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 «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К КУРСОВОЙ РАБОТЕ

НА ТЕМУ:

«База данных для автосервиса»

Студент ИУ7-61Б _____ Мицевич М. Д.
(Группа) (Подпись, дата) (И. О. Фамилия)

Руководитель курсовой работы _____ Тассов К. Л.
(Подпись, дата) (И. О. Фамилия)

2022 г.

РЕФЕРАТ

Курсовая работа представляет собой реализацию приложения, отслеживающего состояние склада. Готовое приложение предоставляет возможность получения информации о автосервисе через REST API.

Приложение реализовано на языке программирования C++ с использованием Boost в качестве фреймворка для получения запросов через REST API.

Ключевые слова: PostgreSQL, Redis, C++, SQL, корутина.

Расчетно-пояснительная записка к курсовой работе содержит 30 страниц, 9 иллюстраций, 2 таблиц, 12 источников, 1 приложение.

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей расчетно-пояснительной записке применяют следующие сокращения и обозначения.

СУБД — система управления базами данных

SQL — Structured Query Language

NoSQL — Not only Structured Query Language

ВВЕДЕНИЕ

В современном мире множество людей передвигается на автомобиле, поэтому в каждом крупном городе существует хотя бы один сервис для починки транспортных средств. Большинство таких мастерских не являются дилерскими и нуждаются в приложении, способном отслеживать состояние склада с запчастями и историю закупок и продаж.

Целью данной работы является разработка приложения и создание базы данных для хранения информации о состоянии склада с автозапчастями. Для достижения поставленной цели требуется выполнить следующие задачи:

- формализовать задание, определить необходимый функционал;
- провести анализ СУБД;
- спроектировать базу данных, описать ее сущности и связи;
- спроектировать приложение для доступа к БД;
- реализовать интерфейс для доступа к базе данных;
- реализовать программное обеспечение, которое позволит получить доступ к данным по средствам REST API [1].
- провести эксперимент на сравнение двух различных СУБД.

1 Аналитический раздел

1.1 Постановка задачи

Необходимо разработать программу для отслеживания состояния склада с автозапчастями. В частности требуется хранить информацию о том, какой пользователь положил деталь на склад и какой ее забрал.

Некоторые запчасти можно заменить на подобные им от другого производителя, поэтому у пользователя должна быть возможность поиска на складе не только самой детали, но и ее аналогов.

1.2 Формализация данных

База данных должна содержать информацию о:

- деталях для автомобилей;
- производителях автозапчастей;
- текущем состоянии склада и истории его изменения;
- пользователях системы.

Информация о детали должна содержать:

- артикул;
- название на русском и английском языках;
- возможные варианты замены;
- идентификатор производителя.

Производитель автозапчасти характеризуется страной и названием.
Пользователь характеризуется следующими параметрами:

- Личные данные: имя, фамилия, дата рождения;
- уровень доступа к системе;
- данными авторизации: логин, пароль.

1.3 Типы пользователей

В системе должно быть разделение пользователей по уровню доступа к информации и возможностям ее изменения. Каждый пользователь должен авторизоваться прежде чем начать работу с приложением. Уровни доступа различных пользователей представлены на таблице 1.1

Таблица 1.1 – Уровни доступа различных пользователей

Тип пользователя	**Функционал**
Неавторизованный	Регистрация, авторизация
Клиент	Просматривать наличие запчастей на складе
Кладовщик	Добавлять запчасти на склад
Продавец	Забирать запчасти со склада
Администратор	Изменять уровни привилегий других пользователей, модифицировать информацию о запчастях, производителях и заменах, просматривать историю изменения склада.

Диаграмма вариантов использования представлена на рисунке 1.1.

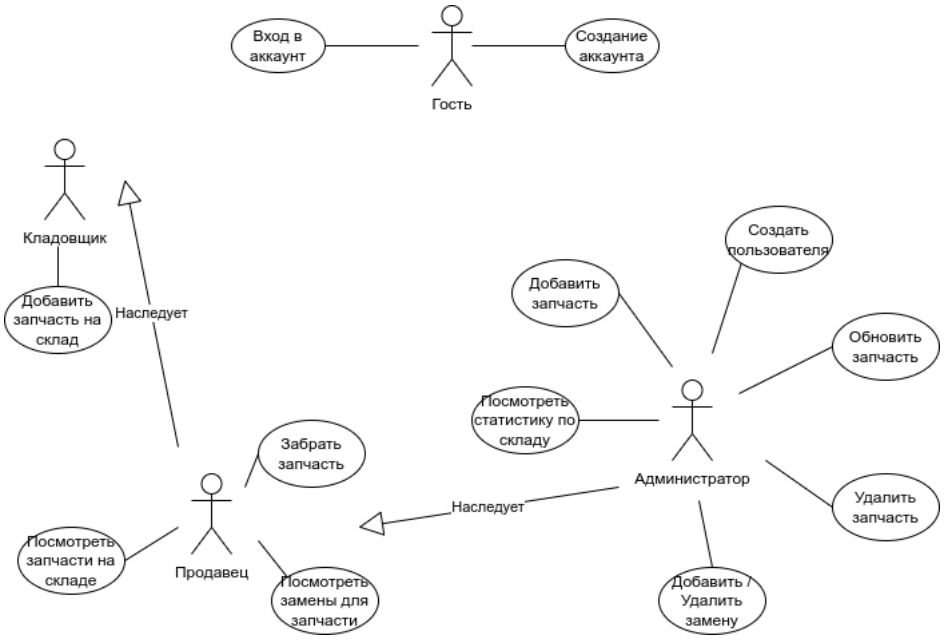

Рисунок 1.1 – Диаграмма вариантов использования

Диаграмма сущностей в нотации Чена представлена на рисунке 1.2

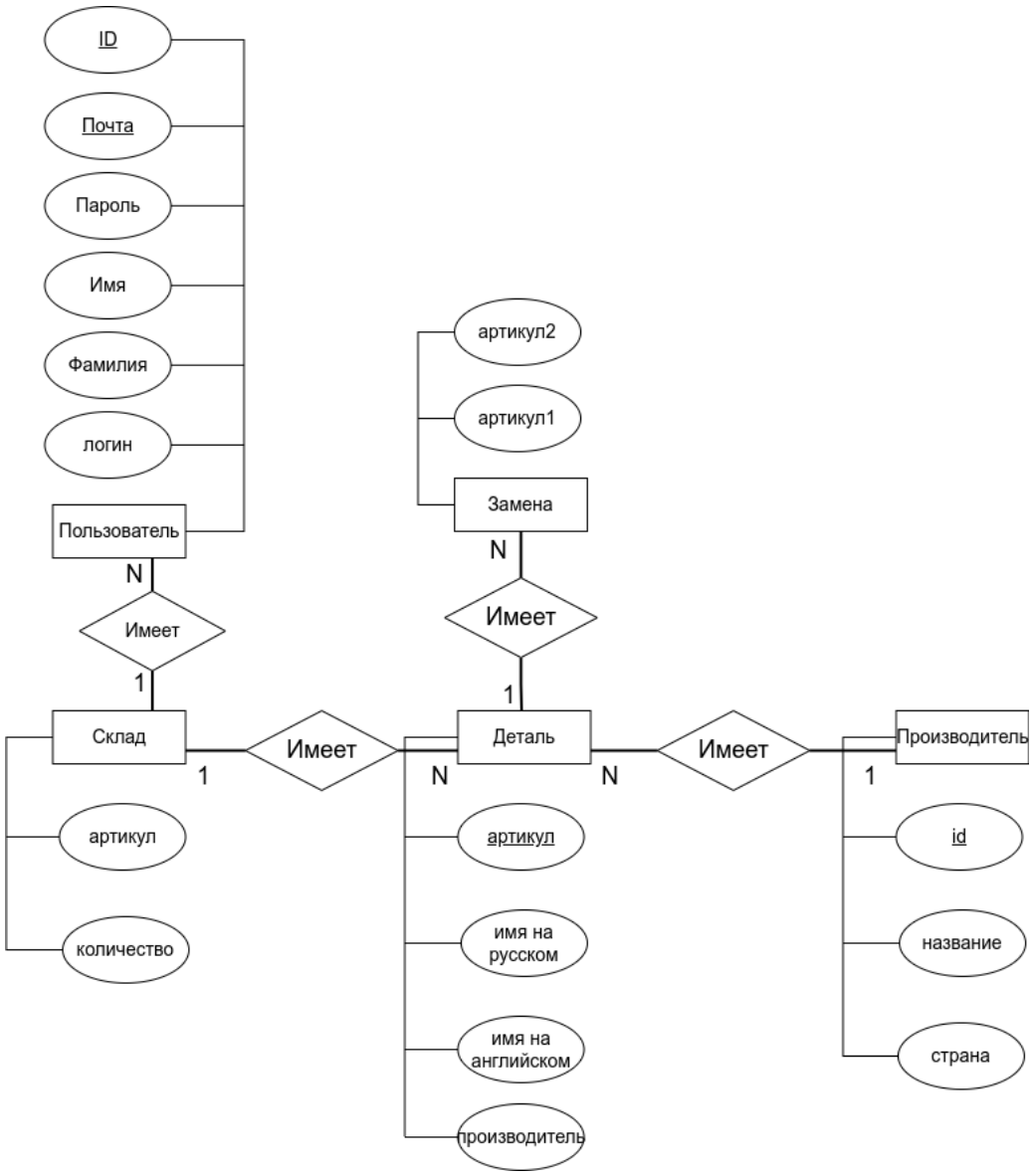

Рисунок 1.2 – Диаграмма сущностей в нотации Чена

1.4 Модели хранения данных

Модели данных — вероятно, важнейшая часть разработки программного обеспечения в силу оказываемого ими глубочайшего воздействия не только на процесс разработки, но и на наше представление о решаемой проблеме [2]. Можно выделить три основных модели хранения данных:

- иерархическая;
- сетевая;

- реляционная.

Иерархическая модель В иерархической модели данных используется представление базы данных в виде древовидной структуры, состоящей из объектов различных уровней. Между объектами существуют связи, каждый объект может включать в себя несколько объектов более низкого уровня. Такие объекты находятся в отношении предка к потомку, при этом возможна ситуация, когда объект-предок имеет несколько потомков, тогда как у объекта-потомка обязателен только один предок.

Сетевая модель В древовидной структуре последней у каждой записи была ровно одна родительская запись. В сетевой же у каждой записи могло быть несколько родительских. Например, в базе может быть одна запись с названием города со ссылкой на нее для каждого живущего там пользователя. В сетевой модели появилась возможность моделировать связи «многие-к-одному» и «многие-ко-многим».

Главным недостатком сетевой модели данных являются жесткость и высокая сложность схемы базы данных, построенной на основе этой модели. Так как логика процедуры выбора данных зависит от физической организации этих данных, то эта модель не является полностью независимой от приложения. Иначе говоря, если будет необходимо изменить структуру данных, то нужно будет изменять и приложение.

Реляционная модель Реляционная модель состоит из набора двумерных таблиц. При табличной организации отсутствует иерархия элементов. Таблицы состоят из строк – записей и столбцов – полей. На пересечении строк и столбцов находятся конкретные значения. Для каждого поля определяется множество его значений. За счет возможности просмотра строк и столбцов в любом порядке достигается гибкость выбора подмножества элементов.

В реляционной базе данных оптимизатор запросов автоматически принимает решение о том, в каком порядке выполнять части запроса и какие индексы использовать.

Выбор модели хранения данных Все сущности системы имеют четкую структуру, которая не меняется от объекта к объекту, поэтому для хранения данных подойдет реляционная модель.

1.5 Аутентификация пользователей в системе

Аутентификацию пользователей в системе можно проводить с использованием спецификации OAuth1 [3]. Такой подход предполагает следующую последовательность:

- пользователь отправляет запрос, в котором указывает свои логин и пароль;
- приложение ищет пользователя, с указанными параметрами в базе данных и, в случает успеха, возвращает токен, с использованием которого клиент может осуществлять последующие запросы.

В таком случае необходимо где-то хранить связь токена с идентификаторам пользователя, которому он принадлежит. Для таких целей подойдет In Memory ключ-значение СУБД, которая обеспечивает быстрый поиск за счет того, что все данные хранятся в оперативной памяти. Главной проблемой таких СУБД является возможность утери данных, к примеру, при отключении электропитания, что не является критичным для хранения токенов, так как пользователю будет достаточно еще раз заполнить форму с логином и паролем для восстановления сессии.

В качестве СУБД для хранения сессий авторизованных пользователей может подойти Redis [4].

2 Конструкторский раздел

2.1 Проектирование базы данных

В соответствии с выделенными в разделе 1.2 сущностями можно выделить следующие таблицы базы данных для хранения:

- запчастей;
- производителей деталей;
- аналогов автозапчастей;
- работников сервиса;
- запчастей на складе;
- истории добавления и удаления деталей на склад;
- токенов авторизации пользователей системы.

В таблице запчастей надо хранить:

- артикул запчасти для идентификации – primary key, строковый тип;
- название на русском и английском языках – оба строкового типа;
- идентификатор производителя – foreign key, целочисленный.

В таблице производителей:

- целочисленный идентификатор;
- название и страна строкового типа.

В таблице замен:

- артикул детали, которую можно заменить – foreign key, строковый тип;
- артикул аналога – foreign key, стоковый тип.

В таблице состояния склада:

- артикул запчасти – foreign key, строковый тип;

- количество таких запчастей – целочисленный тип, может быть нулевым в случае, если деталь когда-то была на складе, но сейчас ее там нет.

В таблице с работниками:

- идентификатор работника – primary key, целочисленный тип;
- его имя, фамилия, логин и пароль – строковый тип, логин должен быть уникальным;
- дата рождения – TIMESTAMP в формате YYYY-MM-DD;
- уровень привилегий – целочисленный тип (1 – администратор, 2 – продавец, 3 – кладовщик, 4 – клиент).

В таблице с историей изменений:

- Идентификатор пользователя – целочисленный тип, foreign key
- Артикул детали – строковый foreign key;
- Время изменения – TIMESTAMP с значением по умолчанию, которое является текущим временем;
- Изменение – положительное или отрицательное число, в зависимости от того, забирали деталь или добавляли.

В таблице с сессиями:

- токен – строковый primary key;
- идентификатор пользователя – целочисленный foreign key.

Такая таблица может быть реализована с помощью ключ-значение СУБД.

Диаграмма сущностей базы данных в нотации Чена представлена на рисунке 2.1

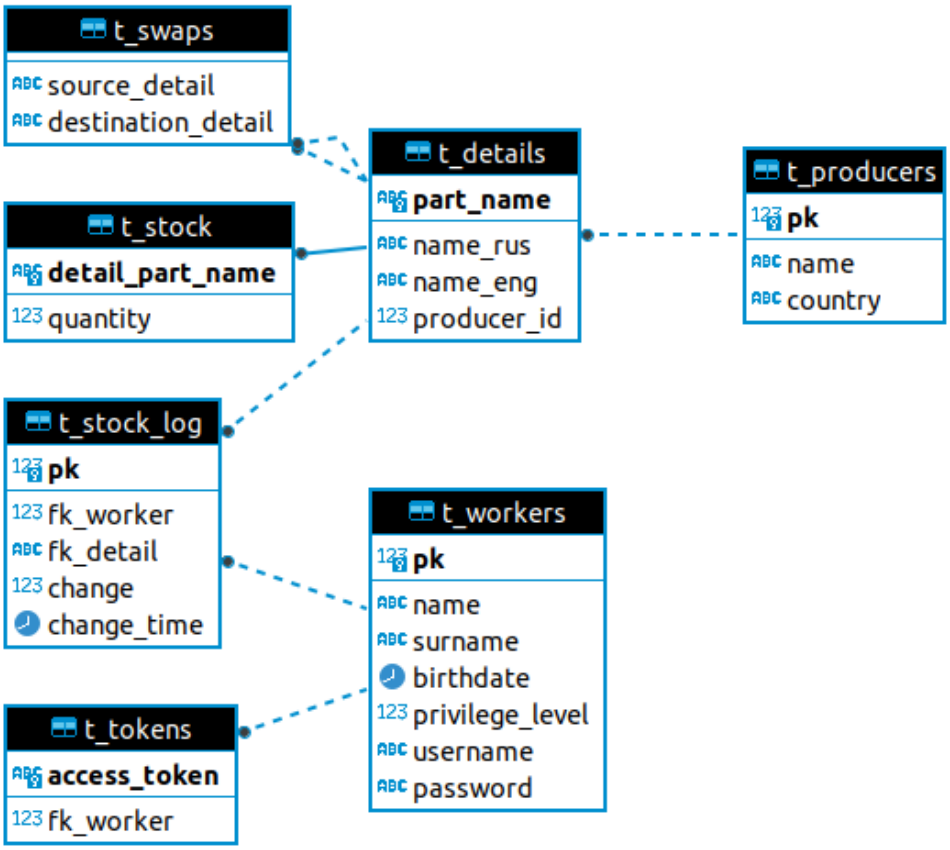

Рисунок 2.1 – Диаграмма сущностей базы данных в нотации Чена

Логины всех пользователей должны быть уникальными, поэтому в БД необходимо создать хранимую функцию добавления нового пользователя, которая будет на вход получать информацию о нем, проверять уникальность логина и, в случае успеха, создавать запись о новом пользователе. Схема алгоритма такой хранимой приведена на рисунке 2.2

Рисунок 2.2 – Схема алгоритма хранимой функции добавления пользователя

2.2 Требования к приложению

Для выполнения поставленной цели приложение должно поддерживать следующий функционал:

- регистрация, авторизация и аутентификация пользователей в системе;
- обновление, добавление, редактирование информации о всех возможных деталях и производителях;
- добавление запчастей на склад и просмотр истории его изменений;
- изменение уровней привилегий пользователей.

Последний пункт может выполнять только работник с правами администратора, добавлять и забирать запчасти со склада – может кладовщик, посмотреть историю и текущее состояние склада – продавец.

2.3 Проектирование приложения

В программе можно выделить три основных компонента: интерфейс, бизнес логика, доступ к данным. Связь между ними представлена на рисунке 2.3

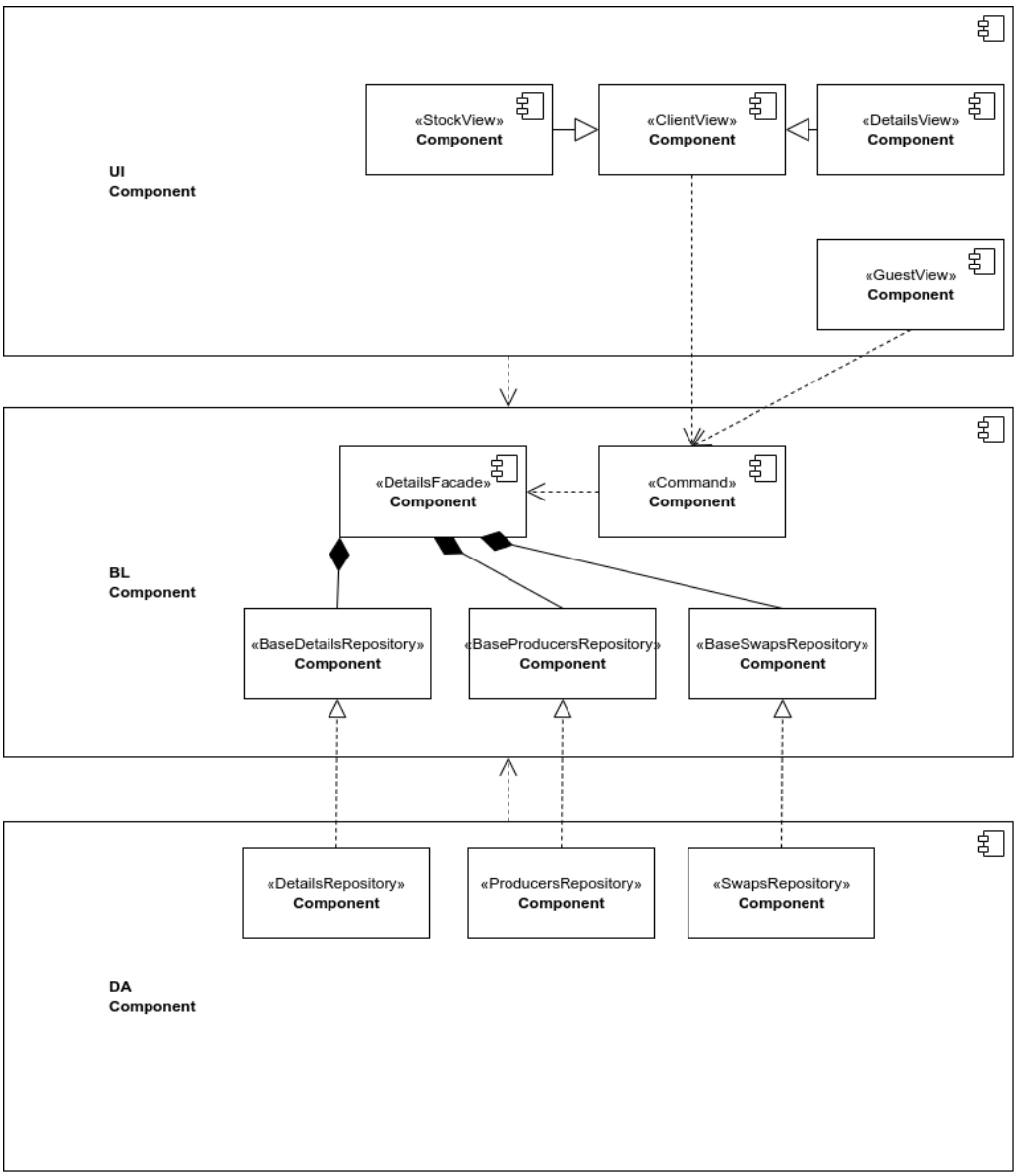

Рисунок 2.3 – Верхнеуровневое разбиение на компоненты

Зависимость компонента доступа к данным от бизнес-логики выполняется по принципу инверсии зависимостей. Для следования этому подходу один модуль должен предоставлять интерфейс, которые реализуется другим, что и представлено на диаграмме компонентов: в бизнес-логике представлены абстрактные классы для доступа к данным базовых сущностей, которые реализованы в компоненте доступа к данным.

UML-диаграмма компонента бизнес-логики представлена на рисунке 2.4

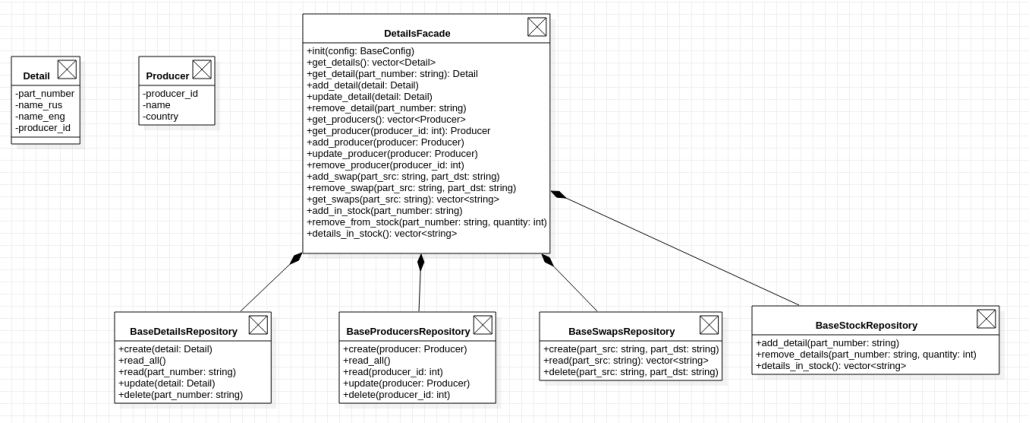

Рисунок 2.4 – UML-диаграмма бизнес-логики

Класс DetalisFacade является единой точкой входа для всех операций бизнес-логики, он содержит реализации интерфейсов для доступа к данным сущностей. Связь интерфейса с бизнес-логикой может быть реализована с помощью REST API [1]. Для этого требуется создать шлюз, который будет принимать запросы от интерфейса, переводить их в понятную для бизнес-логики схему, получать от нее ответ и посылать его обратно интерфейсу.

UML-диаграмма шлюза представлена на рисунке 2.5.

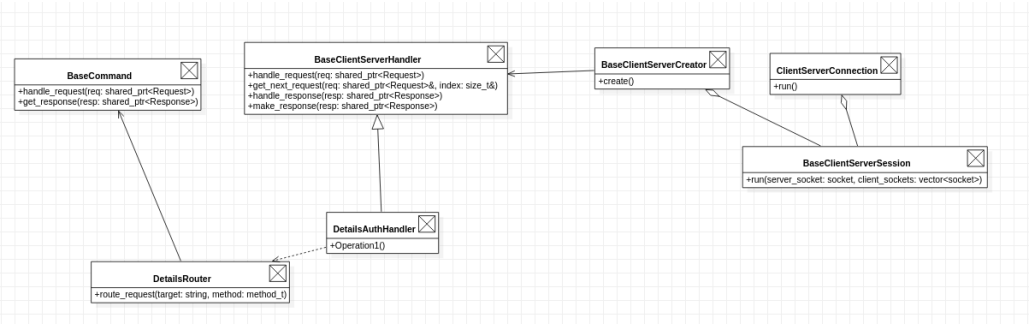

Рисунок 2.5 – UML-диаграмма шлюза

В классе сессии осуществляется прием новых подключений. На каждое из них создается новый объект класса DetailsHandler, который, используя экземпляр DetailsRouter по пути и методу запроса определяет, какую команду нужно использовать. Каждая команда выполняет преобразование данных и запрос к DetailsFacade.

3 Технологический раздел

3.1 Архитектура приложения

Разрабатываемое приложение будет построено по микросервисной архитектуре. Программный продукт можно поделить на три основные части:

- api_gateway – frontend, который принимает все запросы, строит план их исполнения и занимается балансировкой нагрузки на другие микросервисы;
- details – обрабатывает все запросы, которые связаны с запчастями;
- workers – работа с сотрудниками сервиса.

Обобщенная структурная схема архитектуры приложения представлена на рисунке 3.1.

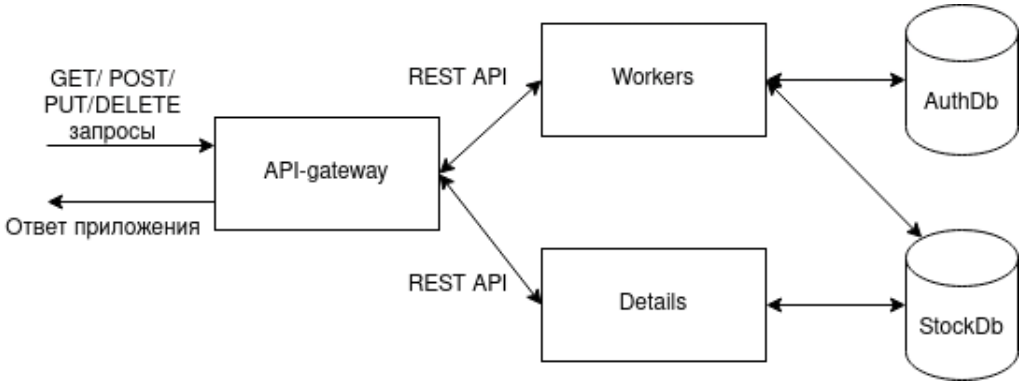

Рисунок 3.1 – Архитектура приложения

3.2 Средства реализации

Для реализации программного продукта был выбран язык программирования C++ [5]. Этот язык является компилируемым, что дает преимущество в скорости в сравнении с интерпретируемыми языками программирования. C++ поддерживает различные механизмы объектно-ориентированного программирования такие как инкапсуляция, наследование и полиморфизм. Эта парадигма сочетается с задачей хранения и обработки объектов реального мира, так как каждый объект можно задать с помощью отдельного класса с соответствующим набором полей и методов [6].

Также для C++ существует фреймворк Boost [7], с помощью которого можно создавать асинхронные подключения, а, начиная с стандарта C++20, это делается с использованием корутин.

Для подключения к СУБД использовались фреймворки pqxx [8] и redis-cpp [9].

Для создания модульных тестов был выбран фреймворк GoogleTests [10]. Данный выбор обусловлен тем, что он включает в себя определения основных классов и макросов, которые используются для создания модульных тестов.

Для тестирования API использовался postman [11]. Он позволяет заранее заготавливать все запросы, а потом в формате одной коллекции проверять результат их исполнения.

3.3 Детали реализации

Взаимодействие приложения с базой данных осуществляется в классах репозиториях для каждой сущности. Далее будет рассмотрен пример на примере деталей, хранящихся в таблицах PostgreSQL.

Подключение к СУБД приведено на листинге 3.1.

Листинг 3.1 – Подключение к СУБД

```
void PostgresDetailsRepository::connect() {
  std::string connection_string = "dbname␣=␣" + name_ + "␣user␣=␣
     " + user_ +
                                  "␣password␣=␣" + user_password_
                                      +
                                  "␣hostaddr␣=␣" + host_ +
                                  "␣port␣=␣" + std::to_string(
                                     port_);

  try {
    connection_ = std::shared_ptr<pqxx::connection>(
        new pqxx::connection(connection_string.c_str()));

    if (!connection_->is_open()) {
      throw DatabaseConnectException("can't␣connect␣to␣" + name_)
         ;
    } else
      std::cout << "Connected␣to␣db␣" << name_ << std::endl;
```

```
  } catch (std::exception& ex) {
    throw DatabaseConnectException("can't␣connect␣to␣" + name_ +
       "␣" +
                                   ex.what());
  }
}
```

Создание заготовленный выражений, которые будут вызываться при чтении, создании, удалении и обновлении объекта представлены на листинге 3.2.

Листинг 3.2 – Заготовленные выражение

```
void PostgresDetailsRepository::add_prepare_statements() {
  connection_->prepare(requests_names[CREATE],
                       "CALL␣PR_ADD_DETAIL($1,␣$2,␣$3,␣$4)");
  connection_->prepare(requests_names[READ_ALL],
                       "SELECT␣*␣FROM␣FN_GET_ALL_DETAILS()");
  connection_->prepare(requests_names[READ_BY_ID],
                       "SELECT␣*␣FROM␣FN_GET_DETAIL_BY_ID($1)");
  connection_->prepare(requests_names[UPDATE],
                       "CALL␣PR_UPDATE_DETAIL($1,␣$2,␣$3,␣$4)");
  connection_->prepare(requests_names[DELETE], "CALL␣
     PR_DELETE_DETAIL($1)");
}
```

Чтение из базы данных всех деталей и перевод их в объекты бизнес логики представлено на листинге 3.3.

Листинг 3.3 – Чтение деталей

```
details_t PostgresDetailsRepository::read_all() {
  pqxx::result res;
  try {
    pqxx::work w(*connection_);
    res = w.exec_prepared(requests_names[READ_ALL]);
    w.commit();
  } catch (...) {
    throw DatabaseExecutionException("can't␣execute␣prepared");
  }

  details_t details;
  for (auto const& row : res) {
    details.push_back(Detail(
```

```
        row["name_rus"].as<std::string>(), row["name_eng"].as<std
           ::string>(),
        row["part_name"].as<std::string>(), row["producer_id"].as
           <size_t>()));
  }

  return details;
}
```

Пример взаимодействия приложения с Redis представлен на листинге 3.4

Листинг 3.4 – Взаимодействие приложения с Redis

```
void RedisAuthRepository::create_session(size_t worker_id,
                                         const std::string& token
                                            ) {
  auto resp = rediscpp::execute(*stream_, "set", token.c_str(),
                                std::to_string(worker_id).c_str()
                                   );

  if (resp.empty())
    throw DatabaseExecutionException("can't␣add␣session␣to␣redis"
       );
}

bool RedisAuthRepository::is_valid_session(size_t& worker_id,
                                           const std::string&
                                              token) {
  auto resp = rediscpp::execute(*stream_, "get", token.c_str());
  if (resp.empty())
    return false;

  worker_id = std::stoull(resp.as<std::string>());
  return true;
}
```

3.4 Взаимодействие с программой

Взаимодействие с программой осуществляется посредством отправки запросов по протоколу HTTP на предоставляемое API. Приложение обрабатывает запросы методов GET, POST, PUT, DELETE на все сущности. Для того,

чтобы подтвердить авторизацию пользователю в каждый из своих запросов необходимо добавить заголовок Authorization со значение токена доступа, полученного при отправки формы с логином и паролем.

Результат работы программы при запросе на получение информации о детали представлен на листинге 3.5.

Листинг 3.5 – Получение информации о детали

```
max@pinguin ~/r/d/d/r/i/lst (main)> curl -v -X GET -H "
   Autharization: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
   eyJleHAiOjE2NTQzMDk0ODV9.
   iNqmjm_POfZfjrA6p6cidESkwFdX31yOBp8oKLr8odw"  --url 'http://
   localhost:8001/details/d1c1991a-89e2-4dd4-a132-cb58ebe7def0'
Note: Unnecessary use of -X or --request, GET is already inferred
   .
*   Trying 127.0.0.1:8001...
* TCP_NODELAY set
* Connected to localhost (127.0.0.1) port 8001 (#0)
> GET /details/d1c1991a-89e2-4dd4-a132-cb58ebe7def0 HTTP/1.1
> Host: localhost:8001
> User-Agent: curl/7.68.0
> Accept: */*
> Autharization: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
   eyJleHAiOjE2NTQzMDk0ODV9.
   iNqmjm_POfZfjrA6p6cidESkwFdX31yOBp8oKLr8odw
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: Boost.Beast/266
< Content-Type: text/html
< Content-Length: 152
<
{"name_eng":"Muffler Fittings","name_rus":"Фитинги глушителя","
   part_number":"d1c1991a-89e2-4dd4-a132-cb58ebe7def0","
   producer_id":"484"}
* Connection #0 to host localhost left intact
```

Результат работы программы при запросе на авторизацию в системе представлен на листинге 3.6.

Листинг 3.6 – Авторизация в системе

```
max@pinguin ~/r/d/d/r/i/lst (main)> curl -v -X POST -d'{
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"username":␣"maxermu",
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣"password":␣"123456"
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣}'  --url 'http://localhost
   :8001/auth/login'
< HTTP/1.1 200 OK
{"access_token":"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
   eyJleHAiOjE2NTQzMDk1OTN9.D512fw_tiALGmZ6D32-6
   zPncnuEkDgmMSSHfXPa742k"}
* Connection #0 to host localhost left intact
```

4 Исследовательский раздел

4.1 Постановка эксперимента

В данном исследовании будет проведено сравнение временных характеристик приложения при хранении токенов авторизованных пользователей в PostgreSQL и Redis. До постановки эксперимента ожидается, что время поиска в Redis будет меньше, так как эта СУБД хранит данные в оперативной памяти, которая быстрее чем вторичная, в которой ищет строки PostgreSQL [4]. Для чистоты эксперимента надо создать индекс по ключу поиска в PostgreSQL.

4.2 Технические характеристики

Технические характеристики устройства, на котором выполнялось тестирование и исследование, приведены ниже.

- Операционная система: Ubuntu Linux 64-bit.
- Оперативная память: 16 GB.
- Количество логических ядер - 8.
- Процессор: Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz [**intel**].

Тестирование проводилось на компьютере, включенном в сеть электропитания. Во время тестирования компьютер был нагружен только встроенными приложениями окружения рабочего стола, окружением рабочего стола, а также непосредственно системой тестирования. Во время тестирования оптимизации компилятора были отключены.

4.3 Замеры времени

Время замерялось с помощью функции now [12].

Результаты замеров времени (в мс) поиска несуществующей строки в различных субд приведены в таблице 4.1. На рисунке 4.1 приведена зависимость времени поиска строки, которой нет базе данных, от количества записей.

Таблица 4.1 – Замер времени поиска строки в базе данных

строк в БД	**Время в мс на n потоков**	
	Redis	**PostgreSQL**
25000	0.0495	0.2145
50000	0.0505	0.2185
75000	0.0517	0.22
100000	0.069	0.2225
125000	0.0945	0.232
150000	0.08	0.222

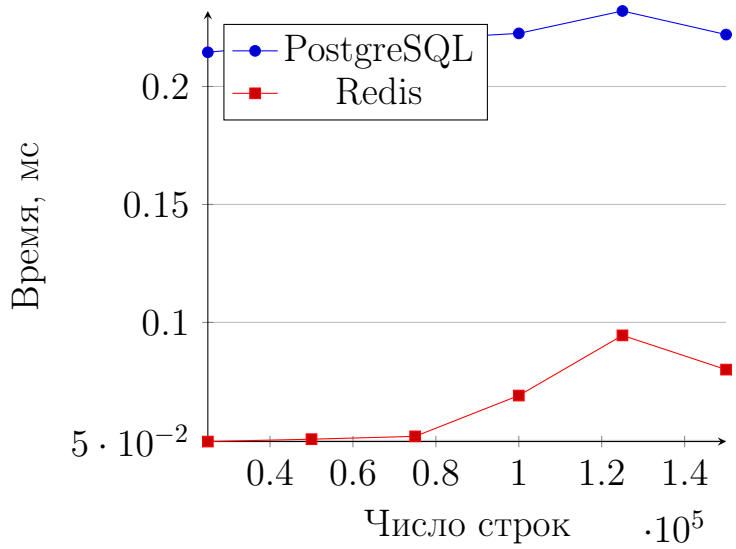

Рисунок 4.1 – Зависимость времени поиска токена от выбранной СУБД.

Вывод

Из графиков видно, что поиск токена в Redis работает на порядок быстрее, чем в PostgreSQL. Это связано с тем, что Redis хранит все данные в оперативной памяти, которая быстрее вторичного хранилища, которое использует PostgreSQL.

В приложении токены используется только для подтверждения пользователем того, что его сессия авторизована, поэтому потеря данных в такой таблице не является критичным случаем, так как для ее восстановления достаточно будет еще раз заполнить форму авторизации.

Также Redis позволяет устанавливать срок существования пары ключ-значение, по истечении которого она будет удалена. Все токены авторизации,

которые выдаются пользователям, также имею срок валидности, что еще раз указывает на преимущество использование Redis, а не PostgreSQL.

ЗАКЛЮЧЕНИЕ

В рамках курсового проекта были решены следующие задачи:

- формализовано задание, определен необходимый функционал;
- проведен анализ СУБД;
- спроектирована база данных, описаны ее сущности и связи;
- спроектировано приложение для доступа к БД;
- реализовано программное обеспечение, которое позволяет получать доступ к данным по средствам REST API [1].
- проведен эксперимент на сравнение времени поиска в двух различных СУБД.

Была создана программа, позволяющая менеджерам и администраторам автосервиса управлять состоянием склада. Приложение было протестировано с помощью модульных тестов и проверки всей функциональности через запросы к API.

В результате исследование было выяснено, что поиск в СУБД Redis на порядок быстрее, чем в PostgreSQL.

Поставленная цель достигнута.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 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: 07.05.2022).
2. *Kleppmann M.* Designing Data-Intensive Applications //. — Edition. Sebastopol: O'Reilly Media, 2017. — с. 640.
3. OAuth Core 1.0 [Электронный ресурс]. — Режим доступа: `https://oauth.net/core/1.0/` (дата обращения: 07.05.2022).
4. 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: 07.05.2022).
5. Документация по Microsoft C++, C и ассемблеру [Электронный ресурс]. — Режим доступа: `https://docs.microsoft.com/ru-ru/cpp/` (дата обращения: 07.05.2022).
6. *Гради Буч Роберт А. Максимчук М. У. Э.* Объектно-ориентированный анализ и проектирование с примерами приложений //. — Издательский дом Вильямс, 2008.
7. Документация по фреймворку Boost [Электронный ресурс]. — Режим доступа: `https://www.boost.org/` (дата обращения: 07.05.2022).
8. Документация по фреймворку pqxx [Электронный ресурс]. — Режим доступа: `http://www.pqxx.org/` (дата обращения: 07.05.2022).
9. Документация по фреймворку redis-cpp [Электронный ресурс]. — Режим доступа: `https://github.com/tdv/redis-cpp` (дата обращения: 07.05.2022).
10. Документация по фреймворку GoogleTests [Электронный ресурс]. — Режим доступа: `https://google.github.io/googletest/` (дата обращения: 07.05.2022).
11. Документация по postman [Электронный ресурс]. — Режим доступа: `https://www.postman.com/` (дата обращения: 07.05.2022).

12. Документация по chrono [Электронный ресурс]. — Режим доступа: `https://www.cplusplus.com/reference/chrono/system_clock/now/` (дата обращения: 07.05.2022).

ПРИЛОЖЕНИЕ А
Презентация