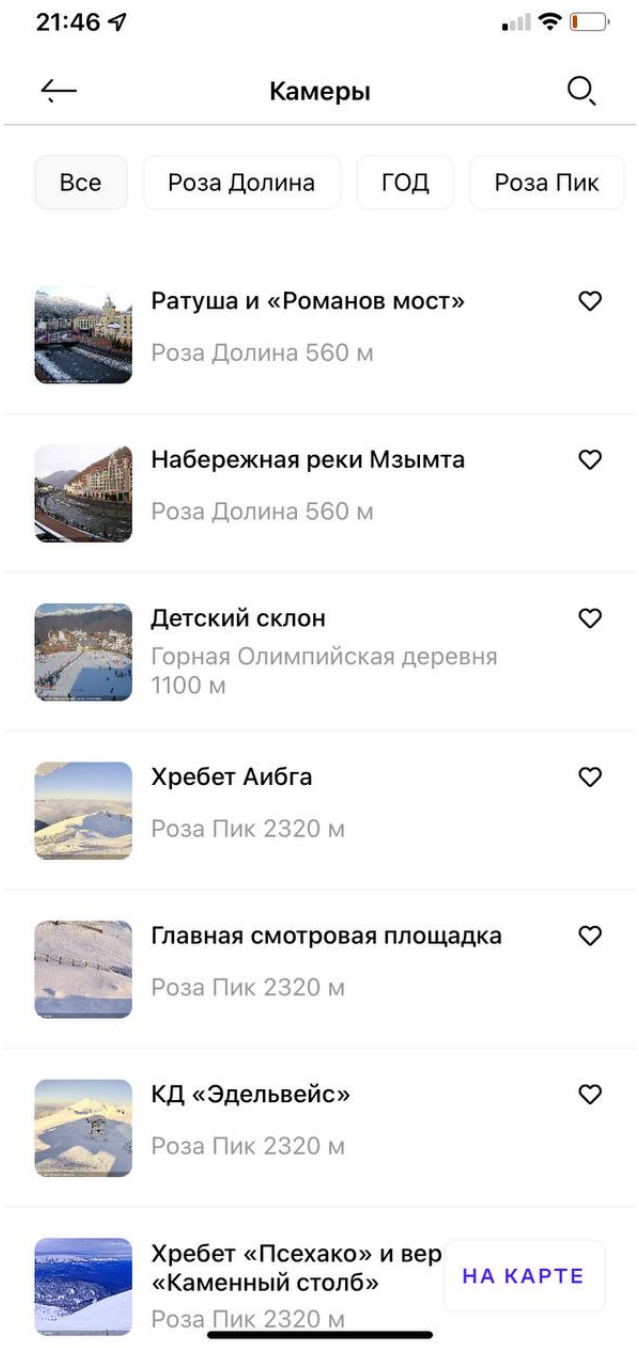


Рис. 1.9: Онлайн-просмотр камер в приложении «Роза Хутор»

1.4.5 Сравнение существующих решений

В таблице 1.2 приведено сравнение решений по наличию в них информации, перечисленной в пункте 1.4.1. В таблице приняты следующие обозначения: открытость – информация об открытости/закрытости трасс и подъемников, очереди – информации об очередях на подъемниках, связи (св.) – сводная информация о связях трасс и подъемников, связи (конкр.) – информация о связях конкретной трассы с подъемниками или наоборот.

Таблица 1.2: Сравнение решений

Название	Открытость	Очереди	Связи (св.)	Связи (конкр.)
Газпром	+ (список)	-	+ (на карте)	-
Courchevel	+ (на карте)	-	+ (на карте)	-
Роза Хутор	+ (список)	+ (камеры)	+ (на карте)	-

Таким образом, ни одно из приложений не отображает всю информацию, которую предполагается предоставлять в разрабатываемом приложении. В частности, наименее представлена информация об очередях на подъемниках, которая позволила бы посетителям выбирать мене загруженные объекты и тем самым делать нагрузку более равномерной. Предоставление же информации о том, как добраться до конкретной трассы позволит пользователям (в особенности, новым) упростить организацию своего катания.

1.5 Анализ баз данных

Для решения поставленной цели необходимо разработать базу данных (БД). БД – это упорядоченный набор структурированной информации или данных, которые обычно хранятся в электронном виде в компьютерной системе[3].

1.5.1 Классификация баз данных по месту хранения информации

По месту хранения информации БД можно разделить на [4]:

- традиционные, которые хранят информацию на жестком диске или другом постоянном носителе;
- in-memory databases (IMDB) (резидентные базы данных), которые хранят информацию непосредственно в оперативной памяти.

IMDB появились как ответ традиционным БД в связи со снижением стоимости оперативной памяти. Хранение данных в этой области позволяет увеличить скорость их обработки более чем в 1000 раз [5]. При этом резидентные БД обеспечивают высокую пропускную способность систем, критичных к производительности [6].

Обратной стороной этих достоинств являются следующие недостатки:

- однопоточность и эффективная утилизация только одного ядра процессора, что не позволяет в полной мере воспользоваться возможностями современных многоядерных серверов;
- энергозависимость и привязка к размеру оперативной памяти.

В практическом плане IMDB-системы особенно востребованы в тех приложениях работы с данными в реальном времени, где требуется минимальное время отклика [7].

1.5.2 Выбор БД по месту хранения информации

Основным требованием к разрабатываемой БД является предоставление возможности **онлайн**-мониторинга состояния объектов горнолыжного курорта. Задача предполагает постоянное добавление и изменение данных. С особо высокой частотой будут добавляться считывания карт на турникетах подъемников. При этом время в очереди на подъемник должно регулярно пересчитываться и обновляться, чтобы пользователь не получил

устаревшую информацию. Решение также предполагает быструю отзывчивость на запросы пользователя.

Таким образом, задача является типовым примером использования IMDB. И поскольку в современных in-memory СУБД существуют надежные и достаточно простые способы устранения указанных недостатков этих БД, было принято решение использовать именно этот подход к хранению данных.

Вывод

В данном разделе была проведена формализация задачи и данных, проанализированы существующие решения, рассмотрены типы пользователей и требуемый функционал. В результате анализа способов хранения данных, для данной задачи был сделан выбор в пользу резидентной БД.

2 Конструкторская часть

В данном разделе представлены этапы проектирования системы: функциональной модели, базы данных и архитектуры приложения.

2.1 Функциональная модель

На рисунке 2.1 изображена функциональная модель, отображающая структуру и функции системы.

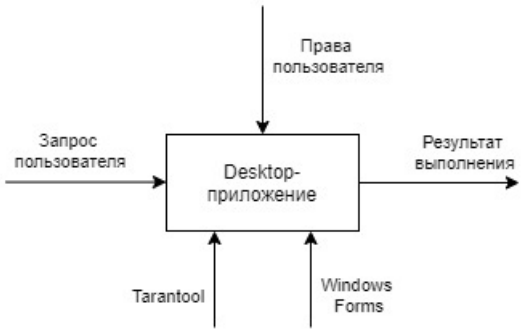


Рис. 2.1: Функциональная модель приложения

2.2 Проектирование базы данных

В соответствии с ER-диаграммой системы, изображенной на рисунке 1.1, база данных должна хранить следующие таблицы:

- таблица трасс slopes;
- таблица подъемников lifts;
- таблица связей трасс и подъемников lifts_slopes;
- таблица турникетов turnstiles;
- таблица проездных карт cards;
- таблица считываний карт на турникетах подъемников card_readings;

- таблица сообщений о происшествиях messages;
- таблица пользователей users.

На рисунке 2.2 представлена диаграмма разрабатываемой базы данных.

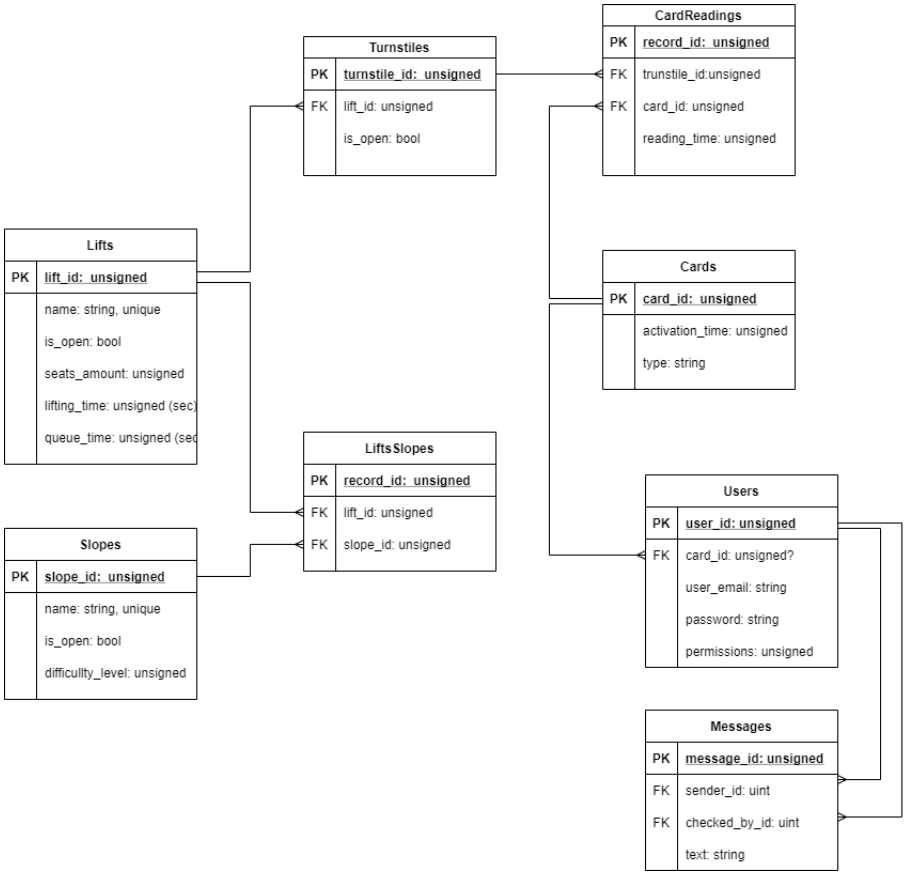


Рис. 2.2: Диаграмма базы данных

Таблица slopes хранит информацию о трассах и содержит следующие поля:

- slope_id – уникальный идентификатор трассы, PK (первичный ключ, англ. primary key);
- slope_name – уникальное название;
- difficulty_level – уровень сложности;
- is_open – открыта или закрыта.

Таблица lifts хранит информацию о подъемниках и содержит следующие поля:

- lift_id – уникальный идентификатор подъемника, PK;
- lift_name – уникальное название;
- is_open – открыт или закрыт;
- seats_amount – количество мест;
- lifting_time – время подъема;
- queue_time – время в очереди;

Таблицы slopes и lifts связаны отношением многие-ко-многим. Таблица lifts_slopes хранит информацию об этом отношении (связи трасс и подъемников) и содержит следующие поля:

- record_id – уникальный идентификатор записи, PK;
- lift_id – идентификатор подъемника, FK на поле lift_id таблицы lifts;
- slope_id – идентификатор трассы, FK на поле slope_id таблицы slopes.

Таблица turnstiles хранит информацию о турникетах и содержит следующие поля:

- turnstile_id – уникальный идентификатор турникета, PK;
- lift_id – идентификатор подъемника, на котором установлен этот турникет, FK на поле lift_id таблицы lifts;
- is_open – открыт или закрыт.

Таблица cards хранит информацию о проездных картах и содержит следующие поля:

- card_id – уникальный идентификатор карты, PK;
- activation_time – дата и время активации;
- type – тип карты (детская, взрослая, временная, ...).

Таблица card_readings хранит информацию о считываниях карт на турникетах подъемников и содержит следующие поля:

- record_id – уникальный идентификатор считывания, PK;
- turnstile_id – идентификатор турникета, на котором произошло считывание, FK на поле turnstile_id таблицы turnstiles;
- card_id – идентификатор проездной карты, которая прошла считывание, FK на поле card_id таблицы cards;
- reading_time – дата и время считывания.

Таблица users хранит информацию о пользователях и содержит следующие поля:

- user_id – уникальный идентификатор пользователя, PK;
- card_id – идентификатор проездной карты, которая принадлежит пользователю (может отсутствовать), FK на поле card_id таблицы cards;
- user_email – адрес электронной почты (он же будет использоваться как логин);
- password – пароль;
- permissions – права доступа (роль).

Таблица messages хранит информацию о сообщениях пользователей о происшествиях и содержит следующие поля:

- message_id – уникальный идентификатор сообщения, PK;
- sender_id – идентификатор отправителя, FK на поле user_id таблицы usres;
- checked_by_id – идентификатор прочитавшего, FK на поле user_id таблицы usres;
- text – текст сообщения.

2.3 Проектирование архитектуры приложения

Интерфейс для работы с базой данных представляет собой многокомпонентное desktop-приложение. На базовом уровне выделены три компонента: компонент доступа к данным, компонент бизнес-логики и компонент реализации пользовательского интерфейса. На рисунке 2.3 представлена архитектура приложения.

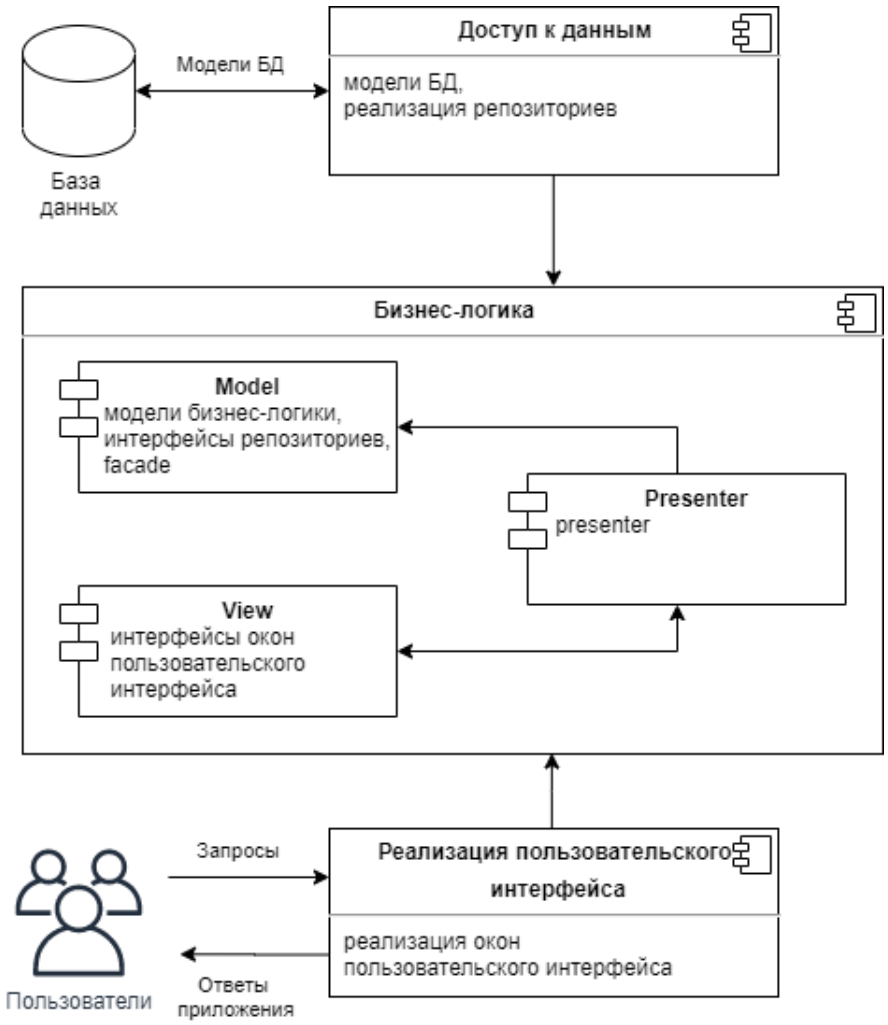


Рис. 2.3: Архитектура приложения

Компонент бизнес-логики спроектирован по схеме MVP (Model-View-Presenter) и является главным (независимым). Он отвечает за логику обработки запросов и данных.

Компонент доступа к данным отвечает за получение данных из БД, их

изменение, добавление и удаление. Для его реализации использован паттерн «репозиторий».

Компонент реализации пользовательского интерфейса предоставляет реализацию UI для выбранного технологического стека.

2.4 Схема алгоритма?

"

1) как лучше назвать этот подпункт

2) должен ли он быть здесь (так как обычно схемы в конструкторской части), или стоит перенести схему в 3.3.2, прям перед реализацией этой функции (и как лучше назвать сам этот пункт 3.3.2, а то «реализация функций» тоже звучит не очень)

"

Для онлайн-мониторинга очередей на подъемниках необходимо реализовать функцию update_queue_time, которая будет обновлять время в очереди к каждому из них. На рисунке 2.4 представлена схема работы алгоритма функций update_queue_time и вспомогательной функции count_card_readings, которая возвращает количество считываний на указанном подъемнике в заданный промежуток времени.

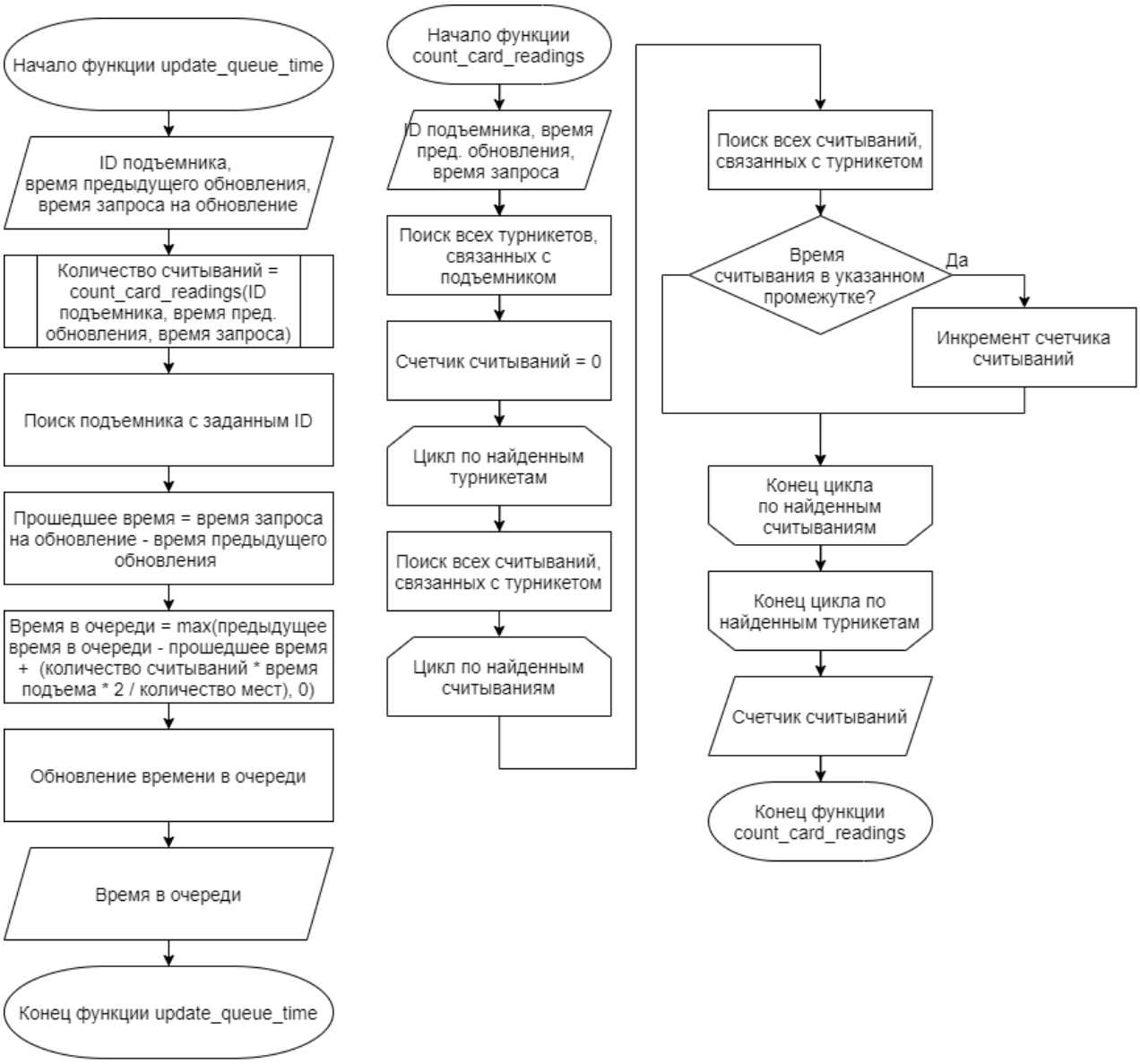


Рис. 2.4: Схема работы алгоритма функции update_queue_time

Вывод

В данном разделе была приведена функциональная модель системы, спроектирована база данных и архитектура приложения, приведена схема работы алгоритма для обновления времени в очереди на подъемниках.

3 Технологическая часть

В данном разделе проводится анализ in-memory СУБД и выбор наиболее подходящей для решения поставленной задачи, производится выбор инструментов разработки, приводятся детали реализации и интерфейс приложения.

3.1 Выбор in-memory СУБД

Система управления базами данных (СУБД) – это совокупность программных и лингвистических средств общего или специального назначения, обеспечивающих управление созданием и использованием баз данных[3].

В пункте 1.5.2 среди подходов к хранению данных был сделан выбор в пользу IMDB-систем, поэтому далее будет приведен обзор in-memory СУБД и произведен выбор наиболее подходящей для поставленной задачи.

3.1.1 Memcached

Memcached - это высокопроизводительная система кэширования данных в оперативной памяти, предназначенная для использования в ускорении динамических веб-приложений за счет уменьшения нагрузки на базу данных. Memcached относится к семейству решений для управления данными NoSQL и основана на модели данных с ключевыми значениями [8].

Данная СУБД спроектирована так, чтобы все операции имели алгоритмическую сложность O(1), то есть время выполнения любой операции не зависит от количества хранящихся ключей. Это означает, что некоторые операции или возможности, реализация которых требует всего лишь линейного (O(n)) времени, в ней отсутствуют. Так, например, в Memcached отсутствует возможность группировки ключей.

Memcached не является надежным хранилищем – возможна ситуация, когда ключ будет удален из кэша раньше окончания его срока жизни.

Управление внутренней памятью Memcached более эффективно в простейших случаях использования (при кэшировании относительно неболь-

ших и статических данных), поскольку оно потребляет сравнительно мало ресурсов памяти для метаданных. Строки (единственный тип данных, поддерживаемый Memcached) идеально подходят для хранения данных, которые только читаются, потому что строки не требуют дальнейшей обработки. Тем не менее, эффективность управления памятью Memcached быстро уменьшается, когда размер данных является динамическим, после чего память Memcached может стать фрагментированной [9].

Еще одно преимущество Memcached – достаточно простая масштабируемость: поскольку данная система многопоточна, ее можно увеличить, просто предоставив больше вычислительных ресурсов. Однако это может привести к потере части или всех кэшированных данных (в зависимости от того, используется ли постоянное хеширование).

3.1.2 Redis

Redis [10] – резидентная система управлениями базами данных класса NoSQL с открытым исходным кодом.

Основной структурой данных, с которой работает Redis является структура типа «ключ-значение», причем значения могут быть пяти различных типов. Данная СУБД используется как для хранения данных, так и для реализации кэшей и брокеров сообщений.

Redis хранит данные в оперативной памяти и снабжена механизмом «снимков» и журналирования, что обеспечивает постоянное хранение данных. Существует поддержка репликации данных типа master-slave, транзакций и пакетной обработки комманд.

Redis позволяет осуществлять мелкомасштабный контроль за вытеснением данных, предоставляя выбор из шести различных политик вытеснения [11].

К недостаткам Redis можно отнести отсутствие вторичных индексов и триггеров. Также транзакции в данной СУБД не удовлетворяют свойствам ACID (Atomicity – Атомарность, Consistency – Согласованность, Isolation – Изолированность, Durability – Долговечность) [12].

3.1.3 Tarantool

Tarantool [13] – это платформа in-memory вычислений с гибкой схемой хранения данных для эффективного создания высоконагруженных приложений. Включает себя базу данных и сервер приложений на языке программирования Lua [14].

Записи в Tarantool хранятся в пространствах (space) – аналог таблицы в реляционной базе данных SQL. Внутри пространства находятся кортежи (tuples), которые похожи на строку в таблице SQL.

Tarantool объединяет в себе преимущества, характерные для кэша:

- «горячие данные»;
- оптимальная работа при высокой параллельной нагрузке;
- низкая задержка (99% запросов $<$ 1 мс, 99,9% запросов $<$ 3 мс);
- поддерживаемая загрузка на запись — до 1 миллиона транзакций в секунду на одном ядре ЦПУ;
- система работает постоянно, не нужно делать перерыв на профилактические работы,

и достоинства классических СУБД:

- персистентность;
- транзакции со свойствами ACID;
- наличие репликации (master-slave и master-master);
- наличие хранимых процедуры.
- поддержка первичных и вторичных индексов (в том числе, составных).

В Tarantool реализован механизм «снимков» текущего состояния хранилища и журналирования всех операций, что позволяет восстановить состояние базы данных после ее перезагрузки.

К недостаткам данной СУБД можно отнести относительно малое количество поддерживаемых языков (C, C#, C++, Erlang, Go, Java, JavaScript, Lua, Perl, PHP, Python, Rust) [13], а также более высокий порог входа по сравнению с ранее рассмотренными СУБД [11].

3.1.4 Выбор СУБД для решения задачи

В данной работе не предполагается хранение в БД очень большого количества информации, и в этом случае требованию о коротком времени отклика удовлетворяют все рассмотренные СУБД.

Рассмотрим несколько требований, которым должна удовлетворять СУБД для решения поставленной задачи.

1. Для удобного хранения данных, указанных в пункте 1.2, СУБД должна предоставлять, как минимум , два типа данных – строки и целые числа.

2. Для предоставления пользователям перечисленных в пункте 1.3 функций, в СУБД должна быть реализована возможность создания вторичных индексов, а также триггеров или хранимых процедур для выполнения сложных вычислений.

3. Необходимо надежное хранение данных, без риска их потери даже в случае сбоя в системе (то есть поддержка репликации данных).

В таблице 3.1 приведено сравнение рассмотренных СУБД по перечисленным критериям.

Таблица 3.1: Сравнение in-memory СУБД

Название	Требование 1	Требование 2	Требование 3
Memcached	-	-	-
Redis	+	-	+
Tarantool	+	+	+

Из рассмотренных in-memory СУБД всем перечисленным требованиям удовлетворяет Tarantool, поэтому именно он был выбран для использования в данной работе.

3.2 Выбор инструментов разработки

В качестве языка программирования выбран язык C# [15]. Во-первых, он объектно-ориентирован, что позволит использовать наследование, интерфейсы, абстракции и так далее. Во-вторых, C# имеет коннектор progaudi.tarantool [16] для платформы in-memory вычислений Tarantool.

В качестве среды разработки была выбрана «Microsoft Visual Studio 2022» [17], поскольку она:

1) имеет множество удобств для написания и отладки кода;

2) является бесплатной для студентов;

3) обеспечивает работу с фреймворком Windows Forms[18], который был выбран для реализации пользовательского интерфейса из-за своей простоты в реализации.

Для упаковки приложения в готовый продукт была выбрана система контейнеризации Docker [19]. С его помощью можно создать изолированную среду для программного обеспечения, которое можно будет развернуть на различных устройствах без дополнительных настроек.

Тестирование программного продукта производилось с помощью фреймворка xUnit [20]. Данный фреймворк позволяет писать функциональные тесты.

3.3 Детали реализации

3.3.1 Ролевая модель

Необходимо реализовать ролевую модель в соответствии с пунктом 1.3. Роль – это разрешение, предоставляемое группе пользователей для доступа к данным.

Ролевая модель реализована на нескольких уровнях, в том числе, на уровне базы данных, что продемонстрировано в листинге 3.1. Последова-

тельно создаются и наделяются правами роли неавторизованного пользователя, авторизованного пользователя, сотрудника лыжного патруля и администратора, каждый наследует и расширяет права предыдущего.

```
box.schema.role.create('unauthorized_user')
box.schema.role.grant('unauthorized_user', 'read', 'space', 'lifts', {if_not_exists=true})
box.schema.role.grant('unauthorized_user', 'read', 'space', 'slopes', {if_not_exists=true})
box.schema.role.grant('unauthorized_user', 'read', 'space', 'lifts_slopes', {if_not_exists=true})


box.schema.role.create('authorized_user')
box.schema.role.grant('authorized_user', 'unauthorized_user')
box.schema.role.grant('authorized_user', 'read,write', 'space', 'messages', {if_not_exists=true})


box.schema.role.create('ski_patrol')
box.schema.role.grant('ski_patrol', 'authorized_user')
box.schema.role.grant('unauthorized_user', 'write', 'space', 'lifts', {if_not_exists=true})
box.schema.role.grant('unauthorized_user', 'write', 'space', 'slopes', {if_not_exists=true})
box.schema.role.grant('unauthorized_user', 'write', 'space', 'lifts_slopes', {if_not_exists=true})

box.schema.role.create('ski_admin')
box.schema.role.grant('ski_admin', 'read,write,execute,create,alter,drop', 'universe')
```

Листинг 3.1: Ролевая модель на уровне БД

3.3.2 Реализация функций

В листинге 3.2 представлены реализации функций update_queue_time и count_card_readings, предназначенных для обновлениия время в очереди к подъемнику. Алгоритм работы этих функций был описан в пункте 2.4.

```
function count_card_readings(lift_id, date_from, date_query)
```

```
2   connected_turnstiles = turnstiles.index.index_lift_id:select({lift_id})
3
4   counter = 0
5   for k,v in pairs(connected_turnstiles) do
6     cur_turnstile_id = v["turnstile_id"]
7
8     card_readings_on_turnstile = card_readings.index.index_turnstile:select({
      cur_turnstile_id})
9
10    for k,v in pairs(card_readings_on_turnstile) do
11      if (v["reading_time"] >= date_from and v["reading_time"] < date_query)
      then
12        counter = counter + 1
13      end
14    end
15  end
16  return counter
17 end
18
19 function update_queue_time(lift_id, date_from, date_query)
20   card_readings_amount = count_card_readings(lift_id, date_from, date_query)
21   lift = lifts:get{lift_id}
22   time_delta = date_query - date_from
23   new_queue_time = math_module.max(
24     math_module.ceil(
25       lift["queue_time"] -
26       time_delta +
27       (card_readings_amount * lift["lifting_time"] * 2 / lift["seats_amount"])
      ),
28     0)
29   lifts:update(lift_id, {{'=', 6, new_queue_time}})
30   return new_queue_time
31 end
```

Листинг 3.2: Функции update_queue_time и count_card_readings

3.3.3 Модели хранения данных

На примере таблицы спусков slopes рассмотрим модель хранения данных и реализацию доступа к данным.

В листинге 3.3 продемонстрировано создание спейса (таблицы) slopes в БД. Особенность Tarantool заключается в том, что для поиска кортежа (строки) в спейсе по значению его поля, это поле должно быть проиндексировано. Так как в спейсе slopes будет необходимо искать кортежи как по

значению поля slope_id (PK), так и по значению поля slope_name, то для этого спейса необходимо создать два соответствующих индекса (первичный и вторичный).

```
slopes = box.schema.space.create('slopes', {field_count=4})
slopes:format({
    {name = 'slope_id', type = 'unsigned'},
    {name = 'slope_name', type = 'string'},
    {name = 'is_open', type = 'boolean'},
    {name = 'difficulty_level', type = 'unsigned'}
})
slopes:create_index('primary')
slopes:create_index('index_name', {parts = {'slope_name'}})
```

Листинг 3.3: Создание спейса slopes

В листинге 3.4 приведена реализация модели slopes, используемой в компоненте бизнес-логики. Отношение между таблицами slopes и lifts многие-ко-многим здесь реализована с помощью хранения связанных со спуском подъемников в модели спусков, и наоборот. Связанные объекты добавляются в модель по запросу.

```
public record class Slope {
  public uint SlopeID { get; }
  public string SlopeName { get;}
  public bool IsOpen { get; }
  public uint DifficultyLevel { get; }
  public List<Lift>? ConnectedLifts { get; }

  public Slope(uint slopeID, string slopeName, bool isOpen, uint
   difficultyLevel){
    this.SlopeID = slopeID;
    this.SlopeName = slopeName;
    this.IsOpen = isOpen;
    this.DifficultyLevel = difficultyLevel;
  }
  public Slope(Slope slope, List<Lift> connectedLifts){
    this.SlopeID = slope.SlopeID;
    this.SlopeName = slope.SlopeName;
    this.IsOpen = slope.IsOpen;
    this.DifficultyLevel = slope.DifficultyLevel;
    this.ConnectedLifts = connectedLifts;
  }
```

```
21 }
```

Листинг 3.4: Модель базы данных на примере таблицы slopes

В листинге 3.5 приведена часть реализации репозитория TarantoolSlopesRepository, который обеспечивает доступ к данным спейса slopes из БД. Отражены функция GetSlopeByNameAsync, осуществляющая поиск спуска по его названию с помощью ранее созданного вторичного индекса, а также функция AddSlopeAutoIncrementAsync, которая осуществляет вставку нового кортежа, используя первичный ключ с автоматическим увеличением.

```
1  public class TarantoolSlopesRepository : ISlopesRepository
2  {
3    private IIndex _indexPrimary;
4    private IIndex _indexName;
5    private ISpace _space;
6    private IBox _box;
7    ...
8
9    public async Task<Slope> GetSlopeByNameAsync(string name)
10   {
11     var data = await _indexName.Select<ValueTuple<string>,
     SlopeDB>
12     (ValueTuple.Create(name));
13
14     if (data.Data.Length != 1)
15     {
16       throw new SlopeNotFoundException($"Error: couldn't find
     slope with name={name}");
17     }
18
19     return SlopeConverter.DBToBL(data.Data[0]);
20   }
21
22   public async Task<uint> AddSlopeAutoIncrementAsync(string
     slopeName, bool isOpen, uint difficultyLevel)
23   {
24     try
25     {
26       var result = await _box.Call_1_6<SlopeDBNoIndex,
     SlopeDB>("auto_increment_slopes", (new
```

```
    SlopeDBNoIndex(slopeName, isOpen, difficultyLevel)));
27          return SlopeConverter.DBToBL(result.Data[0]).SlopeID;
28        }
29        catch (Exception ex)
30        {
31          throw new SlopeAddAutoIncrementException();
32        }
33      }
34      ...
35  }
```

Листинг 3.5: Часть реализации репозитория для доступа к данным спейса slopes из БД

В используемом коннекторе progaudi.tarantool не реализована соответствующая функция «space_object:auto_increment()» СУБД Tarantool, поэтому на уровне базы данных для каждого спейса реализованы обертки для данной функции и конкретного спейса. Именно они вызываются из приложения, что продемонстрировано в функции AddSlopeAutoIncrementAsync в листинге выше. Пример такой функции-обертки на уровне БД приведен в листинге 3.6.

```
1  function auto_increment_slopes(slope_name, is_open,
       difficulty_level)
2    return box.space.slopes:auto_increment{slope_name, is_open,
       difficulty_level}
3  end
```

Листинг 3.6: Обертка для функции «space_object:auto_increment()» и спейса slopes

3.4 Интерфейс приложения

TODO

Вывод

На основе анализа in-memory СУБД был осуществлен выбор наиболее подходящей для решения поставленной задачи. Были выбраны инструменты разработки, приведены детали реализации и интерфейс приложения.

Список использованных источников

[1] Интерес туристов из РФ к горнолыжному отдыху [Электронный ресурс]. Режим доступа: https://booking-ru.ru/2020/09/20/interes-turistov-iz-rf-k-gornolyzhnomu-otdyxu-vyros-na-tret/ (дата обращения: 01.04.2022).

[2] What is a REST API? - Red Hat [Электронный ресурс]. Режим доступа: https://www.redhat.com/en/topics/api/what-is-a-rest-api (дата обращения: 01.04.2022).

[3] Что такое база данных | Oracle Россия и СНГ [Электронный ресурс]. Режим доступа: https://www.oracle.com/ru/database/what-is-database/ (дата обращения: 25.03.2022).

[4] Системы и технологии баз данных в памяти | Microsoft [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/sql/relational-databases/in-memory-database?view=sql-server-ver15 (дата обращения: 25.03.2022).

[5] In-Memory. База данных в оперативной памяти |ECM-Journal [Электронный ресурс]. Режим доступа: https://ecm-journal.ru/material/In-Memory-Baza-dannykh-v-operativnojj-pamjati (дата обращения: 27.03.2022).

[6] In-Memory Database [Электронный ресурс]. Режим доступа: https://ru.bmstu.wiki/IMDB_(In-memory_Database) (дата обращения: 27.03.2022).

[7] 4 крупных примера внедрения Tarantool | Big Data School [Электронный ресурс]. Режим доступа: https://www.bigdataschool.ru/blog/tarantool-use-cases-and-advantages.html (дата обращения: 27.03.2022).

[8] Memcached - a distributed memory object caching system [Электронный ресурс]. Режим доступа: https://memcached.org/ (дата обращения: 15.04.2022).

[9] Memcached | Распределенное хранилище данных типа «ключ-значение» [Электронный ресурс]. Режим доступа: https://memcached.org/ (дата обращения: 15.04.2022).

[10] Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache, and message broker [Электронный ресурс]. Режим доступа: https://redis.io/ (дата обращения: 01.04.2022).

[11] Преимущества Redis [Электронный ресурс]. Режим доступа: https://aws.amazon.com/ru/redis/ (дата обращения: 01.04.2022).

[12] Транзакции, ACID, CAP | GeekBrains [Электронный ресурс]. Режим доступа: https://gb.ru/posts/acid_cap_transactions (дата обращения: 07.06.2021).

[13] Tarantool – Платформа In-memory вычислений [Электронный ресурс]. Режим доступа: https://www.tarantool.io/ru/ (дата обращения: 01.04.2022).

[14] The Programming Language Lua [Электронный ресурс]. Режим доступа: http://www.lua.org/ (дата обращения: 01.04.2022).

[15] Документация C# [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата обращения: 03.04.2022).

[16] Dotnet client for Tarantool NoSql database [Электронный ресурс]. Режим доступа: https://github.com/progaudi/progaudi.tarantool (дата обращения: 03.04.2022).

[17] Документация по семейству продуктов Visual Studio [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/visualstudio/?view=vs-2022 (дата обращения: 03.04.2022).

[18] Windows Forms [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/desktop/winforms/?view=netdesktop-5.0 (дата обращения: 03.04.2022).

[19] Docker: Empowering App Development for Developers [Электронный ресурс]. Режим доступа: https://www.docker.com/ (дата обращения: 03.04.2022).

[20] Documentation site for the xUnit.net unit testing framework [Электронный ресурс]. Режим доступа: https://xunit.net/ (дата обращения: 03.04.2022).