**Міністерство освіти і науки України**

**Одеський Національний Університет імЕНІ І.І. Мечникова**

**факультет математики, фізики та інформаційних технологій**

**КАФЕДРА МАТЕМАТИЧНОГО ЗАБЕЗПЕЧЕННЯ КОМП'ЮТЕРНИХ СИСТЕМ**

**ПОЯСНЮВАЛЬНА ЗАПИСКА**

**до курсового проекту**

**з дисципліни «Організація баз даних»**

**на тему**

**«Інформаційна центру**»

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | студента ІІI курсу  групи 3  спеціальності Комп’ютерна інженерія  Грабовецького Фелікса Олександровича  Керівник:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Захищено «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_202\_\_р.  з оцінкою \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Комісія:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (ПIБ) (Підпис)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (ПIБ) (Підпис)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (ПIБ) (Підпис) |

м.Одесса

АННОТАЦИЯ

В данной курсовой работе разрабатывается информационная система «Сотрудники фитнес-клуба».

Цель работы – разработка информационной системы для «Фитнес клуба». Отличительной чертой данной системы, является полная настраиваемость под нужды сотрудников, начиная от количества пользователей и их прав в системе и заканчивая системой отслеживания работы тренерского состава и системы настройки планирования индивидуальных и групповых занятий . Например: составление расписание для каждого тренера , указание различных типов услуг, которые может приобрести клиент фитнес-клуба. Также проведение анализа различных задач решаемых в предметной области «Фитнес клуб», например: найм на работу сотрудника фитнес-клуба, просмотр расписания, удобно отслеживаемая система учета работы тренерского состава.

Результатом курсовой работы является информационная система с удобным и легко воспринимаемым пользовательским интерфейсом.

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ 4

ВВЕДЕНИЕ 5

1 ПОСТАНОВКА ЗАДАЧИ 6

2 ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ 7

3 ИНФОРМАЦИОННОЕ МОДЕЛИРОВАНИЕ 10

4 ВЫБОР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 15

5 СОЗДАНИЕ БАЗЫ ДАННЫХ 17

6 РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БАЗЕ ДАННЫХ 18

7 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ 19

8 ЗАПРОСЫ К БАЗЕ ДАННЫХ 20

9 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 22

ЗАКЛЮЧЕНИЕ..............................................................................................26

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27

ПРИЛОЖЕНИЕ А. СПИСОК ЗАДАЧ ПОЛЬЗОВАТЕЛЕЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ 28

ПРИЛОЖЕНИЕ Б. ЗАПРОСЫ НА СОЗДАНИЕ БАЗЫ ДАННЫХ 31

ПРИЛОЖЕНИЕ В. ЗАПРОСЫ НА СОЗДАНИЕ ХРАНИМЫХ ПРОЦЕДУР 33

ПРИЛОЖЕНИЕ Г. ЗАПРОСЫ НА СОЗДАНИЕ ПОЛЬЗОВАТЕЛЕЙ БД 38

ПРИЛОЖЕНИЕ Д. ИСХОДНЫЙ КОД БАЗОВЫХ КЛАССОВ 52

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ, УСЛОВНЫХ ОБОЗНАЧЕНИЙ И ТЕРМИНОВ

BLL – Business Logic Layer

DAL – Data Access Layer

MVP – Model-View-Presenter

MVT – Model-View-Template;

PL – Presentation Layer

UI – User Interface

ПрО – предметная область

# ВВЕДЕНИЕ

Фитнес клуб – система, в которой происходит проведение услуг, предоставленных самим клубом. Это довольно популярный тип организаций, в связи с быстрым ритмом жизни, все больше людей нуждаются в нем. Для сотрудников данных организаций необходимо отслеживать посещаемость сотрудников, контролировать расписание тренировок и типов занятий.

Наличие инструмента, который позволяет минимизировать затраты времени, материальных, трудовых ресурсов в ходе своей деятельности и упростить процесс обработки информации, значительно сократит рабочее время сотрудников и увеличит процент структурирорванности организации использующей данный продукт. Что в свою очередь повысит коэффициент полезного действия сотрудников которым приходилось отслеживать все операции производимые в данной программе вручную, или же с помощью другого неспециализрованного програмного обеспечения.

Целью данного курсового проекта является создание информационной системы, которая будет максимально гибкой и настраиваемой. Для достижения указанной цели в курсовой работе необходимо решить следующие задачи:

1. анализ предметной области;
2. формирование требований к фитнес центру;
3. проектирование функциональной модели программы фитнес центра;
4. разработка архитектуры программы фитнес центра;
5. проектирование БД программы фитнес центра;
6. выбор технологий и средств реализации программы фитнес центра;
7. обеспечение разграничения доступа к приложению со стороны различных категорий пользователей и защиты от несанкционированного доступа.

# 1 ПОСТАНОВКА ЗАДАЧИ

Когда клиент хочет записаться в фитнес центр, ему нужно прийти в него, оставить свои персональные данные, в случае, если он ещё не был клиентом данной организации, и выбрать тот абонемент(услугу), которую он желает приобрести. Регистрацией клиента занимается администратор. Для того, чтобы клиент числился в организации его персональные данные, ту услугу которую он выбрал должны занести в базу данных. Далее, на этой основе можно проводить анализ популярности абонементов. В фитнес клубе должен быть директор, который нанимает или увольняет персонал. Исходя из бизнес-процессов, имеющих место в данной ПрО, было выделено четыре категории пользователей, имеющих доступ к информационной системе:

1. Директор – имеет возможность нанять и уволить сотрудников, изменить список услуг;
2. Администратор – имеет возможность зарегистрировать клиента в базу данных;
3. Тренер – имеет возможность на просмотр расписания на прошлую, текущую и следующую неделю;
4. Гость – необходим для верификации пользователя пытающегося подключиться к бд и выдачи ему роли.

Таким образом, каждый пользователь данной информационной системы имеет свою роль, а основываясь на этой роли, функциональность, которую ему предоставляет система. Информационная система также должна обеспечить поддержку решения следующих аналитических задач:

1. просмотр прибыли фитнес клуба в указываемом месяце;
2. просмотр расписания тренировок;

Полный список задач, записанных в формате пользовательских историй ,можно просмотреть в Приложении А.

# ПРОЕКТИРОВАНИЕ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Для реализации информационной системы выбрана трехзвенная архитектура клиент-сервер. В этой архитектуре между клиентом и сервером баз данных имеется промежуточный слой - сервер приложений, являющийся для пользователя сервером, а для системы управления базами данных – клиентом. На сервере приложений хранится программное обеспечение, содержащее основную бизнес-логику информационной системы (основные алгоритмы обработки данных). Сервер приложений анализирует требования пользователя и формирует запросы к серверу БД. Сервер БД помимо хранения данных, осуществляет поддержание их целостности, выполняет запросы к данным, обеспечивает безопасность и разграничение доступа к данным.

Такая архитектура позволяет повысить вычислительную мощность системы путем распределения нагрузки между несколькими компьютерами, а также снижает требования к ресурсам рабочих мест пользователей за счет использования «тонкого» клиента, в функции которого входит только предоставление пользовательского интерфейса. Кроме того, при изменении бизнес-логики нет необходимости изменять клиентские приложения и обновлять их у всех пользователей.

Информационная система была спроектирована на основе паттерна MVT, поскольку Flask сам заботится о части контроллера(программный код, который контролирует взаимодействие между моделью и представлением) . Шаблон представляет собой файл HTML, смешанный с языком шаблонов Jinja2(рис. 2.1).

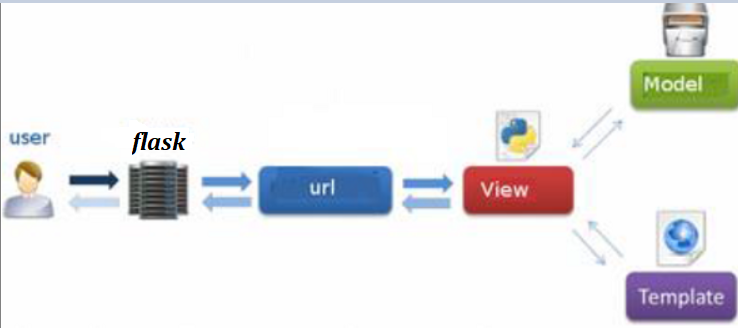


Рисунок 2.1 – Диаграмма работы паттерна MVT

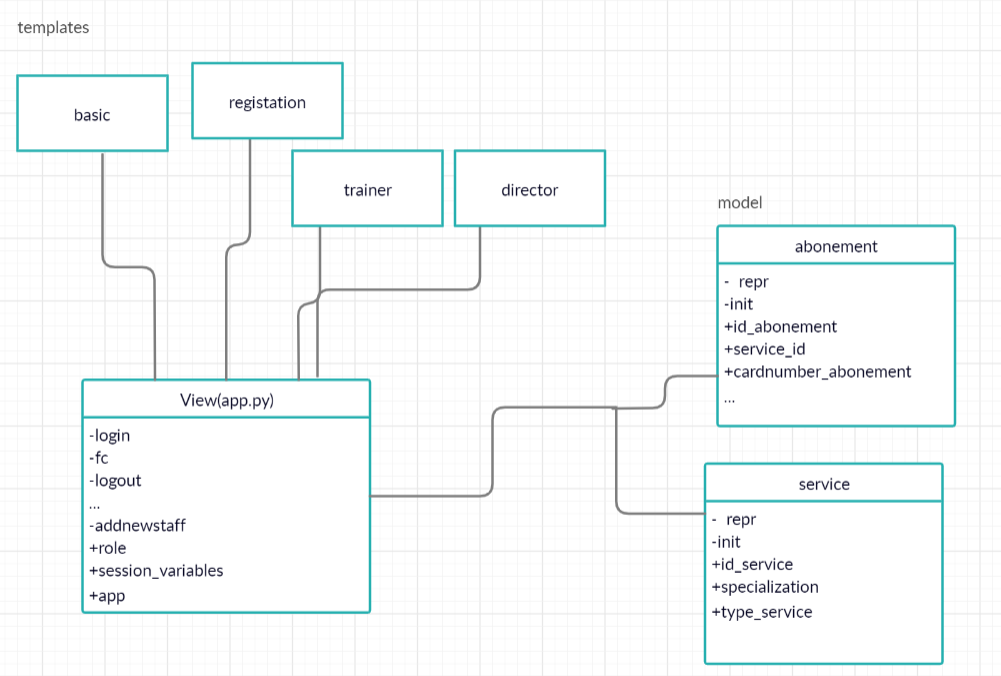


Рисунок 2.2 - Диаграмма классов проекта курсовой работы

MVС и MVT- очень похожи друг на друга, но их применение зависит от условий использования. Причиной выбора паттерна проектирования MVT является рядом особенностей процесса разработка на микропроцессоре Flask и относительная простота реализации в отличии от других трехуровневых шаблонов;

# ИНФОРМАЦИОННОЕ МОДЕЛИРОВАНИЕ

Первый этап проектирования заключается в создании схемы БД, которая включает определение сущностей и существующих между ними связей и атрибутов. Результатом данного проектирования является схема БД, представленная в виде ER-диаграммы (Entity-Relationship), на которой отображаются основные сущности ПрО, их атрибуты и связи. Схема БД создаётся на основе функциональных требований пользователей и абсолютно не зависит от каких-либо особенностей физической реализации информационной системы, таких как тип выбранной СУБД или язык программирования.

Последовательность этапов создания схемы базы данных:

1. определение сущностей;
2. определение связей между сущностями;
3. определение атрибутов сущностей;
4. задание первичных и альтернативных ключей.

Исходя из анализа ПрО, для функционирования информационной системы необходимы следующие сущности:

1. абонемент: дата начала абонемента, дата конца абонемента, стоимость абонемента;
2. клиент: имя клиента, фамилия клиента, отчество клиента, мобильный телефон клиента, пол клиента;
3. услуга: название услуги, тип услуги;
4. сотрудник: имя сотрудника, фамилия сотрудника, отчество сотрудника, мобильный телефон сотрудника, должность сотрудника, зарплата сотрудника;
5. отпускные: начало отпуска, конец отпуска;
6. рабочая смена: день рабочей смены, время начала рабочей смены, конец рабочей смены;
7. запись к клиенту: день записи к тренеру, время начала тренировки, время конца тренировки

Далее для каждой сущности нужно задать первичный ключ – уникальный идентификатор, который однозначно характеризует каждый экземпляр, а также уникальные ключи.

Таблица 3.1 - Описание сущностей предметной области «Фитнес клуб».

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Назначение атрибута | Ограничения целостности |
| Абонемент | | |
| Id\_abonement | Идентификатор абонемента | первичный ключ |
| Service\_id | Идентификатор услуги | Внешний ключ для связи с сущностью service(service\_id) |
| Cardnumber\_abonement | Номер карты сотрудника, который приобрел некоторый абонемент | Внешний ключ для связи с сущностью client(cardnumber\_abonement) |
| Begindate\_abonement | День начала абонемента | Не пустое |
| Enddate\_abonement | День окончания абонемента | не пустое |
| Isactive\_abonement | Активен ли абонемент | не пустое |
| Cost\_abonement | Стоимость абонемента | не пустое |
| Клиент | | |
| Card\_number\_client | Идентификатор номер карты | первичный ключ |
| Name\_client | Имя клиента | Не пустое |
| Surname\_client | Фамилия клиента | Не пустое |
| Patronomyc\_client | Отчество клиента | Не пустое |
| Mobile\_telephone\_client | Мобильный телефон клиента | не пустое |
| Review | Отзыв |  |
| Sex | Пол клиента | Не пустое (возможны два значения мужской (м) и женский (ж) |
|  | Услуга |  |
| Id\_service | Идентификатор услуги | Внешний ключ |
| Specialization\_service | Название услуги | Не пустое |
| Type\_service | Вид услуги | Не пустое |

|  |  |  |
| --- | --- | --- |
| Сотрудник | | |
| Id\_staff | Идентификатор сотрудника | Первичный ключ |
| Service\_id | Идентификатор услуги | Внешний ключ для связи с сущностью service(service\_id) |
| Name\_staff | Имя сотрудника | Не пустое |
| Surname\_staff | Фамилия сотрудника | Не пустое |
| Patronomyc\_staff | Отчество сотрудника | Не пустое |
| Mobile\_telephone\_staff | Мобильный телефон сотрудника | Не пустое |
| Position\_staff | Должность сотрудника | Не пустое(возможны варианты: Тренер, Администратор, Массажист, Директор) |
| Salary | Зарплата сотрудника | Не пустое |
| Login | Логин сотрудника | Не пустое |
| Passw | Пароль сотрудника | Не пустое |
| Role\_login | Роль логина | Не пустое(возможны варианты 1, 2, 3) |
| Отпускные | | |
| Id\_vacation | Идентификатор отпуска | Первичный ключ |
| Vacation\_staff | Отпускные сотрудника | Внешний ключ для связи с сущностью staff(Vacation\_staff) |
| Begin\_date\_vacation | Начало отпуска | Не пустое |
| End\_date\_vacation | Окончание отпуска | Не пустое |
| Paid | Оплачиваемый ли отпуск | Не пустое(возможны варианты true или false) |
|  | Рабочая смена |  |
| Id\_workshift | Идентификатор рабочей смены | Первичный ключ |
| Workshift\_staff | Рабочая смена сотрудника | Внешний ключ для связи с сущностью staff(Workshift\_staff) |
| Date\_workshift | День недели рабочей смены | Не пустое(возможны варианты: дни недели) |
| Begin\_time\_workshift | Начало рабочей смены | Не пустое |
| End\_time\_workshift | Конец рабочей смены | Не пустое |
|  | Запись к тренеру |  |
| Cardnumber\_client\_record | Номер карточки клиента, у которого происходит запись | Не пустое |
| End\_time\_workshift | Конец рабочей смены | Не пустое |

Следующим шагом является формализация связей между сущностями.

Связь многие-ко-многим формализуется с помощью трех таблиц. Две и более таблицы - "источника" и одна соединительная таблица. Первичный ключ соединительной таблицы может быть простым первичным ключом, а может быть составным и состоит из двух полей, двух внешних ключей, которые ссылаются на первичные ключи таблиц A и B.

Для формализации связи друг к другу при наследовании одной сущности несколькими:

1) Используется модификатор атрибута UNIQUE, что обеспечивает уникальность атрибута во всех записях таблицы;

2) Триггер что обеспечивает целостность данных, примером данной проблемы является то, что занятие, параметры которого (начало и конец времени, комната) не подходят ни для 1го занятие в расписании.

В данной ПрО можно выделить следующие связи:

* 1. Связь тренер - клиент, формализуется по правилу «многие ко многим»;
  2. Связи сотрудник -отпускные формализуются по правилу «один ко многим»;
  3. Связь сотрудник – рабочая смена так же является связью «один к многим»
  4. Связь клиент-абонемент также является связью «один к многим»

1. Роли: id – первичный ключ, номер роли. Эта таблица служит справочником для всех ролей в фитнес клубе;
2. Должности: id(position\_name) – первичный ключ, список должностей. Эта таблица служит справочником для всех типов ролей в фитнес клубе;
3. пользователи: id – первичный ключ, логин, пароль, роль, фио. При входе пользователя в систему необходимо установить, какую роль он имеет в БД;

После формализации была получена следующая схема, представленная на рисунке 3.1.

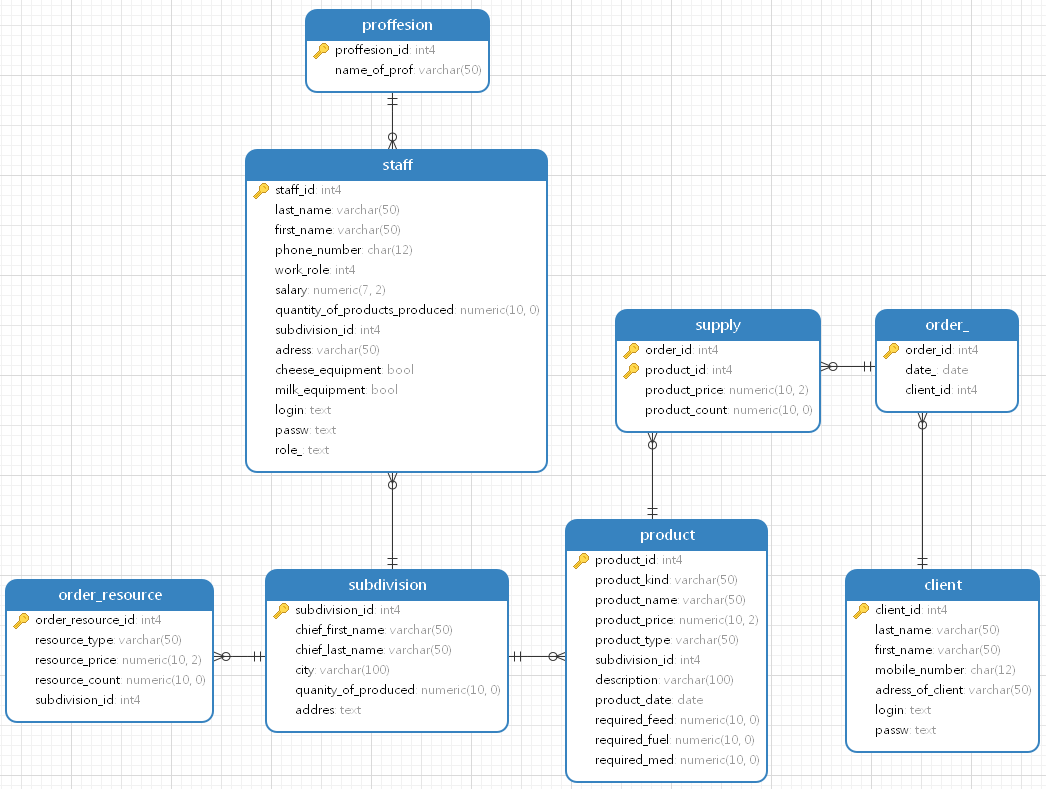


Рисунок 3.1 – ER-диаграмма базы данных фитнес клуба

# ВЫБОР ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

В качестве языка программирования выбран python, и используется микрофреймворк Flask, который использует механизм шаблонов Jinja2, которые обладают рядом преимуществ, а именно:

1. Преимущества Flask в том, что он позволяет создать веб-сайты довольно просто и быстро;
2. использование языка python;
3. Особенности Jinja2: мощная автоматическая система HTML, наследование шаблонов;
4. паттерн MVT(разновидность MVC), который предоставляет ряд преимуществ при разработке продукта.
5. Бесплатность данной библиотеки и языка программирования.

В качестве СУБД выбран PostgreSQL. Такой выбор был осуществлён, так как данная СУБД обладает рядом преимуществ:

1. PostgreSQL не просто реляционная, а объектно-реляционная СУБД. Это даёт ей некоторые преимущества над другими SQL базами данных с открытым исходным кодом;
2. расширения - существует возможность расширения функционала за счет использования хранимых процедур, триггеров, представлений, составных типов данных;
3. существует довольно большое сообщество, в котором можно запросто найти ответы на различные вопросы, а также хорошо проработанная документация;
4. PostgreSQL – бесплатная СУБД с открытым кодом;
5. существует множество API под разные платформы.

Во время разработки использовались также следующие средства программирования:

1. psycopg2 – библиотека предоставляющая возможность языку python взаимодействовать с API postgresql и исполнять запросы;
2. sqlalchemy– библиотека, которая предоставляет полезные настройки по умолчанию и дополнительных помощников, которые облегчают выполнение обычных задач;

5 СОЗДАНИЕ БАЗЫ ДАННЫХ

Подробно рассмотрим запрос на создание одной таблицы БД –public.product.

CREATE TABLE Product(

Product\_ID SERIAL PRIMARY KEY,

Product\_Kind VARCHAR(50) NOT NULL CHECK(Product\_Kind IN('Стерилизованное','Не стерилизованное','Твердый','Мягкий','Плавленный')),

Product\_Name VARCHAR(50) NOT NULL,

Product\_Price NUMERIC(10,2) NOT NULL,

Product\_Type VARCHAR(50) NOT NULL CHECK(Product\_Type IN('Молоко','Сыр')),

Subdivision\_ID INT NOT NULL references Subdivision(Subdivision\_ID),

description varchar (100) not null,

product\_date date not null,

required\_feed numeric(10) not null,

required\_fuel numeric (10) not null,

required\_med numeric (10) not null

);

Создание таблицы происходит с помощью команды CREATE TABLE. Поле product\_id является первичным ключем таблицы product. Поле subdivision\_id является внешним ключом на таблицуsubdivision.

Product\_type – тип продукции (не может быть пустым значением) и принимает только определенные значения( молоко или сыр ). Product\_date и required\_feed не могут быть пустыми значениями). В данной таблице абсолютно всем атрибутам сопутствует выражение NOT NULL, которое означает, что кортеж не может быть создан, пока все значения не будут заполнены.

Все остальные таблицы созданы с использованием соответствующих запросов CREATE TABLE, приведенных в Приложении Б.

# РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БАЗЕ ДАННЫХ

Как уже было упомянуто выше, в данной информационной системе присутствуют четыре роли. Рассмотрим их подробнее с точки зрения прав на таблицы базы данных:

1. директор– имеет возможность выполнять все CRUD операции над всеми таблицами схемы public;
2. сотрудник – имеет возможность выполнять все CRUD операции над всеми таблицами схемы public;
3. клиент – может производить чтение сущностей

сотрудник, рабочая смена, услуга;

1. гость – может производить CRUD операции над таблицей client(колонки логин и пароль) и таблицей staff(колонки логин, пароль и роль).   
   необходимые для установки роли

Более подробно запросы на создание и наделение правами пользователей можно посмотреть в приложении Г.

# 

# 7 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ИНФОРМАЦИОННОЙ СИСТЕМЫ

Для реализации информационной системы написан ряд классов, которые решают различные задачи в зависимости от слоя, в котором они находятся. Рассмотрим основные классы на каждом из слоёв:

1. BLL – position, client. Класс создающий модель БД.
2. DAL – модуль app.py. app py отвечает за доступ к БД частичную обработку возвращенную запросами, расспределениями URL адресов.
3. PL – html страницы. .Позволяют пользоваиелю работать с системой.

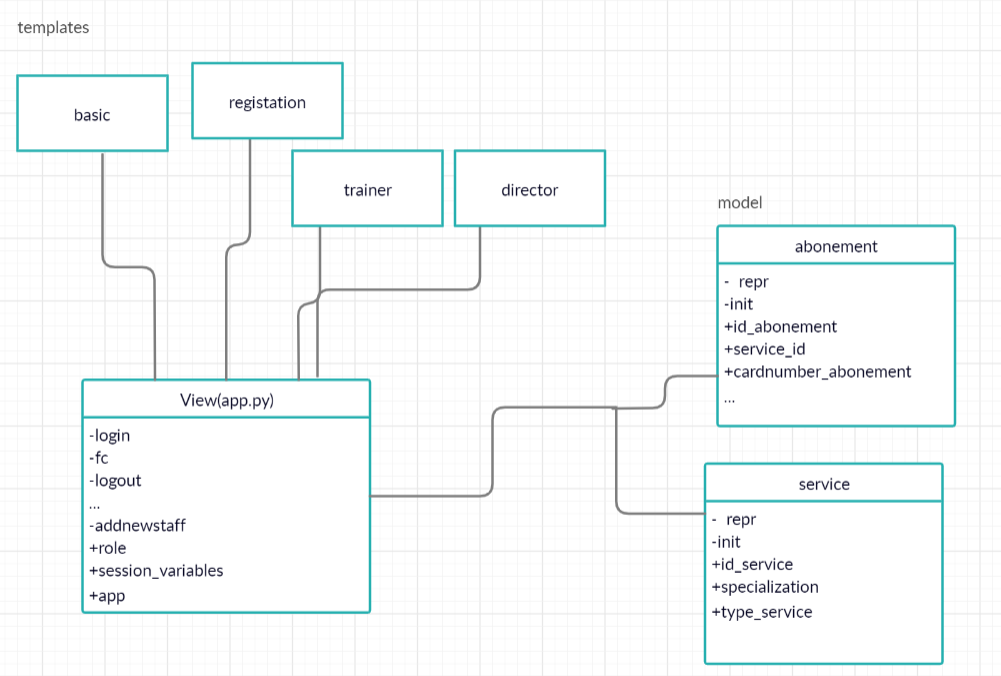


Рисунок 7.1 - Диаграмма классов

# ЗАПРОСЫ К БАЗЕ ДАННЫХ

Технология psycopg2 подразумевает написание чистых SQL запросов. Рассмотрим ниже примеры запросов выполненных в коде с помощью форматированных строк.

Ниже рассмотрены решения для нескольких пользовательских задач, полный список задач находится в приложении А.

Для решения задачи D1 был использован следующие запрос:

CREATE OR REPLACE FUNCTION addnewstaff(stname VARCHAR, surname VARCHAR, patr VARCHAR, tel VARCHAR,

stlogin VARCHAR, rassw VARCHAR, serv VARCHAR, typeserv VARCHAR, pos VARCHAR) RETURNS INT AS $$

DECLARE

idserv INT;

userrole INT;

sal NUMERIC;

BEGIN

idserv := (SELECT id\_service FROM service WHERE specialization\_service=$7 AND type\_service=$8);

IF $9='Директор' THEN userrole = 1;

sal = 30000;

ELSIF $9='Администратор фитнес клуба' THEN userrole = 2;

sal = 22000;

ELSIF $9='Тренер' THEN userrole = 3;

sal = 15000;

ELSIF $9='Массажист' THEN userrole = 3;

sal = 10000;

END IF;

INSERT INTO staff (service\_id, name\_staff, surname\_staff, patronomyc\_staff,

mobile\_telephone\_staff, position\_staff, salary\_₴, login, passw, role\_to\_login)

VALUES (idserv, $1, $2, $3, $4, $9, sal, $5, $6, userrole);

RETURN 1;

END;

$$ LANGUAGE plpgsql;

Для решения задачи D2 был использован следующий запрос:

CREATE OR REPLACE FUNCTION deletestaff(stname VARCHAR, surname VARCHAR, patr VARCHAR) RETURNS INT AS $$

BEGIN

DELETE FROM staff

WHERE name\_staff=$1 AND surname\_staff=$2 AND patronomyc\_staff=$3;

RETURN 1;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM deletestaff('Алексей','Григорьев','Сергеевич');

Для решения задачи A1 был использован следующий запрос:

CREATE OR REPLACE FUNCTION addnewclient(cname VARCHAR, surname VARCHAR, patr VARCHAR, tel VARCHAR) RETURNS TABLE

("Номер карты" INT,"Имя" VARCHAR, "Фамилия" VARCHAR, "Отчество" VARCHAR, "Телефон" CHAR) AS $$

DECLARE

cardn INT;

BEGIN

cardn = (SELECT max(card\_number\_client) FROM client)+1;

INSERT INTO client (card\_number\_client,name\_client, surname\_client, patronomyc\_client, mobile\_telephone\_client) VALUES

(cardn, $1, $2, $3, $4);

RETURN QUERY(SELECT card\_number\_client, name\_client, surname\_client, patronomyc\_client, mobile\_telephone\_client FROM client WHERE card\_number\_client=cardn);

END;

$$ LANGUAGE plpgsql

9 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

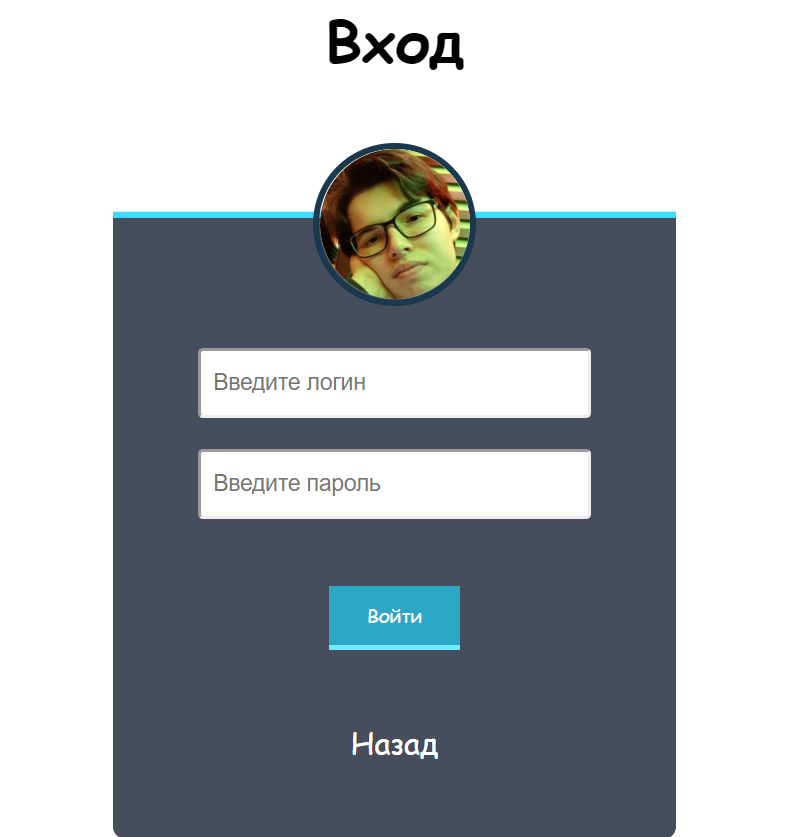
При запуске программы пользователь видит простое окно, содержащее поля для ввода – логин и пароль – а также кнопку подтверждения(рис.9.1) 

Рисунок 9.1 – Страница входа

Так как в рассматриваемой информационной системе существуют несколько ролей, то для каждой из них есть свои окна и свой набор функций.

Рассмотрим главное окно для директора (рис 9.2). В данной системе только директор имеет право регситрировать новых пользователей системы. Для регистрации нового пользователя достаточно ввести его данные и нажать на кнопку подтверждения



Рисунок 9.2 – Страница директора

После успешной регистрации(рис 9.3) можно, посмотрев на список текущих пользователей, увидеть только что созданногопользователя (рис. 9.4).

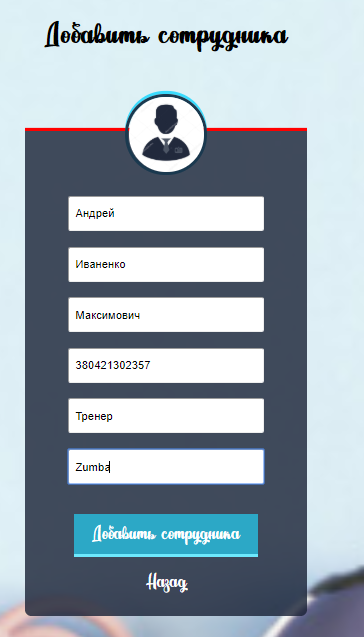


Рисунок 9.3 – Список всех пользователей

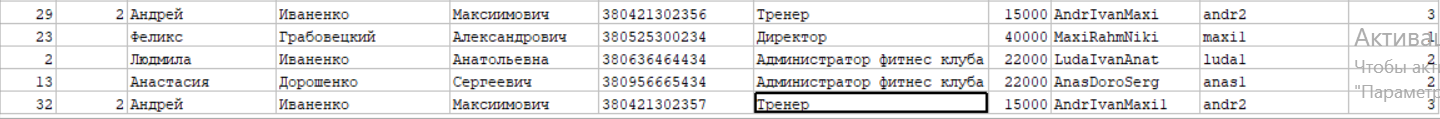


Рис 9.4 – Созданный пользователь

Перейдем к одной из основных задач фитнеса – добавления клиента и ответсвенных лиц в базу данных фитнес центра. Для этого присутствует интерфейс отображенный на рис. 9.5.

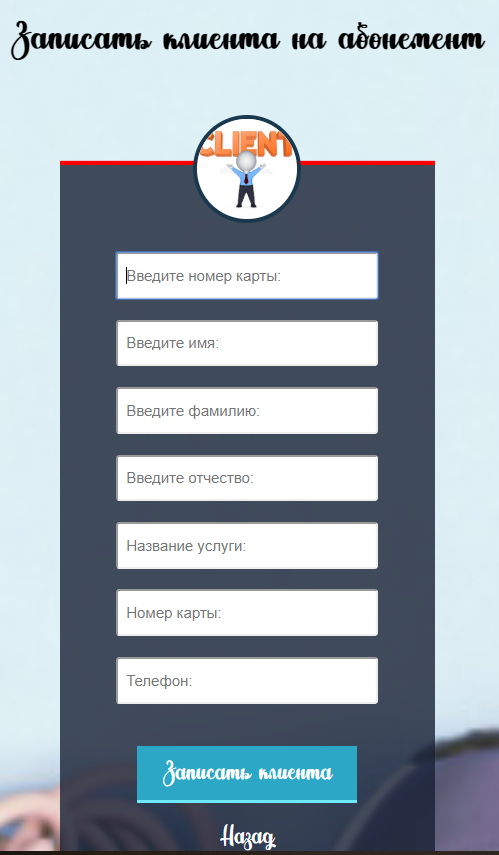


Рисунок 9.5 - Добавление ребенка

Когда администратор хочет получить данные о клиенте он может нажать кнопку «Получить данные о клиенте». Интерфейс окна кторого предоставлен на рис. 9.6.

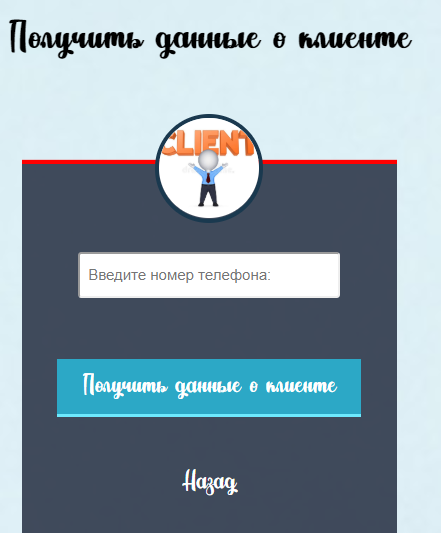


Рисунок 9.6 – Данные о клиенте

Перейдя под наименее привелигированного пользователя – преподователя мы увидим , что часть функицонала пропала. Оображено на рис. 9.6.

ЗАКЛЮЧЕНИЕ

В результате анализа предметной области, проведенного при выполнении курсовой работы, сформулированы цели создания информационной системы фитнес центра, определен перечень её пользователей (директор, администратор, тренер ,гость) и задач, которые они решают в рассматриваемой предметной области, в виде пользовательских историй. Определены требования к хранимым данным и спроектирована база данных из 10 таблиц для использования в информационной системе.

Для создания информационной системы фитнес центра выбраны трёхзвенная архитектура клиент-сервер, шаблон проектирования MVT, СУБД PostgreSQL, язык программирования python, и микрофреймворк flask для разработки пользовательского интерфейса.

Создан веб-сайт , который даёт возможность удобно манипулировать данными предметной области. Кроме того, разграничен доступ к приложению со стороны различных категорий пользователей путем использования механизма ролей и привилегий и реализована защита от несанкционированного доступа путем использования механизма аутентификации и авторизации.

За счёт использования в созданной информационной системе трёхзвенной архитектуры клиент-сервер достигнута гибкость и слабая зависимость между слоями программного обеспечения, вследствие чего приложение является легко расширяемым. Внедрение новых задач по мере появления в них необходимости будет являться проблемой.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. User Stories [Электронный ресурс] – Режим доступа: https://www.mountaingoatsoftware.com/agile/user-stories. – 25.05.2017
2. Малахов Е.В. Организация баз данных: конспект. – О.: ОНУ, 2015–169 с.
3. Паттерны для новичков: MVC и MVT [Электронный ресурс] – Режим доступа: https://www.quora.com/What-is-the-difference-between-an-MVC-and-an-MVT-framework
4. Швецов В.И. Базы Данных. – НОУ ИНТУИТ | Базы данных | Информация [Электронный ресурс] – Режим доступа: http://www.intuit.ru/studies/ courses/508/364/info – 15.05.2017

# ПРИЛОЖЕНИЕ А

# СПИСОК ЗАДАЧ ПОЛЬЗОВАТЕЛЕЙ ИНФОРМАЦИОННОЙ СИСТЕМЫ

ДИРЕКТОР

Д1) Добавить сотрудника в базу данных:

.необходимо знать : ФИО, телефон должность сотрудника на которую он претендует, название услуги которую он хочет вести,

.на выходе : создание личного кабинета сотрудника

Д2) Удалить сотрудника в базе данных :

.необходимо знать : ФИО сотрудника, номер телефона

.на выходе: Новый список сотрудников с учетом удаления предыдущего

Д3) Посмотреть доступные услуги :

.на выходе: список всех доступных услуг фитнес центра

Д4) Добавить услугу :

.необходимо знать : название услуги, вид услуги

.на выходе: список всех доступных услуг с учетом добавления новой

Д5) Удалить услугу:

.необходимо знать: название услуги

.на выходе: список всех доступных услуг с учетом удаления предыдущей

Д6) Просмотреть информацию о рабочей смене:

.необходимо знать : дата о которой мы хотим узнать информацию

.на выходе : день недели рабочей смены, время начала смены, время конца смены, ФИО сотрудника кто работает в эту смену

АДМИНИСТРАТОР

А1) Добавить клиента в базу данных спортзала:

.необходимо знать : ФИО клиента, телефон клиента

.на выходе: список клиентов фитнес клуба с учетом добавления нового клиента

А2) Получить данные о клиенте

.необходимо знать: ФИО клиента, телефон клиента

.на выходе: посещаемые абонементы, дата начала абонемента, дата конца абонемента

А3) Записать клиента на абонемент:

.необходимо знать : ФИО клиента, номер телефона клиента, название услуги

.на выходе : Список людей, которые приобрели эту услугу

А4) Закрыть абонемент клиенту:

.необходимо знать : название услуги, ФИО клиента, номер телефона

А5) Получить данные о посещении клиента

.необходимо знать : ФИО клиента, номер телефона

.на выходе: список текущих актуальных абонементов клиента

КЛИЕНТ

К1) Просмотр истории покупок за период времени

.необходимо знать : дата начала периода, дата конца периода

.на выходе: номер товара, наименование товара, тип товара, номер отделения которое занималось этим товаром, стоимость товара, количество товара

К2) Просмотр отделов в своем городе

.необходимо знать : город

.на выходе: номер отделения, адрес, директор

К3) Просмотр стоимости отдельного товара

.необходимо знать : наименование товара

.на выходе: цена, количество

К4) Сделать заказ

.необходимо знать : имя товара, количество товара

# ПРИЛОЖЕНИЕ Б

# ЗАПРОСЫ НА СОЗДАНИЕ БАЗЫ ДАННЫХ

CREATE TABLE Subdivision(

Subdivision\_ID SERIAL PRIMARY KEY,

Chief\_first\_name VARCHAR(50) NOT NULL,

Chief\_last\_name VARCHAR(50) NOT NULL,

Street VARCHAR(100) NOT NULL,

Quanity\_of\_produced NUMERIC(10)

);

CREATE TABLE Proffesion(

Proffesion\_ID SERIAL PRIMARY KEY,

Name\_of\_prof VARCHAR(50) NOT NULL UNIQUE

);

CREATE TABLE Client(

Client\_ID SERIAL PRIMARY KEY ,

Last\_name VARCHAR(50) NOT NULL,

First\_name VARCHAR(50) NOT NULL,

Mobile\_number CHAR(12) UNIQUE,

Adress\_of\_Client VARCHAR(50) NOT NULL,

Login text not null,

Passw text not null

);

CREATE TABLE Staff(

Staff\_ID SERIAL PRIMARY KEY,

Last\_name VARCHAR(50) NOT NULL,

First\_name VARCHAR(50) NOT NULL,

Phone\_number CHAR(12) UNIQUE,

Work\_role INT NOT NULL references Proffesion(Proffesion\_ID) ,

Salary NUMERIC(7,2) NOT NULL,

Quantity\_of\_products\_produced NUMERIC(10),

Subdivision\_ID INT NOT NULL references Subdivision(Subdivision\_ID),

Adress VARCHAR(50) NOT NULL,

cheese\_equipment boolean not null,

milk\_equipment boolean not null,

login text not null unique,

passw text not null,

role\_ text not null check(role\_ in (‘staff’, ‘director’)

);

CREATE TABLE Product(

Product\_ID SERIAL PRIMARY KEY,

Product\_Kind VARCHAR(50) NOT NULL CHECK(Product\_Kind IN('Стерилизованное','Не стерилизованное','Твердый','Мягкий','Плавленный')),

Product\_Name VARCHAR(50) NOT NULL,

Product\_Price NUMERIC(10,2) NOT NULL,

Product\_Type VARCHAR(50) NOT NULL CHECK(Product\_Type IN('Молоко','Сыр')),

Subdivision\_ID INT NOT NULL references Subdivision(Subdivision\_ID),

description varchar (100) not null,

product\_date date not null,

required\_feed numeric(10) not null,

required\_fuel numeric (10) not null,

required\_med numeric (10) not null

);

CREATE TABLE Order\_(

Order\_ID SERIAL CHECK(Order\_ID>0) PRIMARY KEY,

Date\_ DATE NOT NULL,

Client\_ID INT NOT NULL references Client(Client\_ID)

);

CREATE TABLE Supply(

Order\_ID SERIAL references Order\_(Order\_ID),

Product\_ID INT references Product(Product\_ID),

Product\_Price NUMERIC(10,2) NOT NULL,

Product\_Count NUMERIC(10) NOT NULL,

PRIMARY KEY(Order\_ID, Product\_ID)

);

create table order\_resource (

order\_resource\_id serial primary key,

resource\_type varchar(50) not null check (resource\_type in ('корм', 'медикаменты', 'топливо')),

resource\_price numeric (10,2) not null,

resource\_count numeric (10) not null,

subdivision\_id int not null references Subdivision (Subdivision\_ID)

);

# 

# ПРИЛОЖЕНИЕ В

Запрос для добавления клиента

DROP FUNCTION insert\_new\_client;

CREATE OR REPLACE FUNCTION insert\_new\_client(last\_name\_client varchar, first\_name\_client varchar,

mobile\_number\_client varchar, adress\_of\_client\_client varchar, login\_client varchar, passw\_client varchar) RETURNS INT AS $$

INSERT INTO client(last\_name, first\_name, mobile\_number, adress\_of\_client, login, passw) VALUES

(last\_name\_client, first\_name\_client, mobile\_number\_client, adress\_of\_client\_client, login\_client, passw\_client);

SELECT max(client\_id) FROM client;

$$LANGUAGE sql;

--редактирование клиента в базе

DROP FUNCTION update\_client;

CREATE OR REPLACE FUNCTION update\_client(last\_name\_client varchar, first\_name\_client varchar,

mobile\_number\_client varchar, adress\_of\_client\_client varchar) RETURNS VOID AS $$

UPDATE client

set

client.first\_name = first\_name\_client,

client.last\_name = last\_name\_client,

client.mobile\_number = mobile\_number\_client,

client.adress\_of\_client = adress\_of\_client\_client

WHERE

client.first\_name = first\_name\_client AND

client.last\_name = last\_name\_client

$$ LANGUAGE sql;

SELECT \* FROM insert\_new\_client('surname', 'name', 'phone', 'adress', 'login\_', 'password' );

Запрос для обновления клиента

CREATE OR REPLACE FUNCTION update\_client(last\_name\_client varchar, first\_name\_client varchar,

mobile\_number\_client varchar, adress\_of\_client\_client varchar) RETURNS VOID AS $$

UPDATE client

set

client.first\_name = first\_name\_client,

client.last\_name = last\_name\_client,

client.mobile\_number = mobile\_number\_client,

client.adress\_of\_client = adress\_of\_client\_client

WHERE

client.first\_name = first\_name\_client AND

client.last\_name = last\_name\_client

$$ LANGUAGE sql;

SELECT \* FROM update\_client('name', 'surname', 'phone', 'adres');

Запрос для просмотра наличия и количества товара

drop FUNCTION check\_product;

CREATE OR REPLACE FUNCTION check\_product(name\_product text) RETURNS

TABLE (product\_id int, product\_name varchar, product\_type varchar,

subdivision\_id int, product\_price numeric)

as $$

SELECT product\_id, product\_name, product\_type,

subdivision\_id, product\_price

from product

JOIN subdivision USING(subdivision\_id)

WHERE product\_name=name\_product;

$$ language sql;

select \* from check\_product('Счастье');

Запрос на просмотр информации о клиенте

CREATE OR REPLACE FUNCTION addserv(spec VARCHAR, type VARCHAR ) RETURNS TABLE("Номер" INT, "Услуга" VARCHAR, "Вид" VARCHAR) AS $$

BEGIN

INSERT INTO service(specialization\_service, type\_service)

VALUES ($1,$2);

RETURN QUERY(SELECT \* FROM servicesfordirector);

END;

$$ LANGUAGE plpgsql

SELECT \* from client\_info WHERE last\_name = '' and first\_name = '';

Запрос на удаление услуги

CREATE OR REPLACE FUNCTION delserv(spec VARCHAR, type VARCHAR ) RETURNS TABLE("Номер" INT, "Услуга" VARCHAR, "Вид" VARCHAR) AS $$

DECLARE

staff\_id int;

spec\_id int;

cost\_id int;

BEGIN

select id\_service into spec\_id from service WHERE type\_service = $2 and specialization\_service = $1;

FOR staff\_id IN SELECT id\_staff FROM staff WHERE service\_id = spec\_id loop

update staff

set service\_id = NUll

where id\_staff = staff\_id;

end loop;

FOR cost\_id IN SELECT id\_ca FROM cost\_ab WHERE serv\_id = spec\_id loop

update cost\_ab

set serv\_id = NUll

where id\_ca = cost\_id;

end loop;

DELETE FROM service WHERE specialization\_service=$1 AND type\_service=$2;

RETURN QUERY(SELECT \* FROM servicesfordirector);

END;

$$ LANGUAGE plpgsql;

SELECT\*FROMdelserv('Администрирование','Индвидуальная');

Запрос на информацию о рабочей смене

CREATE OR REPLACE FUNCTION wsinfo(d1 TIMESTAMP) RETURNS TABLE

("Имя" VARCHAR, "Фамилия" VARCHAR, "Отчество" VARCHAR, "День" VARCHAR, "Начало" TIME WITHOUT TIME ZONE, "Конец" TIME WITHOUT TIME ZONE) AS $$

DECLARE

d\_o\_w INT;

dayoftheweek VARCHAR;

BEGIN

d\_o\_w = (SELECT EXTRACT(DOW FROM $1));

IF d\_o\_w = 0 THEN dayoftheweek = 'Воскресенье';

ELSIF d\_o\_w = 1 THEN dayoftheweek = 'Понедельник';

ELSIF d\_o\_w = 2 THEN dayoftheweek = 'Вторник';

ELSIF d\_o\_w = 3 THEN dayoftheweek = 'Среда';

ELSIF d\_o\_w = 4 THEN dayoftheweek = 'Четверг';

ELSIF d\_o\_w = 5 THEN dayoftheweek = 'Пятница';

ELSIF d\_o\_w = 6 THEN dayoftheweek = 'Суббота';

END IF;

RETURN QUERY(SELECT name\_staff, surname\_staff, patronomyc\_staff, date\_workshift,

begin\_time\_workshift, end\_time\_workshift FROM workshift JOIN staff ON workshift\_staff=id\_staff WHERE date\_workshift=dayoftheweek);

END;

$$ LANGUAGE plpgsql

SELECT \* FROM wsinfo(now()::TIMESTAMP);

Запрос на добавление клиента

CREATE OR REPLACE FUNCTION addnewclient(cname VARCHAR, surname VARCHAR, patr VARCHAR, tel VARCHAR) RETURNS TABLE

("Номер карты" INT,"Имя" VARCHAR, "Фамилия" VARCHAR, "Отчество" VARCHAR, "Телефон" CHAR) AS $$

DECLARE

cardn INT;

BEGIN

cardn = (SELECT max(card\_number\_client) FROM client)+1;

INSERT INTO client (card\_number\_client,name\_client, surname\_client, patronomyc\_client, mobile\_telephone\_client) VALUES

(cardn, $1, $2, $3, $4);

RETURN QUERY(SELECT card\_number\_client, name\_client, surname\_client, patronomyc\_client, mobile\_telephone\_client FROM client WHERE card\_number\_client=cardn);

END;

$$ LANGUAGE plpgsql

SELECT \* FROM addnewclient('Максим', 'Степаненко', 'Антонович', '380432113060');

Информация о клиенте

CREATE OR REPLACE FUNCTION infoclient(tel VARCHAR) RETURNS TABLE

("Услуга" VARCHAR, "Дата начала" DATE, "Дата конца" DATE) AS $$

BEGIN

RETURN QUERY

(SELECT specialization\_service, begindate\_abonement, enddate\_abonement FROM abonement

JOIN service ON service\_id=id\_service

JOIN client ON cardnumber\_abonement=card\_number\_client

WHERE mobile\_telephone\_client=$1);

END;

$$ LANGUAGE plpgsql

SELECT \* FROM infoclient('380956435896');

Записать клиента на абонемент

CREATE OR REPLACE FUNCTION recab(tel CHAR, serv VARCHAR, months INT) RETURNS int AS $$

DECLARE

cardn INT;

servid INT;

servab INT;

edate DATE;

r INT;

am INTERVAL;

BEGIN servid = (SELECT id\_service FROM service WHERE specialization\_service=$2);

cardn = (SELECT card\_number\_client FROM client WHERE mobile\_telephone\_client=$1);

servab = (SELECT id\_ca FROM cost\_ab WHERE durationinmonths=$3 and serv\_id=servid );

r := $3;

am = r || ' months';

edate = now()::date + am;

INSERT INTO abonement (service\_id, cardnumber\_abonement, begindate\_abonement, enddate\_abonement, isactive\_abonement, cost\_abonement)

VALUES (servid, cardn, now()::date, edate, 't', servab);

RETURN 1;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM recab('380956435896', 'Zumba', 6);

Закрыть абонемент

CREATE OR REPLACE FUNCTION closeab(tel CHAR, serv VARCHAR) RETURNS int AS $$

BEGIN

UPDATE abonement

SET enddate\_abonement=now()::date

WHERE id\_abonement=(

SELECT id\_abonement from abonement

JOIN client ON cardnumber\_abonement=card\_number\_client

JOIN service ON service\_id=id\_service

WHERE mobile\_telephone\_client=$1 AND specialization\_service=$2);

RETURN 1;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM closeab('380956435896', 'Zumba');

Расписание тренера

CREATE OR REPLACE FUNCTION closeab(tel CHAR, serv VARCHAR) RETURNS int AS $$

BEGIN

UPDATE abonement

SET enddate\_abonement=now()::date

WHERE id\_abonement=(

SELECT id\_abonement from abonement

JOIN client ON cardnumber\_abonement=card\_number\_client

JOIN service ON service\_id=id\_service

WHERE mobile\_telephone\_client=$1 AND specialization\_service=$2);

RETURN 1;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM closeab('380956435896', 'Zumba');

Посещение клиента

select name\_client || ' ' || surname\_client || ' ' || patronomyc\_client as client\_fio ,

day\_record, time\_beginrecord, time\_endrecord from client

join public.record on cardnumber\_client\_record = card\_number\_client

where card\_number\_client = 51322

# ПРИЛОЖЕНИЕ Г

# ЗАПРОСЫ НА СОЗДАНИЕ ПОЛЬЗОВАТЕЛЕЙ БД

CREATE USER farm\_developer WITH LOGIN CREATEROLE PASSWORD 'developer';

CREATE USER farm\_director WITH LOGIN CREATEROLE PASSWORD 'director';

CREATE USER farm\_staff WITH LOGIN PASSWORD 'staff';

CREATE USER farm\_client WITH LOGIN PASSWORD 'client';

CREATE USER farm\_guest WITH LOGIN PASSWORD 'guest';

REVOKE ALL on DATABASE farm FROM farm\_developer;

REVOKE ALL ON SCHEMA public FROM farm\_developer;

REVOKE CREATE ON SCHEMA public FROM public;

REVOKE ALL ON DATABASE farm FROM public;

grant all PRIVILEGES on schema public to farm\_developer;

grant all PRIVILEGES on schema public to farm\_director;

GRANT CREATE ON SCHEMA public to farm\_developer;

GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES, TRIGGER ON ALL TABLES IN SCHEMA public

TO farm\_developer;

GRANT SELECT, INSERT, UPDATE, DELETE, REFERENCES, TRIGGER ON ALL TABLES IN SCHEMA public

TO farm\_director;

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO farm\_director;

GRANT USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO farm\_developer;

GRANT CONNECT ON DATABASE farm to farm\_director;

GRANT CONNECT ON DATABASE farm to farm\_developer;

GRANT CONNECT ON DATABASE farm to farm\_staff;

GRANT CONNECT ON DATABASE farm to farm\_client;

GRANT CONNECT ON DATABASE farm to farm\_guest;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE staff

TO farm\_guest;

grant select on client\_info to farm\_staff;

GRANT SELECT ON TABLE client

TO farm\_guest;

GRANT SELECT, REFERENCES ON TABLE

public.subdivision,

public.client,

public.order\_,

public.supply,

public.product

TO farm\_client;

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE

public.client,

public.order\_,

public.supply,

public.product,

TO farm\_staff;

GRANT SELECT ON TABLE

public.subdivision,

public.order\_resource,

public.staff

TO farm\_staff;

# ПРИЛОЖЕНИЕ Д

# ИСХОДНЫЙ КОД БАЗОВЫХ КЛАССОВ

ViewModel

from flask import Flask, render\_template, url\_for, redirect, request, session, flash, get\_flashed\_messages, abort

from sqlalchemy import create\_engine, MetaData

import psycopg2

from flask\_sqlalchemy import SQLAlchemy

from sqlalchemy.orm import scoped\_session, sessionmaker

db = SQLAlchemy()

app = Flask(\_\_name\_\_)

app.secret\_key = 'somesecretkeythatonlyishouldknow'

session\_variables = []

role = 'farm\_guest:guest'

def execute\_query(user, password, query):

conn = psycopg2.connect(

host="127.0.0.1",

database="farm",

user=user,

password=password

)

cursor = conn.cursor()

cursor.execute(query)

conn.commit()

conn.close()

def execute\_select\_query(user, password, query, f\_all=True):

conn = psycopg2.connect(

host="127.0.0.1",

database="farm",

user=user,

password=password

)

cursor = conn.cursor()

cursor.execute(query)

if f\_all:

data = cursor.fetchall()

else:

data = cursor.fetchone()

conn.close()

return data

def loadSession(role):

engine = create\_engine(

f'postgres+psycopg2://{role}@localhost:5432/farm', convert\_unicode=True)

# metadata = MetaData()

db\_session = scoped\_session(sessionmaker(

autocommit=False, autoflush=False, bind=engine))

metadata = db.metadata

session\_variables.append(engine)

session\_variables.append(db\_session)

session\_variables.append(metadata)

Session = sessionmaker(bind=engine)

session\_ = Session()

return session\_

def shutdown\_session(exception=None):

global session\_variables

session\_variables[1].remove()

# --------------404 PAGE------------------

@app.errorhandler(404)

def pageNotFound(error):

return "<h1>You got 404 mistake please get on correct url adres</h1>"

# ---------------------------------------

# ------------------------LOGIN----------------------------

@app.route('/login', methods=['POST', 'GET'])

def login():

global role

if 'login' and 'username' in session:

if session['login'] == 'staff':

try:

shutdown\_session()

except Exception as e:

pass

session['login'] = 'farm\_staff'

role = 'farm\_staff:staff'

return redirect(url\_for('admin', username=session['username']))

elif session['login'] == 'director':

try:

shutdown\_session()

except Exception as e:

pass

session['login'] = 'farm\_director'

role = 'farm\_director:director'

return redirect(url\_for('director', username=session['username']))

elif session['login']:

try:

shutdown\_session()

except Exception as e:

pass

session['login'] = 'farm\_client'

role = 'farm\_client:client'

return redirect(url\_for('client', username=session['username']))

if request.method == 'POST':

username = request.form["username"]

password = request.form['password']

session\_ = loadSession('farm\_guest:guest')

query = f"SELECT role\_ FROM staff WHERE login = '{username}' AND passw = '{password}' ;"

print(query)

try:

session['login'] = session\_.execute(query).fetchone()[0]

except Exception as e:

try:

query = f"SELECT client\_id FROM client WHERE login = '{username}' AND passw = '{password}' ;"

print(query)

session['login'] = session\_.execute(query).fetchone()[0]

print(session['login'])

if session['login'] > 0:

session['username'] = username

role = 'farm\_staff:staff'

return redirect(url\_for('client', username=session['username']))

except Exception as e:

print(e)

shutdown\_session()

flash("Неверный логин или пароль")

return render\_template('Registation.html')

# return f"{e}"

if session['login'] == 'staff':

try:

shutdown\_session()

except Exception as e:

pass

session['username'] = username

role = 'farm\_staff:staff'

return redirect(url\_for('admin', username=session['username']))

elif session['login'] == 'director':

try:

shutdown\_session()

except Exception as e:

pass

session['username'] = username

role = 'farm\_director:director'

return redirect(url\_for('director', username=session['username']))

elif session['login'] == 'client':

try:

shutdown\_session()

except Exception as e:

pass

session['username'] = username

role = 'farm\_client:client'

return redirect(url\_for('client', username=session['username']))

else:

flash("Неверный логин или пароль")

return render\_template('Registation.html')

return render\_template('Registation.html')

@app.route('/logout', methods=['POST', 'GET'])

def logout():

try:

shutdown\_session()

except Exception as e:

pass

try:

del session['login']

except Exception as e:

pass

try:

del session['username']

except Exception as e:

pass

print(session)

return redirect(url\_for('login'))

# -------------------------------------------------------------------

# --------------------------DIRECTOR PAGES---------------------------

@app.route('/director/<username>', methods=['GET'])

def director(username):

global role

if 'username' not in session or session['username'] != username:

abort(401)

role = "farm\_director:director"

session\_ = loadSession(role)

data1 = session\_.execute(

"SELECT \* from staff WHERE role\_='director';")

data1 = data1.first()

print(data1)

return render\_template('Director.html', dirstaff=data1, username=session['username'])

@app.route('/director/addstaff/<username>', methods=['POST', 'GET'])

def directoraddstaff(username):

if 'username' not in session or session['username'] != username:

abort(401)

try:

shutdown\_session()

except Exception as e:

pass

if request.method == 'POST':

username = request.form["name\_login"]

surname = request.form['surname']

work\_role = request.form['work\_role']

telephone = request.form['telephone']

salary = request.form['salary']

quantity = request.form['quantity']

subdivision = request.form['subdivision']

adres = request.form['adres']

position = request.form['position']

login\_ = request.form['login']

password = request.form['password']

session\_ = loadSession('farm\_director:director')

query = f"""

INSERT INTO Staff(First\_name, Last\_name, Phone\_number, Work\_role, Quantity\_of\_products\_produced,

Salary, Subdivision\_ID, Adress, cheese\_equipment, milk\_equipment, login, passw, role\_)

values('{username}','{surname}','{telephone}','{work\_role}','{quantity}','{salary}','{subdivision}','{adres}','false','true', '{login\_}', '{password}', '{position}');

"""

try:

execute\_query('farm\_director', 'director', query)

return redirect(url\_for('directoraddstaffres', username=session['username']))

except Exception as e:

# return render\_template('Add\_Staff.html')

return render\_template("Add\_Staff.html", username=session['username'])

else:

return render\_template("Add\_Staff.html", username=session['username'])

@app.route('/director/update\_staff/<username>', methods=['POST', 'GET'])

def directorupdatestaff(username):

# fix render template!!

if 'username' not in session or session['username'] != username:

abort(401)

try:

shutdown\_session()

except Exception as e:

pass

if request.method == 'POST':

staff\_id = request.form["staff\_id"]

username = request.form["name\_login"]

surname = request.form['surname']

work\_role = request.form['work\_role']

telephone = request.form['telephone']

salary = request.form['salary']

quantity = request.form['quantity']

subdivision = request.form['subdivision']

adres = request.form['adres']

position = request.form['position']

login\_ = request.form['login']

password = request.form['password']

session\_ = loadSession('farm\_director:director')

query = f'''

UPDATE staff

set

First\_name = '{username}',

Last\_name = '{surname}',

Phone\_number = '{telephone}',

Work\_role = '{work\_role}',

Quantity\_of\_products\_produced = '{quantity}',

Salary = '{salary}',

Subdivision\_ID = '{subdivision}',

Adress = '{adres}',

login = '{login\_}',

passw = '{password}',

role\_ = '{position}'

where staff\_id = {staff\_id};

'''

print(query)

try:

execute\_query('farm\_director', 'director', query)

return redirect(url\_for('directoraddstaffres', username=session['username']))

except Exception as e:

return render\_template("update\_staff.html", username=session['username'])

else:

return render\_template("update\_staff.html", username=session['username'])

@app.route('/director/addstaff\_result/<username>', methods=['POST', 'GET'])

def directoraddstaffres(username):

global role

role = "farm\_director:director"

if 'username' not in session or session['username'] != username:

abort(401)

else:

query = """select \* from staff;"""

data = execute\_select\_query('farm\_director', 'director', query)

return render\_template("Add\_Staff2.html", username=session['username'], data=data)

@app.route('/director/delete/<username>', methods=('POST', 'GET'))

def directordelete(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

number = request.form['number']

query = f"DELETE from staff where staff\_id = {number};"

try:

execute\_query('farm\_director', 'director', query)

return redirect(url\_for('directoraddstaffres', username=session['username']))

except Exception as e:

return render\_template("Delete.html", username=session['username'])

return render\_template("Delete.html", username=session['username'])

@app.route('/director/check\_products\_spent/<username>', methods=('POST', 'GET'))

def director\_spent\_products(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

date\_begin = request.form['date\_begin']

date\_end = request.form['date\_end']

query = f"select \* from sold\_product('{date\_begin}', '{date\_end}');"

try:

data = execute\_select\_query('farm\_director', 'director', query)

return render\_template("spent\_products.html", username=session['username'], data=data)

except Exception as e:

pass

return render\_template("spent\_products.html", username=session['username'])

return render\_template("spent\_products.html", username=session['username'])

@app.route('/director/check\_products/<username>', methods=('POST', 'GET'))

def director\_check\_products(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

name = request.form['name']

query = f"select \* from check\_product('{name}');"

try:

data = execute\_select\_query('farm\_director', 'director', query)

return render\_template("check\_products.html", username=session['username'], data=data)

except Exception as e:

# return render\_template('Add\_Staff.html')

return render\_template("check\_products.html", username=session['username'])

return render\_template("check\_products.html", username=session['username'])

@app.route('/director/staff\_activity/<username>', methods=('POST', 'GET'))

def director\_staff\_activity(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

number = request.form['number']

query = f"""SELECT staff\_id, last\_name || ' ' || first\_name as fio, chief\_first\_name || ' ' || chief\_last\_name as chief\_fio, quantity\_of\_products\_produced

from staff

join subdivision using(subdivision\_id)

where staff\_id = {number};"""

try:

data = execute\_select\_query('farm\_director', 'director', query)

return render\_template("staff\_activity.html", username=session['username'], data=data)

except Exception as e:

pass

return render\_template("staff\_activity.html", username=session['username'])

@app.route('/director/subdiv\_activity/<username>', methods=('POST', 'GET'))

def director\_subdiv\_activity(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

adres = request.form['adres']

query = f"""

select chief\_first\_name || ' ' || chief\_last\_name as chief\_fio, product\_type, quanity\_of\_produced, sum(product\_count) as sold\_amount

from subdivision

join product USING(subdivision\_id)

join supply USING(product\_id)

where addres = '{adres}'

GROUP by 1,2,3;

"""

try:

data = execute\_select\_query('farm\_director', 'director', query)

return render\_template("subdiv\_activity.html", username=session['username'], data=data)

except Exception as e:

pass

return render\_template("subdiv\_activity.html", username=session['username'])

@app.route('/director/addsubdiv/<username>', methods=['POST', 'GET'])

def directoraddsubdiv(username):

if 'username' not in session or session['username'] != username:

abort(401)

try:

shutdown\_session()

except Exception as e:

pass

if request.method == 'POST':

adres = request.form["adres"]

city = request.form["city"]

name = request.form["name"]

surname = request.form["surname"]

quantity = request.form["quantity"]

query = f"""

INSERT INTO subdivision(chief\_first\_name, chief\_last\_name,city, quanity\_of\_produced, addres)

VALUES('{name}', '{surname}', '{city}', {quantity}, '{adres}')

"""

try:

execute\_query('farm\_director', 'director', query)

except Exception as e:

pass

return render\_template("addsubdiv.html", username=session['username'])

else:

return render\_template("addsubdiv.html", username=session['username'])

@app.route('/director/updatesubdiv/<username>', methods=['POST', 'GET'])

def directorupdatesubdiv(username):

if 'username' not in session or session['username'] != username:

abort(401)

try:

shutdown\_session()

except Exception as e:

pass

if request.method == 'POST':

sub\_id = request.form["sub\_id"]

adres = request.form["adres"]

city = request.form["city"]

name = request.form["name"]

surname = request.form["surname"]

quantity = request.form["quantity"]

query = f'''UPDATE subdivision

SET

chief\_first\_name = '{name}',

chief\_last\_name = '{surname}',

city = '{city}',

quanity\_of\_produced = '{quantity}',

addres = '{adres}'

WHERE subdivision\_id = {sub\_id};'''

try:

execute\_query('farm\_director', 'director', query)

except Exception as e:

pass

return render\_template("updatesubdiv.html", username=session['username'])

else:

return render\_template("updatesubdiv.html", username=session['username'])

@app.route('/director/deletesubdiv/<username>', methods=['POST', 'GET'])

def directordeletesubdiv(username):

if 'username' not in session or session['username'] != username:

abort(401)

try:

shutdown\_session()

except Exception as e:

pass

if request.method == 'POST':

sub\_id = request.form["sub\_id"]

query = f"delete from subdivision where subdivision\_id = {sub\_id};"

try:

execute\_query('farm\_director', 'director', query)

except Exception as e:

pass

return render\_template("deletesubdiv.html", username=session['username'])

else:

return render\_template("deletesubdiv.html", username=session['username'])

@app.route('/director/updatesalary/<username>', methods=['POST', 'GET'])

def directorupdatesalary(username):

if 'username' not in session or session['username'] != username:

abort(401)

try:

shutdown\_session()

except Exception as e:

pass

if request.method == 'POST':

number = request.form["number"]

salary = request.form['salary']

query = f"""UPDATE staff

set salary = {salary}

WHERE staff\_id = {number};"""

try:

execute\_query('farm\_director', 'director', query)

except Exception as e:

pass

return render\_template("updatesalary.html", username=session['username'])

else:

return render\_template("updatesalary.html", username=session['username'])

# -------------------------------------------------------------------

# --------------------------CLIENT PAGES-----------------------------

@app.route('/client/<username>')

def client(username):

if 'username' not in session or session['username'] != username:

abort(401)

session\_ = loadSession('farm\_client:client')

data1 = session\_.execute(f"SELECT \* from client WHERE login='{username}';")

data1 = data1.first()

print(data1)

return render\_template('Client.html', dirstaff=data1, username=session['username'])

@app.route('/client/check\_payment/<username>/', methods=('POST', 'GET'))

def client\_check\_payment(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

first\_date = request.form['first\_date']

last\_date = request.form['last\_date']

query = f"""select \* from client\_bought\_product('{first\_date}', '{last\_date}', '{session['username']}');"""

print(query)

try:

data = execute\_select\_query('farm\_client', 'client', query)

return render\_template('payments.html', data=data, username=session['username'])

except Exception as identifier:

pass

return render\_template('payments.html', username=session['username'])

@app.route('/client/check\_places/<username>/', methods=('POST', 'GET'))

def check\_places(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

city = request.form['city']

query = f"""SELECT subdivision\_id, addres, chief\_first\_name || ' ' || chief\_last\_name as chief from subdivision

where city = '{city}'"""

try:

data = execute\_select\_query('farm\_client', 'client', query)

return render\_template('city.html', username=session['username'], data=data)

except Exception as identifier:

pass

return render\_template('city.html', username=session['username'])

@app.route('/client/check\_cost/<username>/', methods=('POST', 'GET'))

def check\_cost(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

product\_name = request.form['product\_name']

try:

query = f"""SELECT product\_price, quanity\_of\_produced from product

JOIN subdivision USING (subdivision\_id)

WHERE product\_name = '{product\_name}';"""

print(query)

data = execute\_select\_query('farm\_client', 'client', query)

return render\_template('cost.html', data=data, username=session['username'])

except Exception as identifier:

pass

return render\_template('cost.html', username=session['username'])

# -------------------------------------------------------------------

# --------------------------STAFF PAGES------------------------------

@app.route('/admin/<username>')

def admin(username):

if 'username' not in session or session['username'] != username:

abort(401)

session\_ = loadSession('farm\_staff:staff')

data1 = session\_.execute(f"SELECT \* from staff WHERE login='{username}';")

data1 = data1.first()

print(data1)

return render\_template('Admin.html', dirstaff=data1, username=session['username'])

@app.route('/admin/addclient/<username>', methods=['POST', 'GET'])

def adminaddnewclient(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

name = request.form['name']

adress = request.form['adress']

surname = request.form['surname']

phone = request.form['phone']

login\_ = request.form['login\_']

password = request.form['password']

try:

query = f"SELECT \* FROM insert\_new\_client('{surname}', '{name}', '{phone}', '{adress}', '{login\_}', '{password}' );"

execute\_query('farm\_staff', 'staff', query)

return redirect(url\_for('admin', username=session['username']))

except Exception as e:

return render\_template("Add\_Client.html", username=session['username'])

return render\_template("Add\_Client.html", username=session['username'])

@app.route('/admin/updateclient/<username>', methods=['POST', 'GET'])

def adminupdateclient(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

name = request.form['name']

surname = request.form['surname']

phone = request.form['phone']

adres = request.form['adres']

try:

query = f"SELECT \* FROM update\_client('{name}', '{surname}', '{phone}', '{adres}');"

execute\_query('farm\_staff', 'staff', query)

return redirect(url\_for('admin', username=session['username']))

except Exception as e:

return render\_template("updateclient.html", username=session['username'])

return render\_template("updateclient.html", username=session['username'])

@app.route('/admin/checkproduct/<username>', methods=['POST', 'GET'])

def admincheckproduct(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

product\_name = request.form['product\_name']

try:

query = f"select \* from check\_product('{product\_name}');"

print(query)

data = execute\_select\_query('farm\_staff', 'staff', query)

return render\_template("checkproduct.html", username=session['username'], data=data)

except Exception as e:

return render\_template("checkproduct.html", username=session['username'])

return render\_template("checkproduct.html", username=session['username'])

@app.route('/admin/checkclient/<username>', methods=['POST', 'GET'])

def admincheckclient(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

client\_name = request.form['client\_name']

client\_sur = request.form['client\_sur']

try:

query = f"SELECT \* from client\_info WHERE last\_name = '{client\_sur}' and first\_name = '{client\_name}';"

data = execute\_select\_query('farm\_staff', 'staff', query)

print(data)

return render\_template("checkclient.html", username=session['username'], data=data)

except Exception as e:

return render\_template("checkclient.html", username=session['username'])

return render\_template("checkclient.html", username=session['username'])

@app.route('/admin/sold\_product/<username>', methods=['POST', 'GET'])

def adminsold\_product(username):

if 'username' not in session or session['username'] != username:

abort(401)

if request.method == 'POST':

date\_beg = request.form['date\_beg']

date\_end = request.form['date\_end']

try:

query = f"select \* from sold\_product('{date\_beg}', '{date\_end}');"

print(query)

data = execute\_select\_query('farm\_staff', 'staff', query)

return render\_template("sold\_product.html", username=session['username'], data=data)

except Exception as e:

return render\_template("sold\_product.html", username=session['username'])

return render\_template("sold\_product.html", username=session['username'])

# -------------------------------------------------------------------

# --------------------------BASE PAGES-------------------------------

@app.route('/home')

@app.route('/')

def home():

return render\_template('farm\_main.html')

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)