**1.** **ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

Изм.

Лист

№ докум.

Подпись

Дата

Лист

3

П.1811618.УП.17.00.ПЗ

* 1. **Постановка задачи**

Создать программный комплекс (ПК), позволяющий автоматизировать контроль, сбор и анализ информации о бронировании номеров в гостинице. Сбор и контроль обработанной информации выполняется одновременно с нескольких клиентских мест. Информация хранится в структурированном виде, позволяющем выполнять заготовленные выборки. Программный продукт должен хорошо горизонтально масштабироваться.

Программный продукт создается в соответствии со следующей спецификацией:

*Спецификация*

Название задачи: «Гостиница».

Название программы – GostClient и GostServer.

Система программирования –Python и SQLite.

Компьютер:

* Процессор– Intel(R) inside Pentium(R);
* Оперативная память –2 Гб, и более;
* 10 Гб на жёстком диске и более.

*Описание:*

Имеются три уровня ПО: клиент, сервер и СУБД.

*Цель:*

Научиться создавать трёхуровневые приложения.

*Логическая модель:*

1. Клиент подает запрос на сервер;
2. Сервер принимает запрос и передаёт команду на СУБД;
3. СУБД принимает команду и передаёт данные серверу;
4. Сервер принимает и посылает полученные данные клиенту.

*Входные данные.*

Вводимые данные при изменении выбранного поля таблицы.

*Требования к входным данным.*

Все вводимые данные должны быть определенных типов, которым соответствуют поля.

*Выходные данные.*

В результате работы программы должна выводиться таблица с данными из БД.

*Требования к выходным данным.*

Должно быть красиво и понятно.

*Ошибки.*

1. При вводе данных, надо предусмотреть их соответствие типу поля;
2. При диагностировании ошибки программа должна выводить сообщение с просьбой повторного ввода.

*Инструкция пользователя*

1. Установить Python 3.7;
2. Открыть cmd;
3. Прописать команды pip install flask, pip install requests.
   1. **Описание концепции трехуровневой архитектуры**

Согласно требованиям технического задания, создаваемый ПП должен состоять из нескольких модулей:

* клиент;
* сервер;
* база данных.

### **Характеристика инструментальных средств**

*Python*

Python – это комбинация нескольких важнейших технологий

* высокопроизводительный компилятор в машинный код;
* объектно-ориентированная модель компонент;
* визуальное (а, следовательно, и скоростное) построение приложений из программных прототипов;
* масштабируемые средства для построения баз данных.

Python – [высокоуровневый язык программирования](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) общего назначения, ориентированный на повышение производительности разработчика и читаемости кода.

[Синтаксис](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D0%BD%D1%82%D0%B0%D0%BA%D1%81%D0%B8%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) ядра Python минималистичен. В то же время [стандартная библиотека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B0%D0%BD%D0%B4%D0%B0%D1%80%D1%82%D0%BD%D0%B0%D1%8F_%D0%B1%D0%B8%D0%B1%D0%BB%D0%B8%D0%BE%D1%82%D0%B5%D0%BA%D0%B0_Python) включает большой набор полезных функций.

Python поддерживает структурное, обобщенное, объектно-ориентированное программирование.

Основные архитектурные черты - динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычислений, высокоуровневые структуры данных. Поддерживается разбиение программ на модули, которые, в свою очередь, могут объединятся в пакеты.

Python – активно развивающийся язык программирования, новые версии с добавлением/изменением языковых свойств выходят примерно раз в два с половиной года.

### *SQLite*

Движок SQLite не является отдельно работающим процессом, с которым взаимодействует программа, а представляет собой библиотеку, с которой программа компонуется, и движок становится составной частью программы. Таким образом, в качестве протокола обмена используются вызовы функций (API) библиотеки SQLite. Такой подход уменьшает накладные расходы, время отклика и упрощает программу. SQLite хранит всю базу данных (включая определения, таблицы, индексы и данные) в единственном стандартном файле на том компьютере, на котором исполняется программа. Простота реализации достигается за счёт того, что перед началом исполнения транзакции записи весь файл, хранящий базу данных, блокируется; ACID-функции достигаются в том числе за счёт создания файла журнала.

Несколько процессов или потоков могут одновременно без каких-либо проблем читать данные из одной базы. Запись в базу можно осуществить только в том случае, если никаких других запросов в данный момент не обслуживается; в противном случае попытка записи оканчивается неудачей, и в программу возвращается код ошибки. Другим вариантом развития событий является автоматическое повторение попыток записи в течение заданного интервала времени.

SQLite возможно использовать как навстраиваемых системах, так и на выделенных машинах с гигабайтными массивами данных.

# **2. ПРАКТИЧЕСКАЯ ЧАСТЬ**

## **2.1 Концептуальная модель программного комплекса**

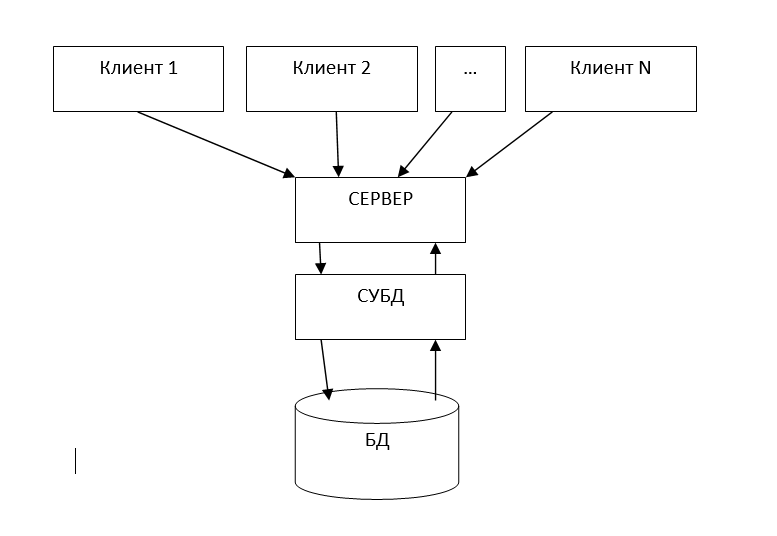


Рисунок 1. Концептуальная модель

**2.2 UML-диаграмма**

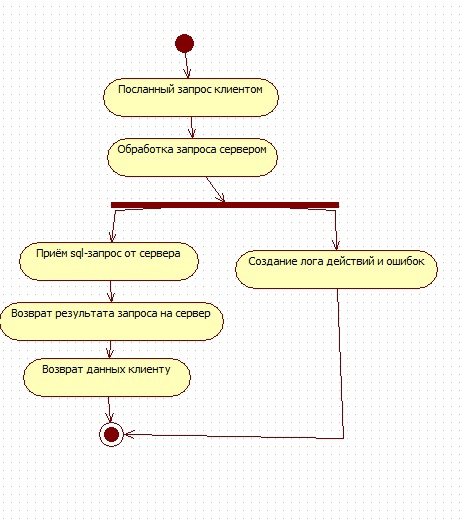


Рисунок 2. Диаграмма активностей

## **2.3 ER- диаграмма (Логическая схема)**

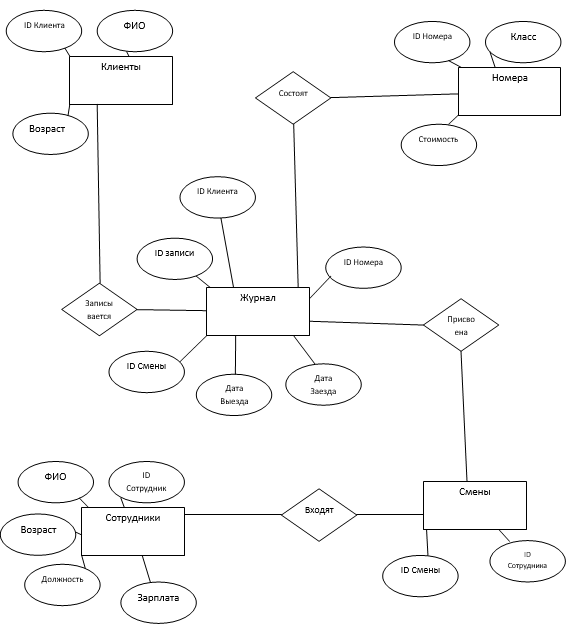


Рисунок 3. ER- диаграмма

**2.4 SQL запросы к базе данных**

Запросы выводов таблиц:

* SELECT \* FROM Clients;
* SELECT \* FROM Sotrudniki;
* SELECT \* FROM Smena;
* SELECT \* FROM Nomera;
* SELECT \* FROM Zhurnal.

Запросы таблицы «Клиенты»:

* SELECT \* FROM Clients Where IdCl {cmbx2} {zapr};
* SELECT \* FROM Clients Where FIO LIKE '{zapr}%';
* SELECT \* FROM Clients Where Old {cmbx2} {zapr}.

Запросы таблицы «Сотрудники»:

* SELECT \* FROM Sotrudniki Where IdSotr {cmbx2} {zapr};
* SELECT \* FROM Sotrudniki Where FIO LIKE '{zapr}%';
* SELECT \* FROM Sotrudniki Where Old {cmbx2} {zapr};
* SELECT \* FROM Sotrudniki Where Zarplata {cmbx2} {zapr}.

Запросы таблицы «Смена»:

* SELECT \* FROM Smena Where IdSmen {cmbx2} {zapr};
* SELECT \* FROM Smena Where IdSot {cmbx2} {zapr}.

Запросы таблицы «Номера»:

* SELECT \* FROM Nomera Where IdNom {cmbx2} {zapr};
* SELECT \* FROM Nomera Where Price {cmbx2} {zapr};
* SELECT \* FROM Nomera Where Class LIKE '{zapr}%'.

Запросы таблицы «Журнал»:

* SELECT \* FROM Zhurnal Where IdZap {cmbx2} {zapr};
* SELECT \* FROM Zhurnal Where IdCl {cmbx2} {zapr};
* SELECT \* FROM Zhurnal Where IdNom {cmbx2} {zapr};
* SELECT \* FROM Zhurnal Where IdSmen {cmbx2} {zapr};
* SELECT \* FROM Zhurnal Where DateZaezd {cmbx2} '{zapr}';
* SELECT \* FROM Zhurnal Where DateViezd {cmbx2} '{zapr}'.

**2.5 Серверный модуль**

Сервер приложений – это программная платформа ([фреймворк](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B5%D0%B9%D0%BC%D0%B2%D0%BE%D1%80%D0%BA)), предназначенная для эффективного исполнения процедур (программ, скриптов), на которых построены приложения. Сервер приложений действует как набор компонентов, доступных разработчику программного обеспечения через API ([интерфейс прикладного программирования](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_%D0%BF%D1%80%D0%B8%D0%BA%D0%BB%D0%B0%D0%B4%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), определённый самой платформой.

Серверы приложений проектируются таким образом, чтобы добавление к ним дополнительных экземпляров обеспечивало горизонтальное масштабирование производительности программного комплекса и не требовало внесения изменений в программный код приложения.

Сервер баз данных (слой данных) обеспечивает хранение данных и выносится на отдельный уровень, реализуется, как правило, средствами систем управления базами данных, подключение к этому компоненту обеспечивается только с уровня сервера приложений.

С помощью СУБД происходит обработка запросов к базе данных, поступаемых от клиентского модуля.

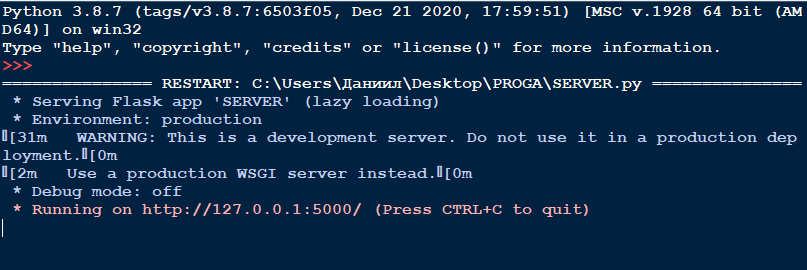


Рисунок 4. Серверный модуль

**2.6 Клиентский модуль**

Клиент (слой клиента) – это интерфейсный (обычно графический) компонент комплекса, предоставляемый конечному пользователю. Этот уровень не должен иметь прямых связей с базой данных, быть нагруженным основной бизнес логикой (по требованиям масштабируемости) и хранить состояние приложения (по требованиям надёжности). На этот уровень обычно выносится только простейшая бизнес-логика: интерфейс авторизации, алгоритмы шифрования, проверка вводимых значений на допустимость и соответствие формату, несложные операции с данными (сортировка, группировка, подсчёт значений), уже загруженными на терминал.

В клиенте располагается таблица для вывода результатов запросов и управляющие кнопки.

В нижней части расположены элементы для редактирования базы данных для каждой таблицы.

Клиент лишь посылает запрос серверу, получает результат запроса и выводит результат запроса в таблицу.

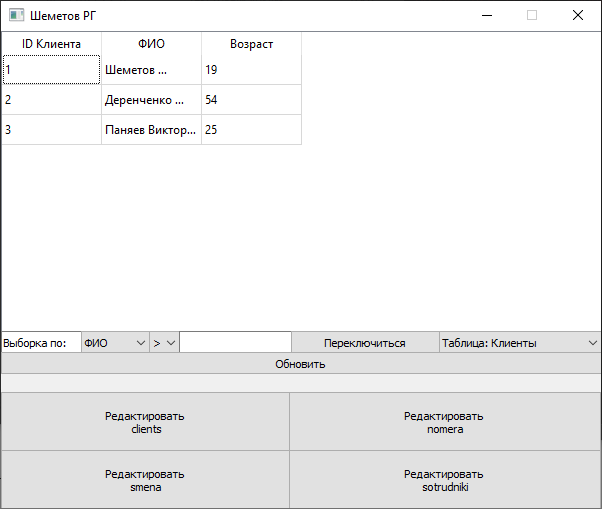


Рисунок 5. Клиентский модуль

**ЗАКЛЮЧЕНИЕ**

Во время учебной практики был создан программный комплекс «Гостиница». Сервер и клиент взаимодействуют с помощью компонента Flask. При разработке была использована среда для разработки приложений Python.

За время практики было изучено, как создавать приложения, взаимодействие которых происходит посредством связи клиент-сервер.

Первым, что необходимо было сделать – это составить спецификацию, разработать алгоритм решения задачи и создание концептуальной модели. В составе спецификации описывается название задачи, системные требования к программному продукту, описание программы, ее логическая модель, управлениями режимами работы, входные и выходные данные, пример работы ПК. Для разработки алгоритма решения задачи решались такие вопросы, как какой модуль будет создаваться в первую очередь, какие основные элементы будут на клиенте и сервере, придумать примерный интерфейс программы. Создание концептуальных моделей требовало изучение литературы и статей, и только после этого происходило само создание, далее происходило создание таких диаграмм, как UML.

Реализация базы данных. В реализации использовалась SQLite. Были созданы пять таблиц, соединенных между собой.

Самым главным заданием во всей разработке программы является написание серверного алгоритма. Серверный модуль представляет собой скрипт который принимает JSON запросы, работает с базой данных и возвращает клиенту данные.

Реализация клиентского приложения. Именно в нем необходимо присоединять базу данных и разработать выполнение запросов, и самое главное, передавать серверу о действиях с базой. Для подключения базы использовались такой компонент, как sqlite3. Через цепочку подключения базы завершающим элементом было подключение базы к таблице, чтобы можно было видеть содержимое и по необходимости изменять ее.

Учебная практика показалась достаточно интересной и полезной. В конечном итоге получилась полезная программа, которая, возможно, может быть доработана и сослужить службу в более серьезных решениях.