**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 система сільгосп підприємства**»

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | 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 «Сельхоз предприятие».

Цель работы – разработка информационной системы для «Сельхоз предприятия». Данная система призвана упростить работу сотрудников и руководителя предприятия. Так же данное приложение позволяет клиентам предприятия взаимодействовать с данной системой (оформление заказа, просмотр покупок за период времени, поиск пунктов выдачи в заданном городе). Данное приложение существует для упрощенной работы сельхоз предприятия. Упрощение заключается в контроле количества произведённой и проданной продукции

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

Сельхоз предприятие – система, в которой происходит проведение услуг, предоставленных самим предприятием. Целевой аудиторией данного проекта являются как продавцы маленьких молочных отделов / рынка. Так и клиенты которые предпочитают продукцию конкретного предприятия (оптовые закупки).

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 сможет оптимизировать и автоматизировать рабочий процесс. Для достижения указанной цели в курсовой работе необходимо решить следующие задачи:

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

# 1 ПОСТАНОВКА ЗАДАЧИ

Когда клиент хочет записаться в сельхоз предприятии, ему нужно связаться с сотрудником, оставить свои персональные данные, если он ещё не был клиентом данной организации, после чего получить логин и пароль для аккаунта клиента. Регистрацией клиента занимается сотрудник. Для того, чтобы клиент числился в организации его персональные данные должны занести в базу данных. Далее, клиент уже может отслеживать собственные покупки, пункты выдачи и собственно делать заказы. 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, создавать и удалять отделения;
2. Сотрудник – имеет возможность зарегистрировать клиента в базу данных, сделать заказ ресурсов;
3. Клиент – имеет возможность на просмотр информации по своим покупкам, оформления заказа;
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.

Полный список задач, записанных в формате пользовательских историй, указан 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).

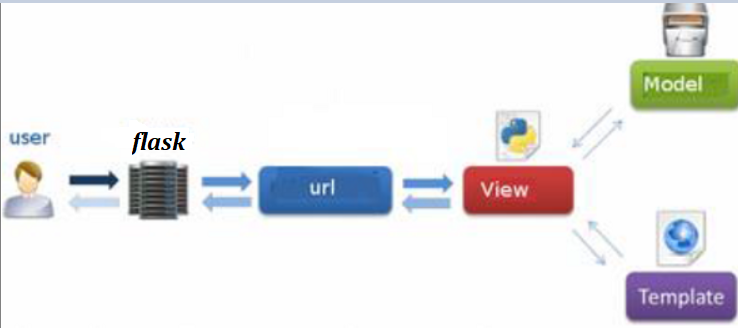


Рисунок 2.1 – Диаграмма работы паттерна MVT

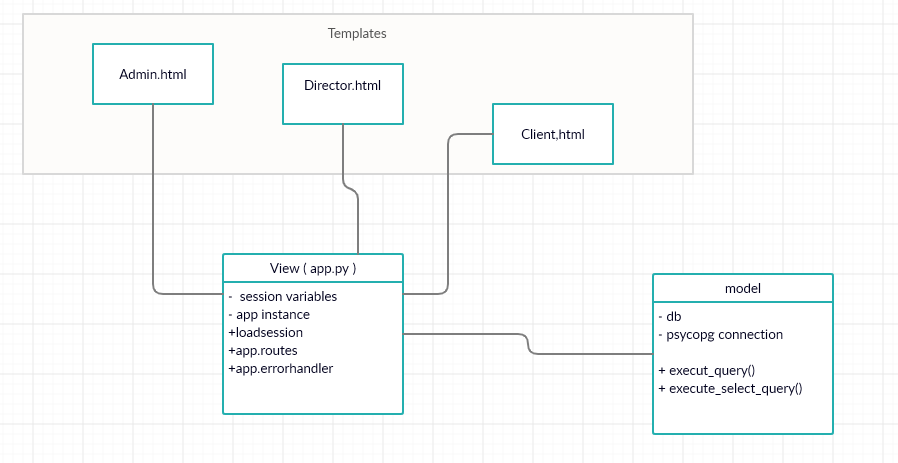


Рисунок 2.2 - Диаграмма классов проекта курсовой работы

MVС и MVT- очень похожи друг на друга, но их применение зависит от условий использования. Причиной выбора паттерна проектирования MVT являются следующие особенности процесса разработки на микропроцессоре Flask:

1. Наличие удобного шаблонизатора;

2. Возможность кастомизации на низком уровне работы продукта.

# ИНФОРМАЦИОННОЕ МОДЕЛИРОВАНИЕ

Следующий 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. заказ: дата, номер клиента;
2. клиент: имя клиента, фамилия клиента, мобильный телефон клиента, логин клиента, пароль клиента;
3. продукт: тип продукта, название продукта, цена продукта, ыид продукта, тип продукта, номер отделения, дата продукта;
4. сотрудник: имя сотрудника, фамилия сотрудника, мобильный телефон сотрудника, должность сотрудника, зарплата сотрудника, кол-во произведенного товара, адрес, роль, логин, пароль, номер отделения;
5. отделение: имя шефа, фамилия шефа, город, адрес, кол-во произведенного товара;
6. ресурсы: номер отделения, тип ресурса, название ресурса, кол-во ресурса;
7. поставка: номер продукта, цена продукта, номер заказа, кол-во товара.

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 «Сельхоз производство».

|  |  |  |
| --- | --- | --- |
| Имя атрибута | Назначение атрибута | Ограничения целостности |
| Profession | | |
| Profession\_id | Номер профессии | Первичный ключ |
| Name\_of\_prof | Название профессии | Не пустое |
| Staff | | |
| Staff\_id | Id сотрудника | Первичный ключ |
| Last\_name | Фамилия | Не пустое |
| First\_name | Имя | Не пустое |
| Phone\_number | Телефон | Не пустое |
| Work\_role | Должность | Не пустое |
| Salary | Зарплата | Не пустое |
| Quantity\_of\_products\_produced | Кол-во сделанного товара | Не пустое |
| Subdivision\_id | Номер отделения | Внешний ключ для связи с таблицей subdivision(subdivision\_id) |
| Address | Адрес | Не пустое |
| Cheese\_equipment | Работал ли на сыре | Не пустое |
| Milk\_equipment | Работал ли на молоке | Не пустое |
| Login | Логин | Не пустое, уникальное |
| Passw | Пароль | Не пустое |
| Role\_ | Роль в бд | Не пустое |
| Subdivision | | |
| Subdivision\_id | Номер отделения | Первичный ключ |
| Chief\_first\_name | Фамилия шефа | Не пустое |
| Chief\_last\_name | Имя шефа | Не пустое |
| City | Город | Не пустое |
| Quantity\_of\_produced | Кол-во сделанного товара | Не пустое |
| addres | Адрес | Не пустое |
| Order\_resource | | |
| Order\_resource\_id | Id заказа на ресурсы | Первичный ключ |
| Resource\_type | Тип ресурсов | Не пустое, значение из списка ('корм', 'медикаменты', 'топливо') |
| Resource\_price | Цена ресурса | Не пустое |
| Resource\_count | Кол-во ресурсов | Не пустое |
| Subdivision\_id | Номер отделения | Внешний ключ для связи с таблицей subdivision(subdivision\_id) |
| Order\_ | | |
| Order\_id | Номер заказа | Primary key |
| Date\_ | Дата заказа |  |
| Client\_id | Id клиента | Внешний ключ для связи с таблицей client |
| Supply | | |
| Order\_id | Id заказа | Внешний ключ для связи с таблицей order(order\_id) |
| Product\_id | Id продукции | Внешний ключ для связи с таблицей product(product\_id) |
| Product\_price | Цена продукции | Не пустое |
| Product\_count | Кол-во продукции | Не пустое |
|  |  | Уникальная комбинация  (order\_id, product\_id) |
| Product | | |
| Product\_id | Id продукции | Первичный ключ |
| Product\_kind | Вид продукции | Не пустое, значение из списка ('Стерилизованное','Не стерилизованное','Твердый','Мягкий','Плавленный') |
| Product\_name | Название продукции | Не пустое |
| Product\_price | Цена продукции | Не пустое |
| Product\_type | Тип продукции | Не пустое, значение из списка ('Молоко','Сыр') |
| Subdivision\_id | Номер отделения | Внешний ключ для связи с таблицей subdivision(subdivision\_id) |
| Description | Описание | Не пустое |
| Product\_date | Дата продукции | Не пустое |
| Required\_feed | Необходимый корм | Не пустое |
| Required\_fuel | Необходимое топливо | Не пустое |
| Required\_med | Необходимое участие | Не пустое |
| Client | | |
| Client\_id | Id клиента | Первичный ключ |
| Last\_name | Фамилия клиента | Не пустое |
| First\_name | Имя клиента | Не пустое |
| Mobile\_number | Телефон | Не пустое |
| Address\_of\_client | Адрес клиента | Не пустое |
| Login | Логин | Не пустое, уникально |
| passw | пароль | 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 заказ -товар, формализуется по правилу «многие ко многим»;
  2. Связи клиент - заказ формализуются по правилу «один ко многим»;
  3. Связь отделение – сотрудник так же является связью «один к многим»
  4. Связь отделение - заказ ресурсов также является связью «один к многим»

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

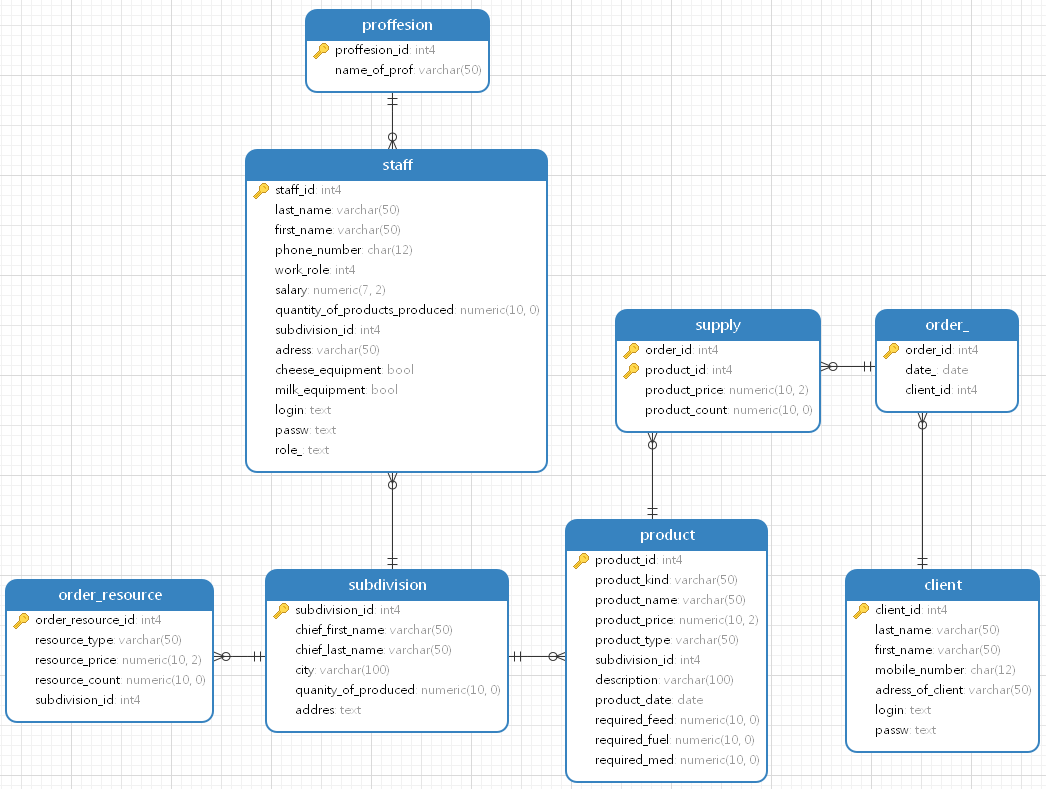


Рисунок 3.1 – ER-диаграмма базы данных фитнес клуба

# 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 использовалась также следующие обертка API postgresql: psycopg2 – библиотека предоставляющая возможность языку python взаимодействовать с API postgresql и исполнять запросы.

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 SELECT 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.

Таблица 6.1 – Привилегии ролей на таблицы БД

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Таблица\Роль | Директор | Сотрудник | Клиент | Гость |
| Profession | CRUD | R |  |  |
| Staff | CRUD | R |  | R |
| Subdivision | CRUD | R | R |  |
| Product | CRUD | CRUD |  |  |
| Order\_ | CRUD | CRUD | R |  |
| Client | CRUD | CRUD | R | R |
| Supply | CRUD | CRUD | R |  |
| Order\_resource | CRUD | CRUD | R |  |

# 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 – в данном проекте вся бизнес логика была вынесена на уровень баз данных. Подключение к бд при помощи ОРМ и row connection string реализовано в скрипте model.py.
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.

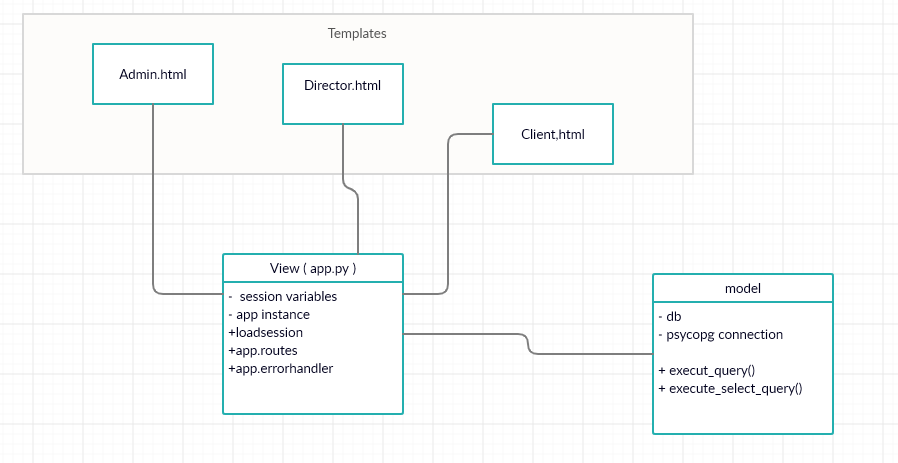


Рисунок 7.1 - Диаграмма классов информационной системы производство молочных продуктов

# 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 пользовательских задач, полный список задач которых находится в приложении А.

Для решения задачи А1 был использован запрос

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}');

Для решения задачи А2 был использован запрос

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};

Для решения задачи А3 был использован запрос

DELETE from staff where staff\_id = {number};

Для решения задачи А4 была написана хранимая процедура check\_product

обращение к которой является следующим запросом

select \* from check\_product('{name}');

Для решения задачи А5 была написана хранимая процедура sold\_product

обращение к которой является следующим запросом

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

Для решения задачи А6 был использован запрос

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};

Для решения задачи А7 был использован запрос

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;

Для решения задачи А8 был использован запрос

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

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

Для решения задачи А9 был использован запрос

UPDATE subdivision

SET

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

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

city = '{city}',

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

addres = '{adres}'

WHERE subdivision\_id = {sub\_id};

Для решения задачи А10 был использован запрос

delete from subdivision where subdivision\_id = {sub\_id};

Для решения задачи А11 был использован запрос

UPDATE staff

set salary = {salary}

WHERE staff\_id = {number};

Для решения задачи Б1 была написана хранимая процедура insert\_new\_client

обращение к которой является следующим запросом

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

Для решения задачи Б2 была написана хранимая процедура update\_client обращение к которой является следующим запросом

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

Для решения задачи Б3 была написана хранимая процедура check\_product обращение к которой является следующим запросом

select \* from check\_product('{product\_name}');

Для решения задачи Б4 была написано представление client\_info обращение к которой является следующим запросом

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

Для решения задачи Б5 была написана хранимая процедура sold\_product обращение к которой является следующим запросом

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

Для решения задачи В1 была написана хранимая процедура client\_bought\_product обращение к которой является следующим запросом

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

Для решения задачи В2 был использован запрос

SELECT subdivision\_id, addres, chief\_first\_name || ' ' || chief\_last\_name as chief from subdivision

where city = '{city}'

Для решения задачи В3 был использован запрос

SELECT product\_price, quanity\_of\_produced from product

JOIN subdivision USING (subdivision\_id)

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

Для решения задачи В4 была написана хранимая процедура client\_order обращение к которой является следующим запросом

9 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

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

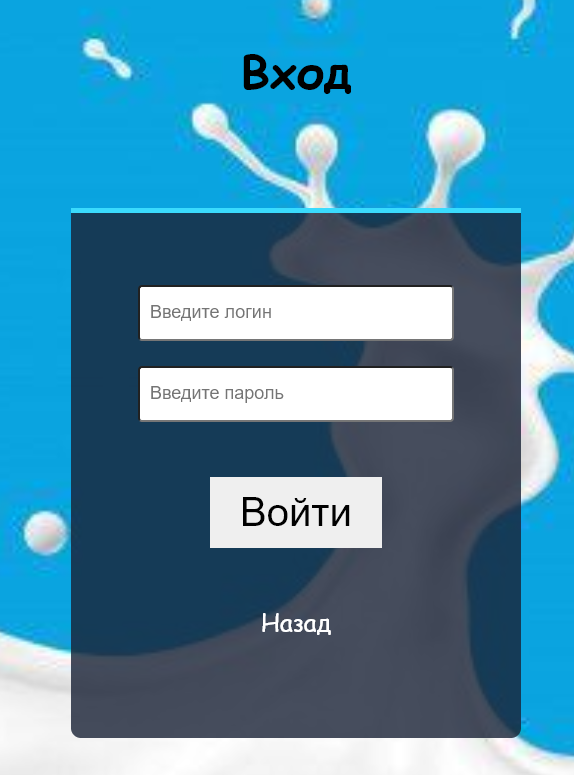


Рисунок 9.1 – Страница входа

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 регистрировать новых сотрудников системы. Для регистрации нового пользователя достаточно ввести его данные и нажать на кнопку подтверждения. Регистрацию клиентов же проводят сотрудники.



Рисунок 9.2 – Страница директора

Окно регистрации отображено на рис. 9.3.

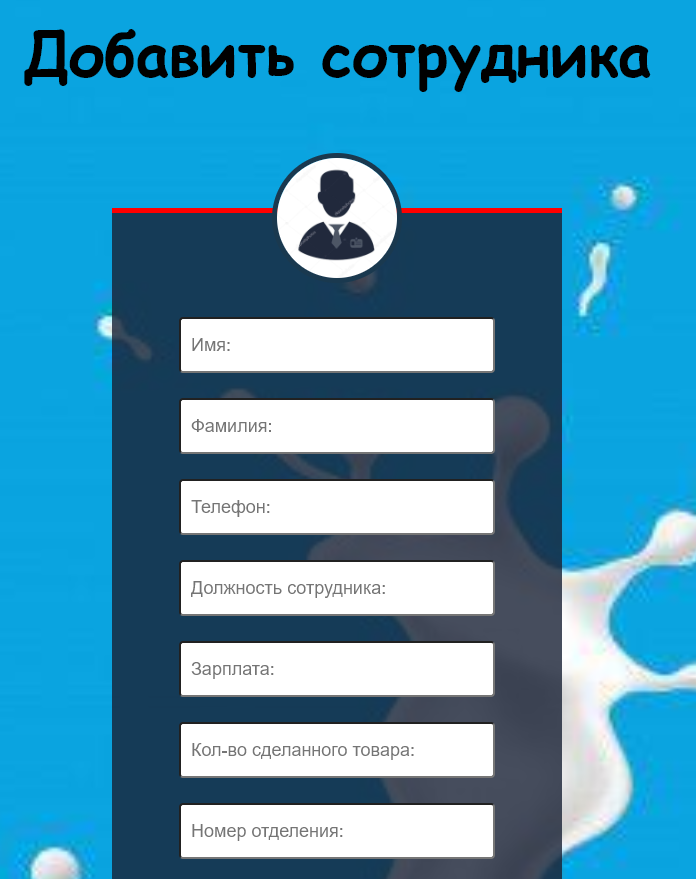


Рисунок 9.3 – Регистрация сотрудника

Перейдем к одной из основных задач производства – добавления клиента.

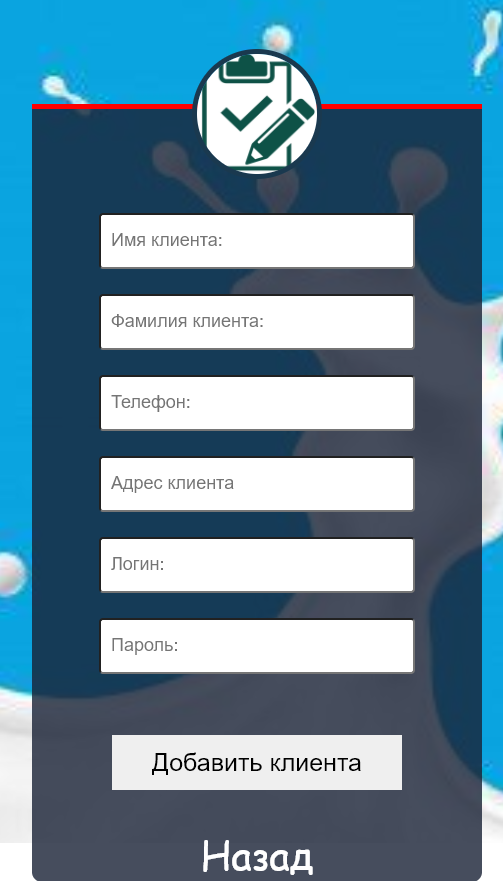


Рисунок 9.4 - Добавление клиента

Когда администратор хочет получить данные о клиенте он может нажать кнопку «Получить данные о клиенте». Интерфейс окна которого предоставлен на рис. 9.5.

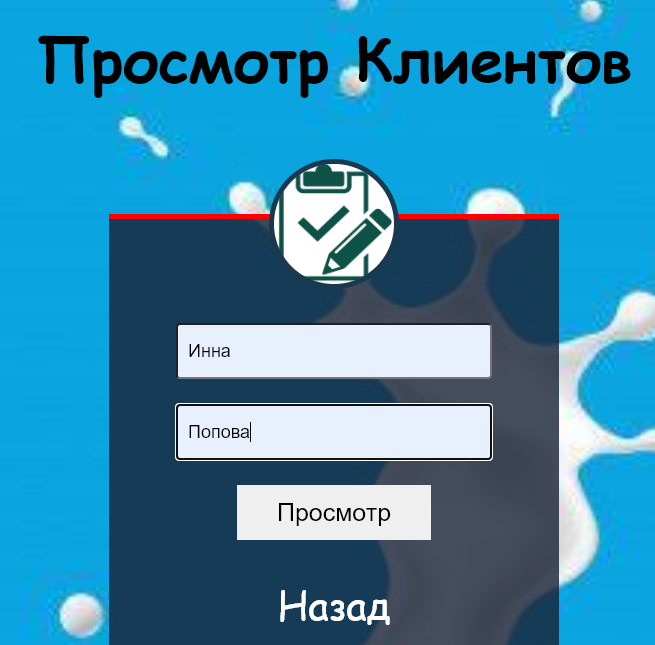


Рисунок 9.5 – Данные о клиенте

ЗАКЛЮЧЕНИЕ

В результате анализа предметной области, проведенного при выполнении курсовой работы, сформулированы цели создания информационной системы сельхоз предприятия, определен перечень её пользователей (директор, сотрудник, клиент, гость) и задач, которые они решают в рассматриваемой предметной области, в виде пользовательских историй. Определены требования к хранимым данным и спроектирована база данных из 8 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 не 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

**Задачи Пользователей**

Директор

A1) Добавить персонал в базу:

Вход: Вход: ФИО, должность, пол, дата рождения, зарплата, рабочее

место, начальник (если есть)

Выход: номер сотрудника

А2) Редактирование персонала в базе

Вход: ФИО, должность, пол, дата рождения, зарплата, рабочее место,

начальник (если есть)

Выход: номер сотрудника

А3) Удаление персонала

Вход: ФИО или номер сотрудника

А4) Просмотр наличия продукции

Вход: номер клиента и/или ФИО клиента

Выход: номер клиента, ФИО, адрес, тип заказываемой продукции

А5) Просмотреть расходы продукции за период времени

Вход: номер продукта и/или наименование продукта

Выход: номер продукта, наименование продукта, тип продукта, количество продукта

А6) Просмотреть активность отдельного сотрудника за период времени

Вход: номер сотрудника

Выход: номер сотрудника, ФИО, количество продукции, начальник (если есть)

А7) Просмотреть активность подразделение за период времени

Вход: адрес

Выход: начальник отделения, тип продукции, количество проданного и произведенного товара

А8) Добавление подразделения

Вход: Адрес, начальник

А9) Редактирование подразделения

Вход: номер подразделения

А10) Удаление подразделения

Вход: номер отдела

А11) Редактирование зарплаты персоналу

Вход: номер сотрудника и/или ФИО сотрудника, зарплата

Сотрудник

Б1) Добавление клиента в базу

Вход: ФИО клиента, адрес клиента

Выход: порядковый номер клиента

Б2) Редактирование клиента в базе

Вход: ФИО клиента

Б3) Просмотр наличия количества товара

Вход: номер товара и/или наименование товара

Выход: номер товара, наименование товара, тип товара, номер отдела,

который занимается производством товара, стоимость товара

Б4) Просмотр клиентов

Вход: номер клиента и/или ФИО клиента

Выход: номер клиента, ФИО клиента, адрес клиента, продукция

Б5) Просмотреть количество произведенной и проданной продукции за период времени

Вход: дата начала периода, дата окончания периода

Выход: тип товара, наименование товара, количество произведенного товара, количество проданного товара ,номер отдела который занимается производством этого товара, стоимость товара

Клиент

В1) Просмотр истории покупок за период времени

Вход: дата начала периода, дата конца периода.

Выход: номер товара, наименование товара, тип товара, номер отдела который занимался этим товаром, стоимость товара, количество товара

В2) Просмотр отделов в своём городе

Вход: Город

Выход: номер отделения, Адрес, директор

В3) Просмотр стоимости отдельного товара

Вход: наименование товара

Выход: цена, количество в наличии

В4) Добавление заказа продукции

Вход: тип продукции, наименование продукции, количество продукции, способ оплаты

Выход: номер заказа, стоимость продукции

# 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

# ЗАПРОСЫ НА СОЗДАНИЕ ХРАНИМЫХ ПРОЦЕДУР

Процедура для вставки клиента

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;

Процедура для редактирования 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;

Процедура для просмотра продукции

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;

Процедура для просмотра проданного продукта

CREATE OR REPLACE FUNCTION sold\_product(begin\_date date, end\_date date)

RETURNS TABLE

(product\_id int, product\_name varchar, product\_type varchar, subdivision\_id int, product\_price numeric,

quanity\_of\_produced numeric, sold\_amount numeric)

AS $$

SELECT product\_id, product\_name, product\_type, subdivision\_id, product.product\_price,

quanity\_of\_produced, sum(product\_count) as sold\_amount

from product

JOIN supply USING(product\_id)

JOIN order\_ USING(order\_id)

JOIN subdivision USING(subdivision\_id)

WHERE product\_date between begin\_date and end\_date

GROUP by 1, 2, 3, 4, 5, 6

Процедура для просмотра клиентом истории покупок

CREATE OR REPLACE FUNCTION client\_bought\_product(begin\_date date, end\_date date, c\_login varchar)

RETURNS TABLE

(product\_id int, product\_name varchar, product\_type varchar, subdivision\_id int, product\_price numeric,

product\_count numeric)

AS $$

SELECT product.product\_id, product.product\_name, product.product\_type, subdivision.subdivision\_id, product.product\_price, supply.product\_count

from client

JOIN order\_ USING(client\_id)

JOIN supply USING(order\_id)

JOIN product USING(product\_id)

JOIN subdivision USING(subdivision\_id)

WHERE client.first\_name=(select first\_name from client where login=c\_login) and client.last\_name=(select last\_name from client where login=c\_login)

and order\_.date\_ between begin\_date and end\_date;

$$ LANGUAGE sql;

# 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 USAGE, SELECT ON ALL SEQUENCES IN SCHEMA public TO farm\_staff;

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(login, passw, role\_) ON TABLE staff

to farm\_guest;

grant select on client\_info to farm\_staff;

grant SELECT(client\_id, login, passw) 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,

public.order\_resource

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 model import \*

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)

**model.py**

from sqlalchemy import create\_engine, MetaData

import psycopg2

from flask\_sqlalchemy import SQLAlchemy

from sqlalchemy.orm import scoped\_session, sessionmaker

db = SQLAlchemy()

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