|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |
| --- | --- |

ФАКУЛЬТЕТ ИУ, Информатика и системы управления

КАФЕДРА ИУ7, Программное обеспечение ЭВМ и информационные технологии

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К КУРСОВОЙ РАБОТЕ***

***НА ТЕМУ:***

***«Агрегация разнотипных каналов передачи данных»***

Студент ИУ7-32М **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. А. Андреев**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель курсовой работы **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.М. Никульшин**

(Подпись, дата) (И.О.Фамилия)

*2024 г.*

**РЕФЕРАТ**

Расчетно-пояснительная записка 33 с., 14 рис., 4 табл., 13 источников.

Объектом разработки является протокол агрегации разнотипных каналов передачи данных.

Цель работы — проектирование и реализация протокола, который осуществляет агрегацию разнотипных каналов передачи данных.

В ходе исследования была проведена аналитика предметной области, определены основные целевые аудитории, а также предложены ключевые сценарии применения разработанного протокола.

В рамках работы формализованы детали реализации клиент-серверного приложения, использующего разработанный протокол. Приложение предоставляет пользователю возможность авторизоваться, а также просматривать и управлять задачами. В рамках разработки успешно интегрированы механизмы проверки целостности данных, что обеспечивает надежность передаваемой информации, а также механизмы адаптации к различным характеристикам каналов передачи. Это позволяет улучшить эффективность и качество передачи данных, что особенно важно в условиях современной динамичной сети с переменными параметрами.

Таким образом, работа не только демонстрирует теоретическую конструкцию протокола, но и его практическую реализацию.

# Содержание

[**ВВЕДЕНИЕ**](#_rf61kp4knkff)4

[**1. Аналитический раздел**](#_mpjep0xrvzsa)5

[1.1 Концепция методологии Agile](#_vr2ty7uq8im) 5

[1.1.1 Итеративный подход к выполнению задач](#_ta1ssi4tukf8) 5

[1.1.2 Продуктовые команды](#_z3legs4ljd8d) 5

[1.1.3 Рабочее пространство](#_xcj3fm2glsy2) 6

[1.2 Существующие решения](#_gczpn3xm71ub) 7

[1.3 Предметная область](#_vm9f218hms0q) 8

[1.4 Системы управления базами данных](#_yihbvsy25pbp) 9

[1.4.1 Предметная область информационной системы](#_67luf8kbaqr5) 9

[1.4.2 Модель данных](#_e6sfncez1u9h) 10

[1.4.3 Типы моделей данных](#_7rby927pk3m6) 10

[1.5 Вывод из исследовательского раздела](#_ukvjagkhg8yn) 11

[**2. Конструкторский раздел**](#_ou5chkwfhhqd)12

[2.1 Проектирование структуры и отношений сущностей](#_uwjskgry7b9t) 12

[2.3 Внешние дополнительные хранилища](#_7fdbetrfkwpk) 13

[2.4 Стратегия кэширования](#_94snravfy6j2) 14

[2.2 Ролевая модель](#_vu5q1t7r41p4) 15

[2.5 Проектирование клиентского приложения для СУБД](#_ekg18bmobfiz) 16

[2.6 Вывод из исследовательского раздела](#_msyfo9pctlr0) 20

[**3. Технологический раздел**](#_cxude8mr8ftr)20

[3.2 Обоснование выбора языка, среды и платформы программирования](#_9cy7vpxfg6m0) 21

[3.3 Реализация клиентского приложения](#_u16j4ga06828) 21

[3.3 Реализация базы данных](#_ro83h0hmcbz5) 25

[**4 Исследовательский**](#_kcz3te7w87j) **раздел** 26

[4.1 Технические характеристики устройства](#_myvpeuvjfwse) 27

[4.2 Авторизация пользователей с включенным кэшем](#_a5f02nxygxwh) 27

[4.3 Повторная авторизация пользователей с включенным кэшем](#_oq6q6qjwgsz4) 29

[4.4 Повторная авторизация пользователей без кэша](#_jit1ibkdc1vk) 31

[4.5 Вывод из исследовательского раздела](#_q1ka9lzbufjo) 32

**ЗАКЛЮЧЕНИЕ** 33

[**СПИСОК**](#_krr503ad2d5j) **ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 34

# ВВЕДЕНИЕ

В современной эпохе стремительного развития информационных технологий надежная и эффективная передача данных становится основополагающим аспектом для многих сфер деятельности. С увеличением объема передаваемой информации и разнообразия типов каналов возникает необходимость в разработке протоколов, способных интегрировать различные каналы передачи данных. Это особенно актуально в условиях, когда требования к скорости и качеству связи становятся все более высокими, а также в условиях динамической сети, где характеристики каналов могут изменяться.

Данная работа посвящена разработке протокола для агрегации разнотипных каналов передачи данных. Основное внимание уделяется не только теоретическим основам создания такого протокола, но и практической его реализации, что способствует повышению качества передачи информации, отказоустойчивости систем и адаптивности к изменяющимся условиям сети.

Ключевыми аспектами разработки являются возможность гибкой адаптации протокола под различные типы каналов, высокая степень надежности и целостности передаваемых данных, а также упрощение управления сетью для конечного пользователя. В рамках исследования будет проведён анализ существующих решений, определены потенциальные целевые аудитории и предложены основные сценарии использования разработанного протокола.

Цель работы — проектирование и реализация протокола, который осуществляет агрегацию разнотипных каналов передачи данных.

Для достижения поставленной цели необходимо решить следующие задачи:

* проанализировать современных технологий агрегации каналов и протоколов, которые уже применяются в индустрии;
* определить целевую аудитории, которая может извлечь выгоду из использования протокола;
* спроектировать и реализовать структуры протокола, учитывающей разнообразные сценарии передачи данных;
* реализовать клиент-серверное приложение, обеспечивающее взаимодействие пользователей с протоколом и его функционалом.

# Аналитический раздел

Современные информационные системы предъявляют все более высокие требования к скорости и надежности передачи данных. С увеличением объемов передаваемых данных и необходимости обеспечения бесперебойного доступа возникает потребность в использовании нескольких каналов связи одновременно. Агрегация различных каналов передачи данных позволяет повысить общую пропускную способность и устойчивость системы к отказам отдельных компонентов. В данной работе рассматривается протокол, предназначенный для агрегирования различных каналов передачи данных, обеспечивающий эффективную и надежную передачу информации в распределенных системах.

## 1.1 Анализ существующих решений

Гибкая методология разработки (англ. Agile software development) — серия подходов к разработке программного обеспечения, ориентированных на использование итеративной разработки и динамическое формирование требований и обеспечение их реализации в результате постоянного взаимодействия внутри самоорганизующихся рабочих групп, состоящих из специалистов различного профиля. [1]

### 1.1.1 Протоколы агрегации каналов

Существуют различные подходы к объединению нескольких каналов связи. Наиболее распространенными являются:

* **Link Aggregation Control Protocol (LACP):** стандарт IEEE 802.1AX, позволяющий объединять несколько физических каналов в один логический канал на канальном уровне модели OSI. Однако LACP ограничен использованием однотипных каналов и не предоставляет гибкости в использовании разнородных сетей;
* **MultiPath TCP (MPTCP):** расширение протокола TCP, позволяющее установление соединений через несколько путей одновременно. MPTCP функционирует на транспортном уровне и обеспечивает прозрачную для приложений агрегацию каналов. Однако его внедрение требует поддержки на уровне операционной системы и сетевого оборудования;
* **Software-Defined Networking (SDN):** подход, в котором управление сетью осуществляется программно, позволяя более гибко распределять нагрузки и использовать различные каналы передачи данных. Тем не менее, внедрение SDN связано со сложностью архитектуры и необходимостью масштабных изменений инфраструктуры.

### 1.1.2 Недостатки существующих протоколов

Формализуем общие недостатки существующих протоколов:

* **Ограниченная совместимость:** многие протоколы требуют однородности сетевой инфраструктуры или специальной поддержки на уровне оборудования и операционных систем;
* **Сложность внедрения:** необходимость изменений на разных уровнях сетевой модели увеличивает сложность и стоимость внедрения;
* **Отсутствие универсальности:** существующие решения часто не рассчитаны на агрегацию разнородных каналов, таких как Wi-Fi, сотовые сети и проводные подключения одновременно.

### 1.1.3 Рабочее пространство

Доска задач — один из основных инструментов Agile-подхода, позволяющий отслеживать всю работу. На доске размечены столбцы, в которых размещаются задачи в виде карточек. Каждая задача (карточка) — это результат, которого должна добиться команда в рамках разработки. Результатом считается улучшение (инкремент) продукта, которое доступно для пользователя продукта. Каждый столбец — это процесс, который должна пройти задача, чтобы перейти в статус «выполнено». Использование доски позволяет в доступной форме визуализировать ход проекта разработки по всем направлениям для всех участников, предоставляет возможность быстро вычленить сложные и проблемные этапы или элементы и внести требуемые корректировки.

Как правило, на доске отображается следующий процесс [3]:

1. «Бэклог» — список задач команды для работы над продуктом;
2. «Нужно сделать» — список задач команды на итерацию разработки продукта;
3. «В процессе» — все задачи, находящиеся в работе;
4. «Нужно проверить» — все выполненные задачи, находящиеся на проверке после разработки;
5. «Готово» — выполненные задачи, проверенные согласно критериям определения готовности задачи.

## 1.2 Существующие решения

Управление проектами в Agile парадигме зачастую требует инструмент отслеживания прогресса задач в рамках процесса разработки продукта. Для этого используются специализированное программное обеспечение для управления задачами. Наибольшее распространение в сфере инструментов управления задачами получили продукты компании Atlassian, некоторые из которых рассматриваются далее.

Jira — это набор Agile-решений для управления работой, который обеспечивает совместную работу между всеми командами, начиная с разработки идеи продукта и заканчивая его доставкой клиенту, а также позволяет вам вместе добиваться наилучших результатов [4].

Организация работы команды в парадигме Agile стала ключевой особенностью этого решения, появившегося на рынке одним из первых. На текущий момент Jira — наиболее распространенное решение для управления проектами в рассматриваемой парадигме. Система, разработанная компанией Atlassian, позволяет вести работу в любых реализациях методологии Agile и координировать работы команд с различными реализациями рассматриваемой парадигме. Jira предоставляет большой набор возможностей для расширения при помощи пользовательских модификаций и интеграций со сторонними системами, в том числе с другими продуктами Atlassian.

Trello — это визуальный инструмент для управления работой, который позволяет командам обдумывать, планировать и вести совместную работу, а также отмечать успехи [5]. В отличии от другого рассмотренного продукта компании Atlassian — Jira, Trello представляет более упрощенный функционал работы над задачами. Все задачи представляются и привязываются только к одной доске, команды менее связаны общими целями и действуют более как обособленные структурные единицы. В отличии от Jira, в этом инструменте возможно удобное управление личными задачами в качестве списка дел отдельного человека. Аналогично, имеет набор возможностей для интеграции с другими внешними инструментами.

Redmine — это серверное приложение для управления проектами и задачами с открытым исходным кодом. В отличии от решений рассмотренных выше, распространяется по некоммерческой лицензии, что делает Redmine более распространенным среди малых и некоммерческих организаций. Аналогично продуктам Atlassian, предоставляет возможности интеграциями с системами управления версиями.

## 1.3 Предметная область

В организации процесса управления проектами, в соответствии с парадигмой Agile, можно выделить следующие сущности – объекты предметной области: Пользователь, Команда, Доска, Задача, Комментарий.

Пользователь — сущность, представляющая человека, участника процесса разработки продукта и относящегося конкретной команде. Пользователь может являться создателем (держателем) задач, а также исполнителем конкретной задачи. Этот человек может управлять задачами – редактировать описание, назначать другого исполнителя, менять статус в процессе работы над задачей. Рабочие пространства (доски) должны быть разграничены и принадлежать конкретной команде, которая управляет задачами на этих досках. Подобное ограничение обусловлено необходимостью ограничения списка лиц-владельцев задач и их исполнителями для достижения большей независимости и самоорганизации у команд-разработчиков продукта.

Команда — это некоторое количество специалистов, объединенных для работы над общим списком задач в целях разделения ответственности за разработку и доставку частей продукта его владельцу. В следствии того, что команда специалистов предполагается самоорганизующейся и кросс-функциональной, она имеет полное владение своим списком задач, распределенным по ее рабочим пространствам. Доступ лиц извне к списку задач команды возможен только для ознакомления к конкретным задачам в целях синхронизации работы нескольких команд, работающих над одним продуктом. Самоорганизация предполагает самостоятельное управление командой списком своих задач и их приоритезацией в согласии с целями владельцев продукта.

Рабочие пространства (доски) — это группы задач, которыми владеет и управляет определенная команда. В целях упрощения представления для владельцев продукта и самих членов команды, доски зачастую разделяются на колонки, соответствующие текущим статусам задач, размещенных на них. В процессе работы над каждой задачей и соответствующего изменения в ее статусе, она перемещается из столбца в столбец, обычно из правого в левый. Начальный статус определяет задачу обычно в самую левую колонку и, по прохождении всех этапов работы над задачей, она перемещается в самую правую. Таким образом, при помощи доски становится возможно быстро и понятно отследить прогресс каждой конкретной задачи команды.

Задачи — это единица декомпозиции работы команды над продуктом. Создаются обычно владельцем продукта и исполняются специалистами определенной команды. Зачастую задача имеет предполагаемую дату завершения и ответственного члена команды за ее исполнение на текущем этапе. Проблема невыполнения задачи в указанный срок в Agile считается ошибкой планирования а не исполнителя. Вследствие перехода в следующий статус, задача может быть назначена другому исполнителю для разделения работы над задачей между специалистами разных уровней и компетенций внутри одной команды.

Комментарий — это единица дополнительных сведений относящихся к задаче, добавляемая любым пользователем и представляемая в хронологическом по времени добавления к задаче порядке. Комментарий обычно содержит описание этапов прогресса работы, пояснения к требованиям задачи, описания трудности или полезные заметки к задаче, оставленные автором (держателем), исполнителем задачи или любым другим пользователями, которому доступен этот функционал для конкретной задачи. Несмотря на незначительные возможности представления информации, зачастую именно комментарии позволяют уточнить требования, поделиться обновлениями относительно хода выполнения задачи или быстро изменить требования.

## 1.4 Системы управления базами данных

Основным принципом организации баз данных является совместное хранение данных и их описания. Одна и та же база данных может быть использована для решения многих прикладных задач. Наличие метаданных и возможность информационной поддержки решения многих задач – это принципиальные отличия базы данных от любой другой совокупности данных, расположенных во внешней памяти ЭВМ. [6]

### 1.4.1 Предметная область информационной системы

Предметная область информационной системы рассматривается как совокупность реальных процессов и объектов (сущностей), представляющих интерес для ее пользователей. Каждая из сущностей предметной области обладает определенным набором свойств (атрибутов), среди которых можно выделить существенные и малозначительные. Признание какого-либо свойства существенным носит относительный характер [7].

Между сущностями предметной области могут существовать связи, имеющие различный содержательный смысл. Связи могут быть факультативными или обязательными. Если вновь порожденная сущность одного из типов оказывается по необходимости связанной с сущностью другого типа, то между этими типами сущностей есть обязательная связь. Иначе связь является факультативной.

Совокупность типов сущностей и типов связей между ними характеризует структуру предметной области. Собственно данные представлены экземплярами сущностей и связей между ними. Данные экземпляров сущностей и связей хранятся в базе данных информационной системы, а описание типов сущностей и связей является метаданными.

Множества экземпляров сущностей, значения атрибутов сущностей и экземпляры связей между ними могут изменяться во времени. Поэтому каждому моменту времени можно сопоставить некоторое состояние предметной области. Состояния ПО должны подчиняться совокупности правил, которые характеризуют семантику предметной области. В базе данных эти правила могут быть заданы с помощью так называемых ограничений целостности (консистентности), которые накладываются на атрибуты сущностей, типы сущностей, типы связей и/или их экземпляры. Фактически, ограничения целостности – это правила, которым должны удовлетворять значения данных в БД [6].

### 1.4.2 Модель данных

Модель данных – это совокупность правил порождения структур данных в базе данных, операций над ними, а также ограничений целостности, определяющих допустимые связи и значения данных, последовательность их изменения. Модель данных состоит из трёх частей [6]:

1. Набор типов структур данных. Здесь можно провести аналогию с языками программирования, в которых тоже есть предопределенные типы структур данных.

2. Набор операторов или правил вывода, которые могут быть применены к любым поддерживаемым типам данных конкретной реализации базы данных, чтобы находить, выбирать или преобразовывать информацию, содержащуюся в любых частях этих структур в любых комбинациях. Такими операциями являются: создание и модификация структур данных, внесение новых данных, удаление и модификация существующих данных, поиск данных по различным условиям.

3. Набор общих правил целостности, которые прямо или косвенно определяют множество непротиворечивых состояний базы данных и/или множество изменений её состояния.

### 1.4.3 Типы моделей данных

Рассмотрим три наиболее распространенных типа моделей данных: иерархическую, сетевую, реляционную и объектно-ориентированную [8]:

1. Иерархическая модель данных. Иерархическая структура представляет совокупность элементов, связанных между собою по определенным правилам. Объекты, связанные иерархическими отношениями, образуют ориентированный граф. К основным понятиям иерархической структуры относятся: уровень, элемент (узел), связь. Иерархическую модель организовывает данные в виде древовидной структуры. Узел - это совокупность атрибутов данных, которые описывают некоторый объект. На схеме иерархического дерева узлы имеют вид вершин графа.
2. Сетевая модель означает представление данных в виде произвольного графа. Достоинством сетевой и иерархической моделей данных является возможность их эффективной реализации по показателям затрат памяти и оперативности. Недостатком сетевой модели данных является высокая сложность и жесткость схемы БД, построенной на ее основе.
3. Реляционная модель данных. Понятие реляционной модели связано с разработками известного американского специалиста в области систем баз данных Э.Ф. Кодда. Эти модели характеризуются простотой структуры данных, удобной для пользователя формой представления в виде таблиц и возможностью использования аппарата алгебры отношений и реляционного исчисления для обработки данных.

## 1.5 Вывод из исследовательского раздела

Была проанализирована предметная область проекта, а также проведено подробное сравнение различных существующих решений. Ни одно из представленных решений не обладает возможностями гибкого расширения функционала на уровне кода программного продукта и создания настраиваемой и кросс-платформенной клиентской части приложения.

Для решения обозначенной проблемы, необходимо разработать систему управления данными и процессами, способную к гибким изменениям на уровне программного кода и интеграцией с любым типом клиентской платформы. Задача представляется в виде создания готового решения, которое будет иметь возможность в кратчайшие сроки быть измененным под конкретные цели, задачи и процессы, имеющие место в некоторой структуре.

Программный продукт, разрабатываемый в данной работе, должен включать в себя базу данных для хранения сущностей предметной области: Пользователей, Команды, Доски и Задачи Реляционная модель данных и связей наиболее близка к рассматриваемой области, что упрощает создание структуры базы данных, используемой в проекте.

# 

# 2. Конструкторский раздел

Для перехода к разработке программного обеспечения сначала необходимо сформулировать требования к разрабатываемой программе, описать схему отношения сущностей предметной области, выделить ключевые объекты и определить их представление в базе данных, спроектировать приложение и структуру базы данных, а также описать бизнес-процесс взаимодействия с программным продуктом пользователями с разными ролями.

## 2.1 Проектирование структуры и отношений сущностей

На рисунке 2.1 приведена концептуальная схема «Сущность-Связь» для представленных в аналитическом разделе сущностей предметной области в проектируемой базе данных. Помимо основных сущностей, также присутствует сервисная сущность аудита для создания записи об изменениях полей сущности Задачи. Создается соответствующая запись этой сущности при обновлении строки таблицы Задач при помощи триггера. В ней фиксируется время изменения и задача, которая была изменена.

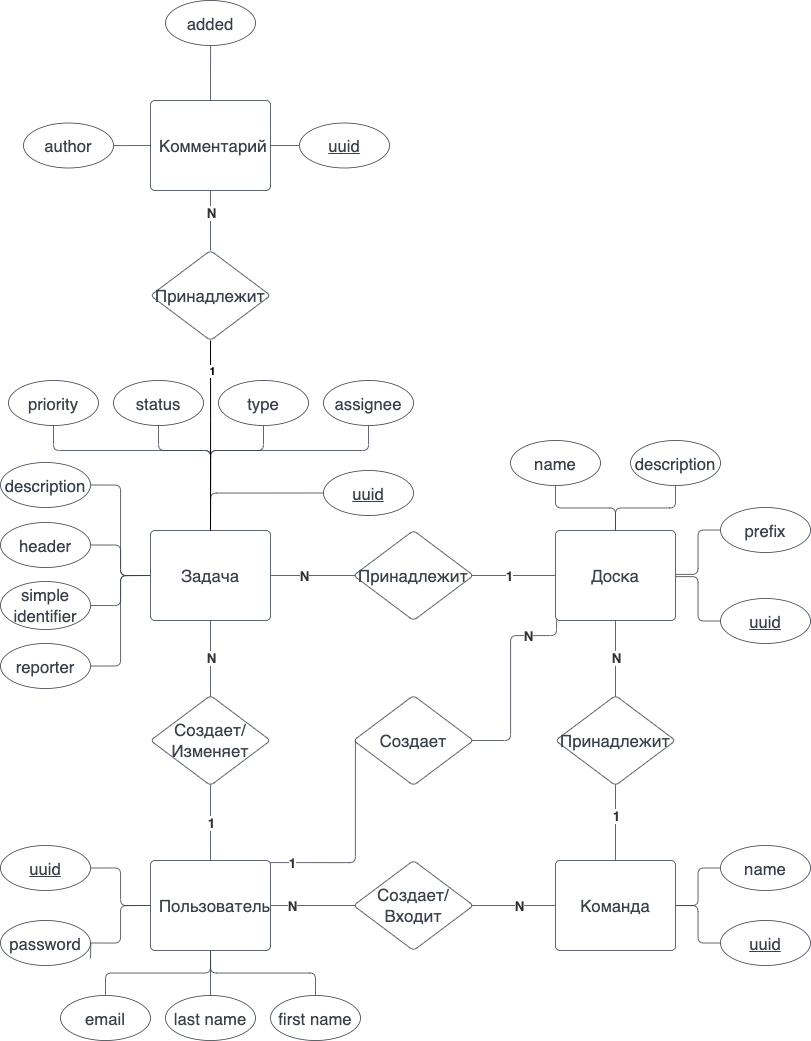


Рисунок 2.1. ER-диаграмма сущностей базы данных

## 

## 2.3 Внешние дополнительные хранилища

Для реляционных баз наиболее распространенным средством оптимизации являются сохранения «горячих данных» — данных, к которым наиболее часто обращается клиент в данный период времени, в оперативной памяти, тем самым на порядки ускоряя чтение таких данных. Данный процесс носит название «кэширование»[10]. В данном случае имеет смысл в качестве кэша рассматривать исключительно L2-кэш, в не зависимости от способа хранения данных: табличном (реляционным) или «ключ-значение». Если в подобный кэш данные были выгружены из основного хранилища, то все обращения сначала адресуются в хранилище кэша и только потом, если не все обращения были удовлетворены данными в L2-кэше, оставшиеся инструкции будут отправлены новым запросом уже в основное хранилище. В случае, если к некоторым данным не было обращений в течении некоторого заданного промежутка времени, они вытесняются из кэш-хранилища новыми «горячими данными» — подобная стратегия ротации используется в целях поддержании актуальности данных, находящихся в кэше и носит название — «истекающий кэш» (от англ. expirable cache).

Для достижения фрагментированности хранимых данных, зачастую используется регионирование кэша. В данном случае, истечение кэша возможно отследить для всего региона и обновить все данные в конкретном регионе.

Использование подобного механизма для оптимизации скорости обращений к «горячим данным» имеет смысл использовать, например, при работе с сущностями Пользователей, так как подавляющее количество запросов ожидается на взаимодействие с сущностями, связанными с представлением Пользователя в базе данных.

## 2.4 Стратегия кэширования

Стратегии кэширования на примере Hibernate ORM от разработчика Red Hat [11]:

1. Read Only — используется для данных, которые читаются, но не изменяются. Защищен от побочных эффектов при параллельного обращении или взаимодействии в кластеризованном окружении.
2. Read / Write — используется для регулярных чтений и записи. В данном случае от реализации кэш-хранилища требуется поддержка блокировок. Это требуется для поддержки не сериализованных уровней изоляции транзакций при обращении к кэшу, так как при использовании данной стратегии может иметь место параллельное обращение к одним и тем же данным.
3. Nonstrict read / write — аналогично используется для регулярных чтений и записи. В данном случае от реализации кэш-хранилища не требуется поддержка блокировок. При использовании данной стратегии может иметь место параллельное обращение к одним и тем же данным в кэш-хранилище. Во избежания побочных эффектов параллельной модификации данных, ответственность за управление обращениями к кэш-хранилищу переходит к самому разрабатываемому приложению.
4. Transactional — используется для регулярных чтений и записи. В данном случае от реализации кэш-хранилища требуется поддержка транзакций. При использовании данной стратегии, управление блокировками и параллельным доступом к данным делегируется реализации кэш-хранилища, которое должно реализовывать и использовать механизм транзакций и поддерживать их изоляцию.

Вследствие того, что приложение предполагается использовать с сервером приложений, имеющим значительное количество работающих потоков, наиболее рациональным будет использование Read / Write стратегии для достижения наибольшей производительности при параллельном обращении к кэш-хранилищу, а также независимости от реализации кэш-хранилища.

## 2.2 Ролевая модель

В ролевой модели [9] операции, которые необходимо выполнять в рамках какой-либо служебной обязанности пользователя системы, группируются в набор, называемый «ролью».

В целях максимизации защищенности и поддержания согласованности прав доступа между разрабатываемым приложением и базой данных, ролевая модель базы данных соответствует таковой на уровне приложения в доступности предоставляемых функциональных возможностях конечному пользователю программного продукта.

В разрабатываемой системе выделены три роли, каждой из которых со- ответствует конкретный функционал системы, при этом каждая роль расширяет доступный функционал роли с меньшим доступным функционалом:

1. Участник — этой роли соответствует функционал авторизации, регистрации, вступления в команду, просмотра и изменения задач и комментариев к ним с тех досок, которыми владеют команды, которым принадлежит пользователь.
2. Владелец продукта — этой роли доступен весь функционал роли «Участник», а также возможности создавать команды и доски.
3. Администратор — этой роли доступен весь функционал остальных ролей, а также удаления пользователей, досок, команд.

Каждый пользователь должен иметь запись о себе в базе данных и четко указанную роль в ней. В случае, если входящий запрос на конечную точку обязан определенную роль пользователя, клиентское приложение осуществляет проверку прав доступа у данного пользователя. В случае, если прав пользователя достаточно, ему функционал, предоставляемый конечной точкой.

## 2.5 Проектирование клиентского приложения для СУБД

На рисунке 2.2 представлена модель бизнес-процессов в нотации BPMN в соответствии с выделенными ранее ролями. Также выделены подпроцессы в целях декомпозиции диаграммы.

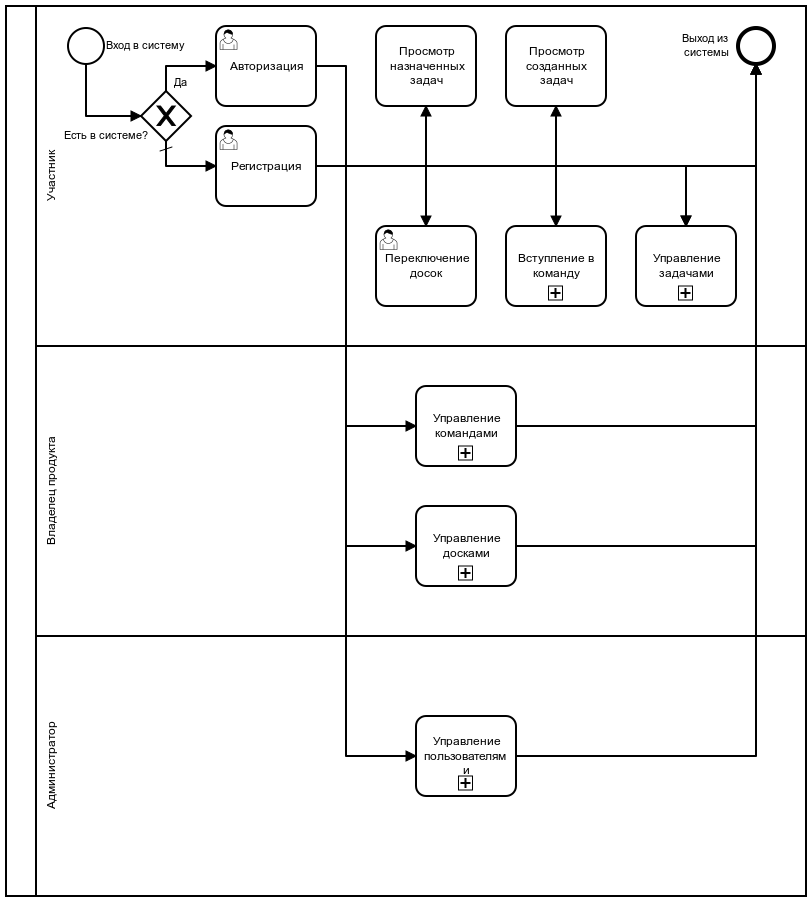


Рисунок 2.2. Модель общего бизнес-процесса в нотации BPMN

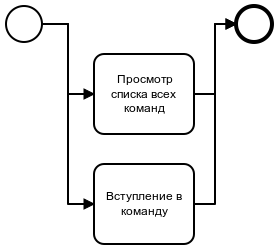


Рисунок 2.2.1. Подпроцесс вступления в команду в нотации BPMN

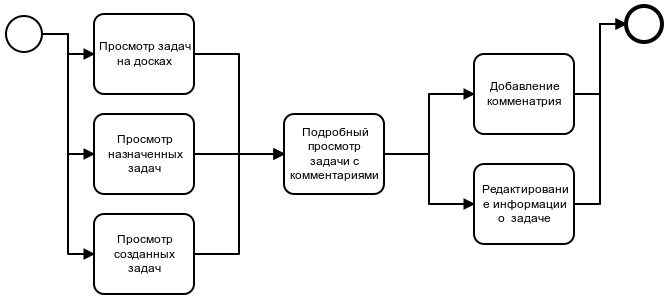


Рисунок 2.2.2. Подпроцесс управления задачами в нотации BPMN

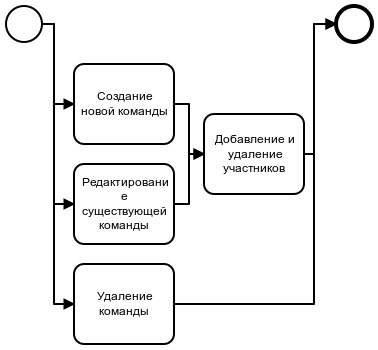


Рисунок 2.2.3. Подпроцесс управления командами в нотации BPMN

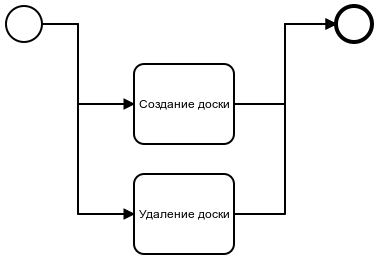


Рисунок 2.2.4. Подпроцесс управления досками в нотации BPMN

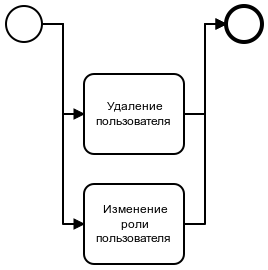


Рисунок 2.2.5. Подпроцесс управления пользователями в нотации BPMN

На рисунках 2.2.1 — 2.2.5 представлены модели вложенных бизнес-процессов в нотации BPMN.

## 2.6 Вывод из исследовательского раздела

Проектируемая база данных хранит информацию, получаемую из web- приложения, реализующего базовую функциональность управления задачами в парадигме Agile. В базе данных выделен соответствующий набор сущностей, соответсвующий базовым бизнес-сущностям: Пользователь, Команда, Доска, Задача и Комментарий, приведенные на ER-диаграмме (рисунок 2.1). Для самой СУБД выделено три роли: «Участник», «Владелец продукта» и «Администратор», каждая из которых напрямую соответствует дорожке пула «пользователь», присутствующей в бизнес-процессе (рисунок 2.2).

Использование сторонних кэширующих хранилищ, в том числе как L2-кэш, накладывает некоторые ограничения на размерность и связанность данных, однако, при правильном использовании, L2-кэш позволяет значительно ускорить доступ к данным и их изменение.

# 3. Технологический раздел

## 3.2 Обоснование выбора языка, среды и платформы программирования

Для реализации клиентского приложения в рамках поставленной задачи был выбран язык Java, 11 версии. Это объектно-ориентированный язык, 11 версия которого распространяется по лицензии GNU GPL. Был выбран по следующим причинам.

1. Высокая скорость выполнения.
2. Поддержка ООП.
3. Широкий функционал для разработки web-приложений.
4. Богатый набор возможностей для взаимодействия с кэш-хранилищами и различными СУБД.

Благодаря возможностям настройки размера доступной программе памяти в heap, возможность работы с off-heap памятью, и поддержки различных реализаций кэш-хранилищ, возможно произвести полноценный анализ производительности программы в разных условиях работы.

Клиентское приложение должно использовать ORM Hibernate [12] для обеспечения поддержки L2-кэша и возможности управления транзакциями JPA [13]. В качестве кэш-хранилища выбран HazelCast за счет гибких возможностей распределения партиций кэш-хранилища и продвинутых средств мониторинга.

В качестве СУБД, была выбрана база данных MySQL — одновременно высокопроизводительное при небольших объемах данных и простое решение.

## 3.3 Реализация клиентского приложения

Клиентское приложение должно удовлетворять следующим требованиям:

* в программном продукте обязано присутствовать взаимодействие с разработанной структурой базы данных и самой СУБД;
* программа не должна аварийно завершаться;
* программа не должна задействовать иррационально большой объем оперативной памяти или времени и ресурсов процессора.

Модельные классы для сущностей соответствуют сущностям базы данных, представленным на рисунке 2.1.

На листинге 3.1 представлены базовый модельный класс, наследуемый всеми остальных классами сущностей. Его функционал включает генерацию строкового идентификатора формата UUID с длинной 36 символов на стороне клиентского приложения в целях дополнительной проверки на вставку сущности в базу данных и соответствия идентификатора в базе данных и кэш-хранилище, куда прежде будет выполнена вставка.

Листинг 3.1: Модельный класс базовой сущности

| public abstract class BaseEntity {  @Id  @GeneratedValue(generator = "UUID")  @GenericGenerator(name = "UUID",  strategy = "org.hibernate.id.UUIDGenerator")  @Column(name = "uuid", updatable = false, nullable = false)  String uuid;  } |
| --- |

На листинге 3.2 представлен модельный класс сущности «Пользователь».

Листинг 3.2: Модельный класс сущности «Пользователь»

| public class User extends BaseEntity {  String firstName;  String lastName;  String email;  String password;  Boolean active;  @ManyToMany(mappedBy = "users")  Set<Team> teams;  @OneToMany(mappedBy = "reporter")  List<Task> reportedTasks;  @OneToMany(mappedBy = "assignee")  List<Task> assignedTasks;  @OneToMany(mappedBy = "author")  List<Commentary> commentaries;  } |
| --- |

На листинге 3.3 представлен модельный класс сущности «Команда».

Листинг 3.3: Модельный класс сущности «Команда»

| public class Team extends BaseEntity {  String name;  @ManyToMany(fetch = FetchType.*EAGER*)  @JoinTable(name = "users\_teams")  Set<User> users;  @OneToMany(mappedBy="owner")  Set<Dashboard> dashboards;  } |
| --- |

На листинге 3.4 представлен модельный класс сущности «Доска».

Листинг 3.4: Модельный класс сущности «Доска»

| public class Dashboard extends BaseEntity {  String name;  String description;  String prefix;  @OneToMany(mappedBy="dashboard")  List<Task> tasks;  @ManyToOne(fetch = FetchType.*EAGER*)  @JoinColumn(name="owner\_team\_uuid", nullable=false)  Team owner;  } |
| --- |

На листинге 3.5 представлен модельный класс сущности «Задача».

Листинг 3.5: Модельный класс сущности «Задача»

| public class Task extends BaseEntity {  String simpleIdentifierPrefix;  Long simpleIdentifierNumber;  String header;  String description;  @Enumerated(EnumType.*STRING*)  TaskType type;  @Enumerated(EnumType.*STRING*)  TaskStatus status;  @Enumerated(EnumType.*STRING*)  TaskPriority priority;  @ManyToOne(fetch = FetchType.*LAZY*)  @JoinColumn(name="dashboard\_uuid", nullable=false)  Dashboard dashboard;  @ManyToOne(fetch = FetchType.*EAGER*)  @JoinColumn(name="reporter\_uuid", nullable=false)  User reporter;  @ManyToOne(fetch = FetchType.*EAGER*)  @JoinColumn(name="assignee\_uuid")  User assignee;  @OneToMany(mappedBy="task")  List<Commentary> commentaries;  public enum TaskPriority {  *MINOR*,  *NORMAL*,  *MAJOR*,  *BLOCKER*  }  public enum TaskType {  *STORY*,  *DEFECT*,  *IMPROVEMENT*,  *CHANGE\_REQUEST*  }  public enum TaskStatus {  *IDEA*,  *ANALYSIS*,  *READY\_FOR\_DEVELOPMENT*,  *IN\_DEVELOPMENT*,  *READY\_FOR\_QA*,  *IN\_QA*,  *DONE*  }  } |
| --- |

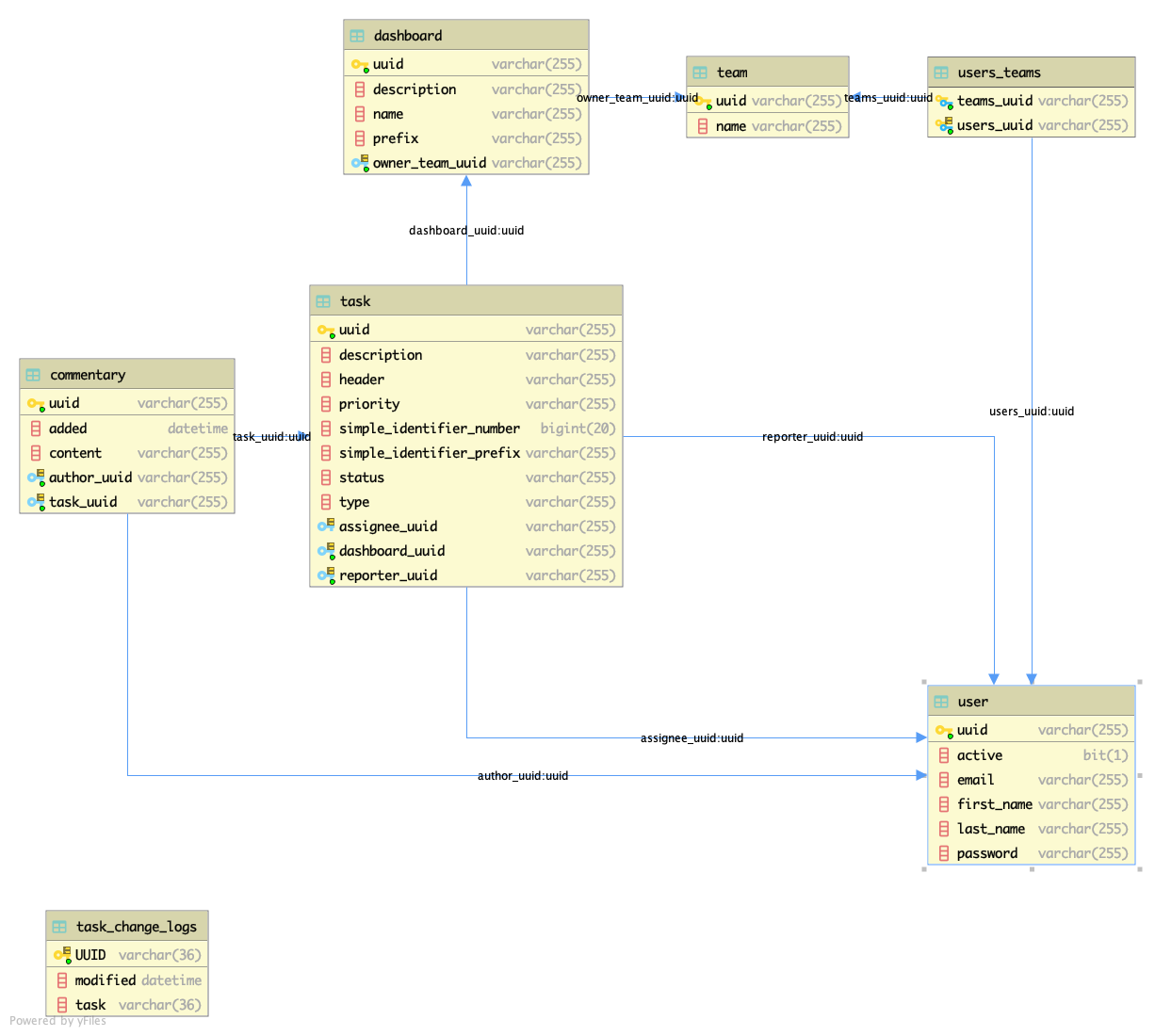
На листинге 3.6 представлен модельный класс сущности «Комментарий».

Листинг 3.6: Модельный класс сущности «Комментарий»

| public class Commentary extends BaseEntity {  String content;  LocalDateTime added;  @ManyToOne(fetch = FetchType.*EAGER*)  @JoinColumn(name="author\_uuid", nullable=false)  User author;  @ManyToOne(fetch = FetchType.*LAZY*)  @JoinColumn(name="task\_uuid", nullable=false)  Task task;  } |
| --- |

## 3.3 Реализация базы данных

Визуализация структуры базы данных с сущностями, приведенными на рисунке 2.1, а также служебной таблицей для аудита task\_change\_logs представлены на рисунке 3.1.

Рисунок 3.1. Визуализация структура базы данных со связями между сущностями.

На листинге 3.7 представлено описание SQL триггера, осуществляющего запись изменений в сущностях «Задача» в целях ведения аудита проекта.

Листинг 3.7: SQL триггер, осуществляющий запись изменений в сущностях «Задача»

| CREATE TRIGGER log\_task\_change  AFTER UPDATE ON task  FOR EACH ROW  INSERT INTO task\_change\_logs VALUES (*UUID*(), *NOW*(), OLD.uuid) |
| --- |

# 

# 4 Исследовательский раздел

## 4.1 Технические характеристики устройства

В Таблице 4.1 приведены технические характеристики устройства (ЭВМ), использованного для проведения экспериментов на базе разработанного программного обеспечения. Стоит заметить, что на ЭВМ используется технология Hyper-Threading, расширяющая количество потоков процессора до 4.

Таблица 4.1: Технические характеристики устройства, на котором проводилось тестирование программного обеспечения

| ОС | macOS 10.14.6 64-bit |
| --- | --- |
| ОЗУ | 2х4 Gb 1600 MHz DDR3 |
| Процессор | 2,3 GHz Intel Core i7 (2 ядра, 4 с технологией Hyper-Threading) |

## 

## 4.2 Авторизация пользователей с включенным кэшем

Задача эксперимента — наблюдение зависимости времени доступа к данным и количества чтения из кэш-хранилища при размере хранилища для 2710 экземпляров данных пользователя, 5420 регистрируемых и 5420 авторизирующихся в случайном порядке пользователей. Вначале выполнялось 5420 запросов на регистрацию пользователя, затем в случайном порядке (не исключающем возможности повторения обращения к уже авторизованным ранее) отправлялись запросы на авторизацию. При использовании стратегии помещения объектов в кэш, ровно половина (2710) данных пользователей, зарегистрировавшихся последними, находится в кэше.

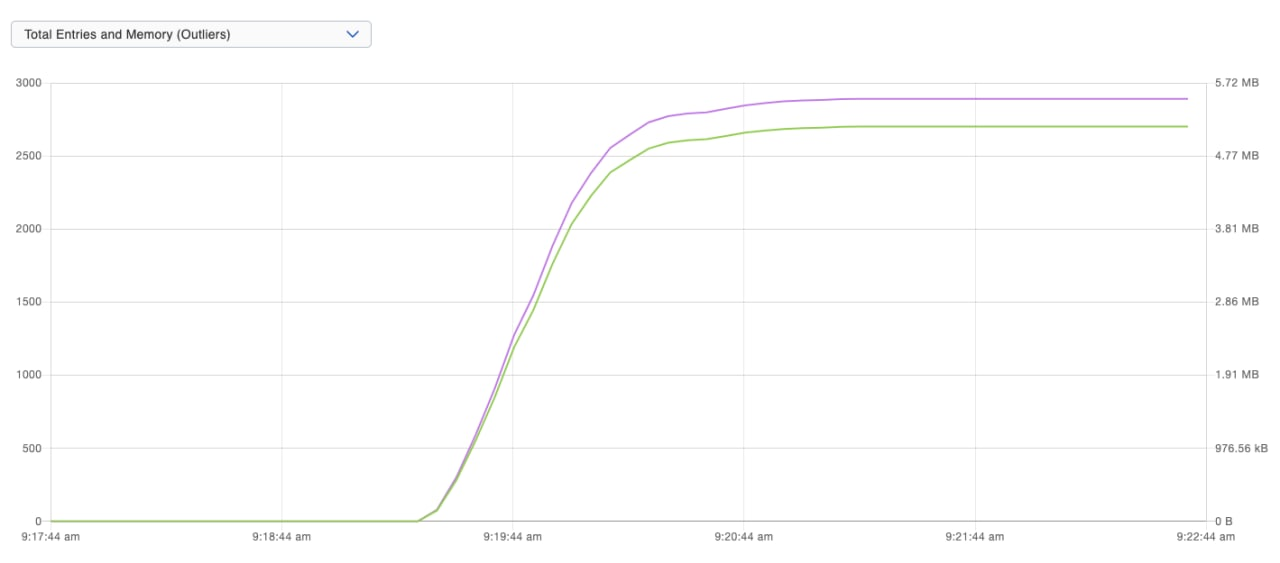
Вследствие того, что конфигурация количества партиций не поддерживается для используемой стратегии разбиения по регионам и взаимодействия с мониторинговым сервисом HazelCast Management Center, количество запросов выбрано кратным 271 — количеством партиций хранилища по умолчанию.

Результаты исследования представлены в таблице 4.2.

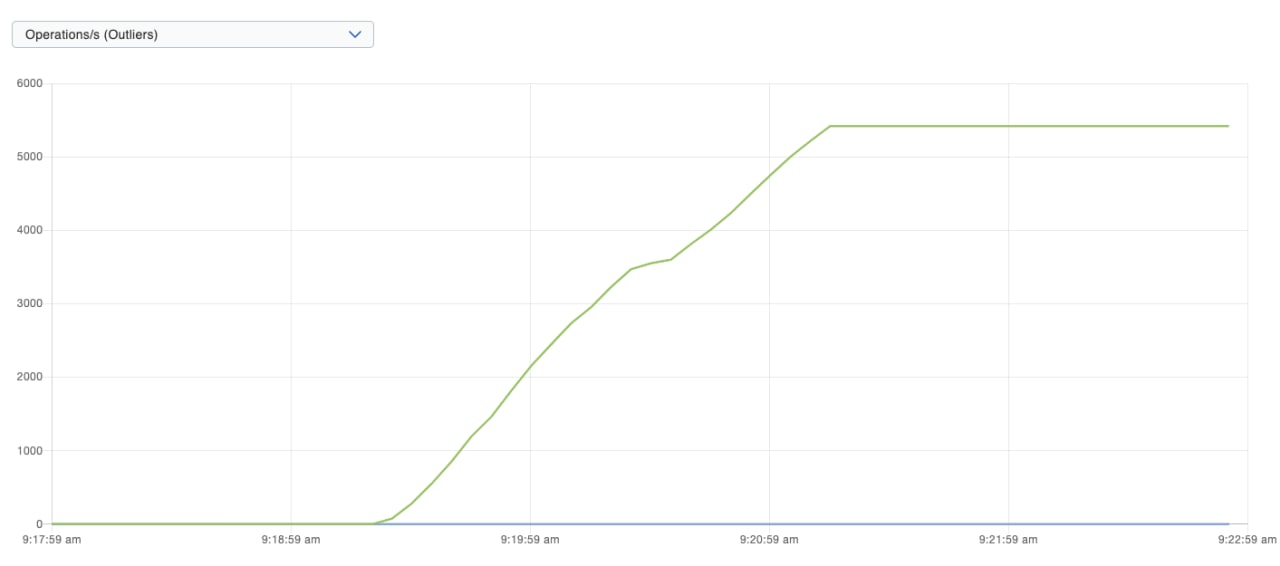
Таблица 4.2. Время выполнения запроса

| Действие | Время выполнения, мс |
| --- | --- |
| Регистрация | 113400 |
| Авторизация | 95040 |

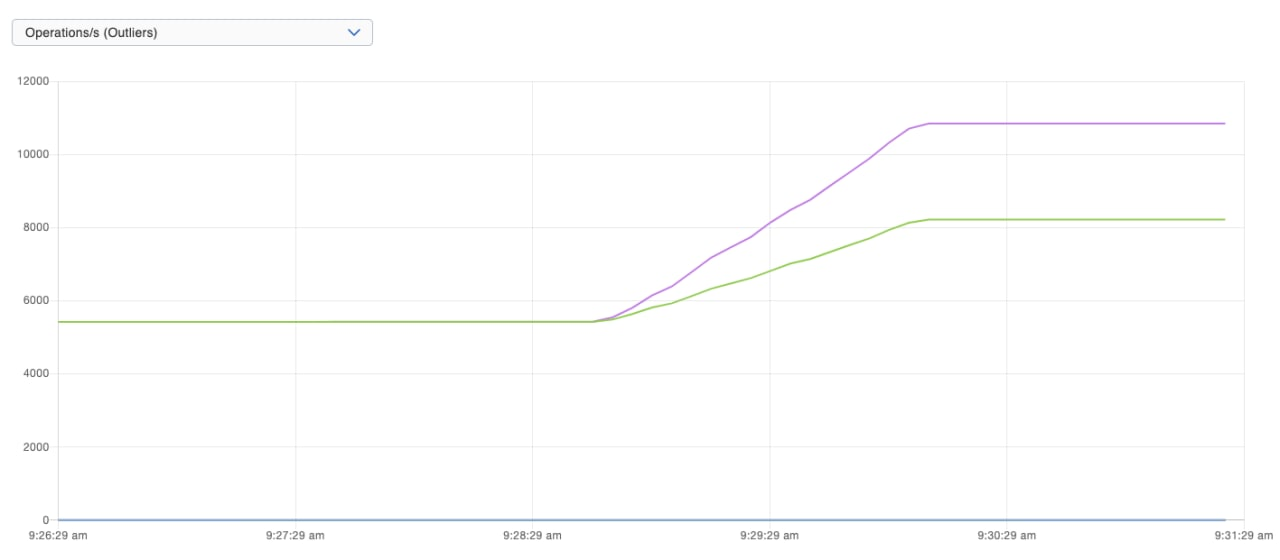
На рисунке 4.2.1. представлена зависимость количества сущностей и используемой памяти в кэше от времени при регистрации пользователей.

Рисунок 4.2.1. Зависимость количества сущностей (зеленый) и используемой памяти (фиолетовый) в кэше от времени при регистрации пользователей, генерируемая программой мониторинга HazelCast Management Center.

На рисунке 4.2.2. представлена зависимость количества операций добавления данных в кэш от времени при регистрации пользователей.

Рисунок 4.2.2. Зависимость количества операций добавления данных в кэш от времени при регистрации пользователей, генерируемая программой мониторинга HazelCast Management Center.

На рисунке 4.2.3. представлена зависимость количества операций добавления и чтения данных в кэше от времени при авторизации пользователей.

Рисунок 4.2.3. Зависимость количества операций добавления (зеленый) и чтения (фиолетовый) данных в кэше от времени при авторизации пользователей, генерируемая программой мониторинга HazelCast Management Center.

## 4.3 Повторная авторизация пользователей с включенным кэшем

Задача эксперимента — наблюдение зависимости времени доступа к данным и количества чтения из кэш-хранилища при размере хранилища для 2710 экземпляров данных пользователя, 5420 регистрируемых и 542 авторизирующихся по 10 раз в случайном порядке пользователей. Вначале выполнялось 5420 запросов на регистрацию пользователя, затем в случайном порядке (не исключающем возможности повторения обращения к уже авторизованным ранее) отправлялись запросы на авторизацию. При использовании стратегии помещения объектов в кэш, ровно половина (2710) данных пользователей, зарегистрировавшихся последними, находится в кэше.

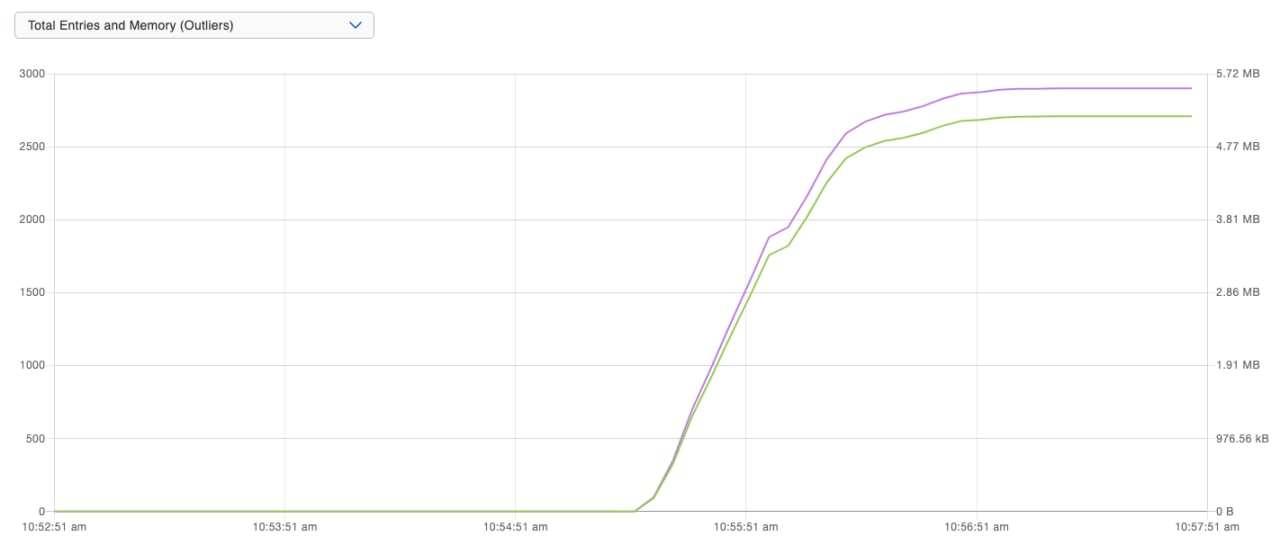
Вследствие того, что конфигурация количества партиций не поддерживается для используемой стратегии разбиения по регионам и взаимодействия с мониторинговым сервисом HazelCast Management Center, количество запросов выбрано кратным 271.

Результаты исследования представлены в таблице 4.3.

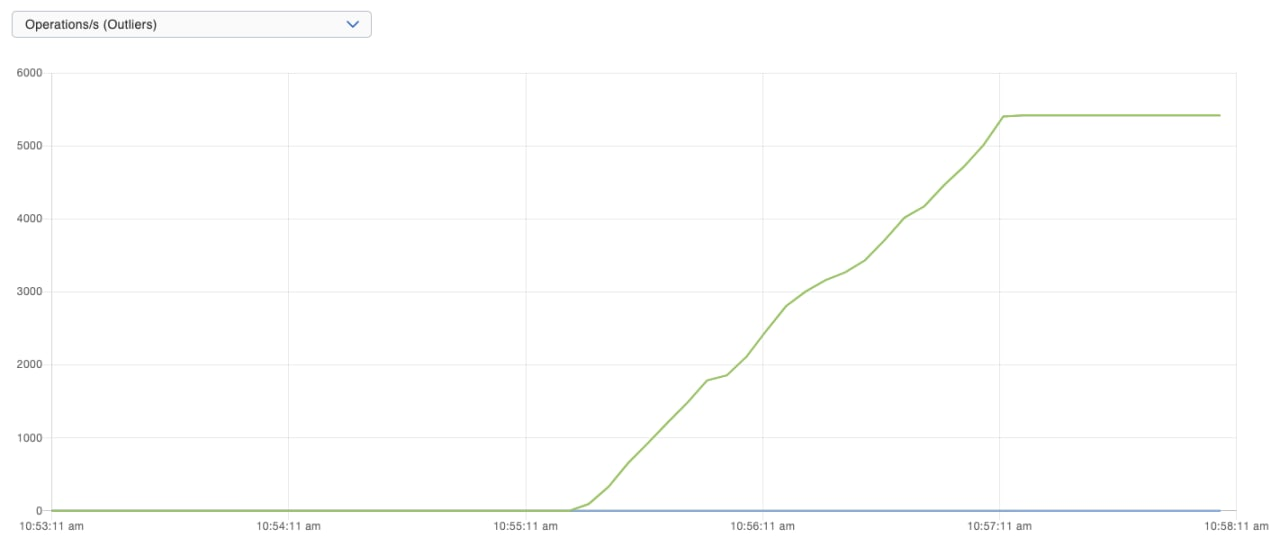
Таблица 4.3. Время выполнения запроса

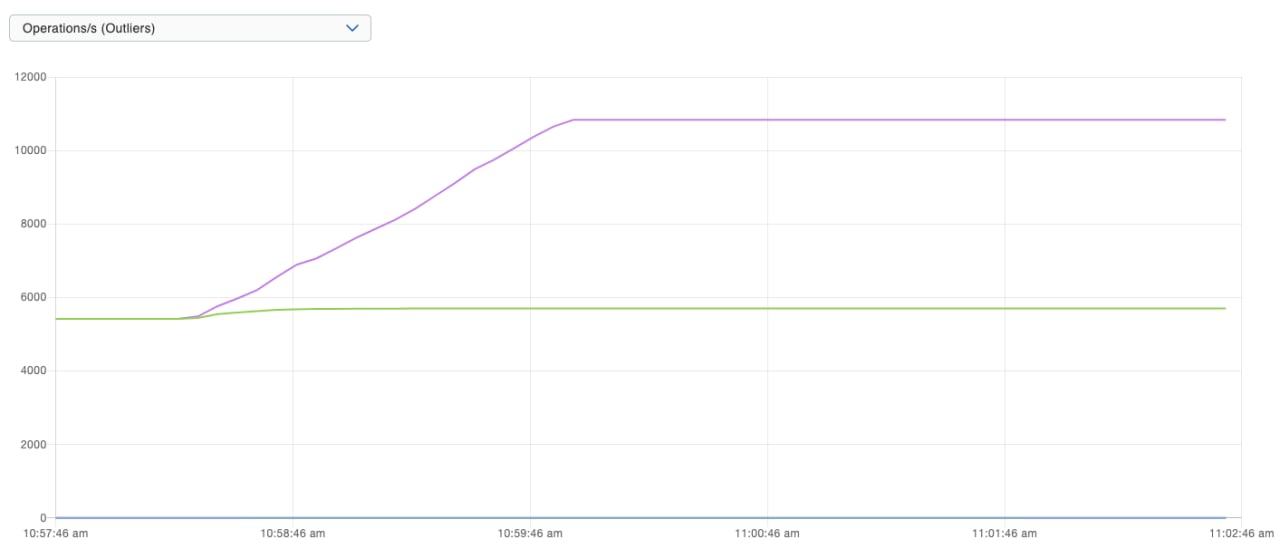
| Действие | Время выполнения, мс |
| --- | --- |
| Регистрация | 111760 |
| Авторизация | 88080 |

На рисунке 4.3.1. представлена зависимость количества сущностей и используемой памяти в кэше от времени при регистрации пользователей.

Рисунок 4.3.1. Зависимость количества сущностей (зеленый) и используемой памяти (фиолетовый) в кэше от времени при регистрации пользователей, генерируемая программой мониторинга HazelCast Management Center.

На рисунке 4.3.2. представлена зависимость количества сущностей и используемой памяти в кэше от времени при регистрации пользователей.

Рисунок 4.3.2. Зависимость количества операций добавления данных в кэш от времени при регистрации пользователей, генерируемая программой мониторинга HazelCast Management Center.

Рисунок 4.3.3. Зависимость количества операций добавления (зеленый) и чтения (фиолетовый) данных в кэше от времени при авторизации пользователей, генерируемая программой мониторинга HazelCast Management Center.

## 4.4 Повторная авторизация пользователей без кэша

Задача эксперимента — наблюдение зависимости времени доступа к данным и количества чтения из кэш-хранилища при размере хранилища для 2710 экземпляров данных пользователя, 5420 регистрируемых и 5420 авторизирующихся в случайном порядке пользователей. Вначале выполнялось 5420 запросов на регистрацию пользователя, затем в случайном порядке (не исключающем возможности повторения обращения к уже авторизованным ранее) отправлялись запросы на авторизацию. В данном эксперименте использование L2 кэша было отключено, однако для поддержания соответствующей нагрузки на систему, кэш-хранилище и мониторинговые сервисы оставались включенными.

Результаты исследования представлены в таблице 4.4.

Таблица 4.4. Время выполнения запроса

| Действие | Время выполнения, мс |
| --- | --- |
| Регистрация | 104760 |
| Авторизация | 98080 |

## 4.5 Вывод из исследовательского раздела

Из проведенных исследований можно сделать вывод о том, что в случае добавления пользователей в базу данных и кэш, за счет синхронизации вставки между двумя хранилищами с соответствующими блокировками ресурсов, кэш-хранилище создавало дополнительные издержки на каждую операцию и, как следствие, система демонстрировала меньшую общую производительность. В случае авторизации, не было получено значительного прироста производительности при доступе к данным всех пользователей в случайном порядке в экспериментах с выключенным кэшом относительно включенного. Однако в случае, если чтений было более одного, наблюдается прирост производительности за счет большего количества объектов, находящихся в L2-кэше.

# 

# 

# 

# 

# ЗАКЛЮЧЕНИЕ

Цель проекта была достигнута — была спроектирована и реализована структура базы данных для хранения и анализа данных представляющих сущности задач продуктовой команды. В дополнение, реализована программа и ее интерфейс для предоставления пользователю возможности просмотра задач, находящихся в базе данных. Предусмотрены ролевая модель и хранимый триггер, а также возможность авторизации в интерфейсе, создания и изменения задач.

Все поставленные задачи были решены:

* проведен анализ существующих решений, используемой методологии и ее реализаций;
* проведен анализ предметной области и выделены основные сущности;
* выбрана подходящая программная платформа для реализации поставленной задачи и тип системы управления базы данных;
* спроектирована и реализована структура базы данных для выбранного типа базы данных, программное обеспечение и его интерфейс.

# 

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Д.Н. Лопатин «Agile - новый уровень мотивации в менеджменте» — Омск, Омский государственный технический университет, 2012г. — 32 с.
2. Д.В. Карпов «Гибкая методология разработки программного обеспечения» — Нижний Новгород, Нижегородский госуниверситет им. Н.И. Лобачевского, 2011г. — 47 с.
3. Е. Г. Потапова «Agile-подход в государственном управлении» — М.: РАНХиГС, 2019. — 162 с.
4. Руководства по продукту и учебные материалы Atlassian Jira [Электронный ресурс] *—* Режим доступа: https://www.atlassian.com/ru/software/jira/guides/getting-started/overview
5. Начало работы с Trello [Электронный ресурс] *—* Режим доступа: https://trello.com/ru/guide/trello-101
6. «Redmine Wiki» [Электронный ресурс] *—* Режим доступа: https://www.redmine.org/
7. И.П. Карпова «Базы данных. Учебное пособие.» — Московский государственный институт электроники и математики (Технический университет). — М., 2009.
8. «Компьютерные данные: типы данных, обработка и управление» — Луганский государственный медицинский университет
9. Leonard J. L., D. Elliott B. «Secure Computer Systems: A Mathematical Model» // MITRECorporation Technical Report 2547, Volume II. — 1973.
10. «What is Caching?» [Электронный ресурс] *—* Режим доступа: https://azure.microsoft.com/en-gb/overview/what-is-caching/#overview
11. «Hibernate: The second level cache» [Электронный ресурс] *—* Режим доступа: https://docs.jboss.org/hibernate/core/3.3/reference/en/html/performance.html#performance-cache-mapping
12. «Hibernate Official Documentation» [Электронный ресурс] *—* https://hibernate.org/orm/documentation/6.0/
13. Oliver G., Thomas D., Christoph S., Mark P., Jay B., «Spring Data JPA Reference Documentation» — 2022.