**Министерство образования Рязанской области**

**Областное государственное бюджетное профессиональное образовательное учреждение**

**«Касимовский нефтегазовый колледж»**

**Отчет по учебной практике**

**УП ПМ 02**

**«Осуществление интеграции программных модулей»**

|  |  |
| --- | --- |
| Руководитель | |
| Аначно С.Н | |
| (фамилия, инициалы) | |
|  | |
| (подпись) (дата) | |
| Студент | 23И-1 |
| (группа) | |
| Баранов А.А | |
| (фамилия, инициалы) | |
|  | |
| (подпись) (дата) | |

Касимов 2025

**Областное государственное бюджетное профессиональное**

**образовательное учреждение**

**“Касимовский нефтегазовый колледж”**

**ДНЕВНИК**

**По практическому обучению**

|  |
| --- |
| **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_Учебная практика ПМ. 02\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |
|  |
| **«Осуществление интеграции программных модулей»** |

Специальность 09.02.07 «Информационные системы и программирование»

Студент Баранов Артём Александрович

Группа 23И-1

|  |  |  |
| --- | --- | --- |
| **Дата** | **Вид деятельности** | **Подпись студента** |
|  | Инструктаж по технике безопасности. |  |
|  | Анализ предметной области. Разработка и оформление технического задания. |  |
|  | Построение архитектуры программного средства. |  |
|  | Построение диаграмм вариантов использования и диаграммы последовательности. |  |
|  | Построение диаграмм компонентов  Построение диаграммы бизнес - процессов |  |
|  | Преобразование реляционной базы данных в модель сущность - связь |  |
|  | Проектирование реляционной базы данных |  |
|  | Проектирование реляционной базы данных |  |
|  | Сортировка и фильтрация данных. Поиск данных по одному или нескольким параметрам. Поиск в таблице. |  |
|  | Создание меню различных видов. |  |
|  | Создание форм и отчетов |  |
|  | Анализ современных САПР программного обеспечения |  |

Руководитель практики ОГБПОУ “КНГК”

Аначко С.Н

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_”\_\_\_”\_\_\_\_\_\_2024 г.

**Содержание**

1. [**Анализ предметной области** 3](#_Toc164685441)
2. [**Функциональная модель информационной системы** 4](#_Toc164685442)
3. [**Моделирование процессов информационной системы** 9](#_Toc164685443)
4. [**Логическая модели базы данных** 14](#_Toc164685444)
5. [**Физическая модель данных** 15](#_Toc164685445)
6. [**Репозиторий проекта на GitHub** 16](#_Toc164685446)
7. [**Реализация базы данных** 17](#_Toc164685448)
8. [**Реализация информационной системы** 26](#_Toc164685449)
9. Анализ предметной области

Анализ предметной области "Грузовые перевозки"

1. Введение

Грузовые перевозки – это ключевая отрасль логистики и экономики, обеспечивающая перемещение товаров между производителями, поставщиками и потребителями. Эффективность грузоперевозок влияет на себестоимость продукции, скорость доставки и общую конкурентоспособность бизнеса.

2. Основные виды грузовых перевозок

В зависимости от используемого транспорта выделяют:

- Автомобильные перевозки – наиболее гибкий и популярный способ, подходит для коротких и средних дистанций.

- Железнодорожные перевозки – экономичны для крупных партий грузов на большие расстояния.

- Морские перевозки – используются для международной торговли, перевозки крупногабаритных и массовых грузов.

- Авиаперевозки – самые быстрые, но дорогие, подходят для срочных и ценных грузов.

- Мультимодальные перевозки – комбинация нескольких видов транспорта для оптимизации доставки.

3. Участники рынка грузоперевозок

- Грузоотправители и грузополучатели\*\* (производители, торговые компании).

- Перевозчики (транспортные компании, частные автоперевозчики).

- Логистические операторы (3PL, 4PL-провайдеры).

- Таможенные и контролирующие органы (особенно в международных перевозках).

4. Ключевые технологии и тенденции

- Цифровизация: использование TMS (Transportation Management Systems), GPS-мониторинг, блокчейн для отслеживания грузов.

- Автоматизация: беспилотные грузовики, дроны, роботизированные склады.

- Экологизация: переход на электромобили, использование альтернативных видов топлива.

- Оптимизация маршрутов: применение ИИ для снижения затрат и времени доставки.

5. Проблемы и вызовы отрасли

- Высокая конкуренция среди перевозчиков.

- Рост цен на топливо и обслуживание транспорта.

- Нехватка квалифицированных кадров (водителей, логистов).

- Регуляторные ограничения (экологические нормы, таможенные барьеры).

6. Перспективы развития

- Развитие мультимодальных перевозок для снижения затрат.

- Внедрение беспилотного транспорта для сокращения издержек.

- Увеличение доли зеленой логистики под давлением экологических стандартов.

7. Заключение

Грузовые перевозки остаются важнейшим звеном в цепочке поставок. Развитие технологий и изменение рыночных условий требуют от участников отрасли гибкости, инвестиций в инновации и адаптации к новым требованиям.

Этот анализ можно дополнить конкретными примерами компаний, статистикой или углубленным изучением отдельных сегментов (например, международные перевозки или логистика e-commerce).

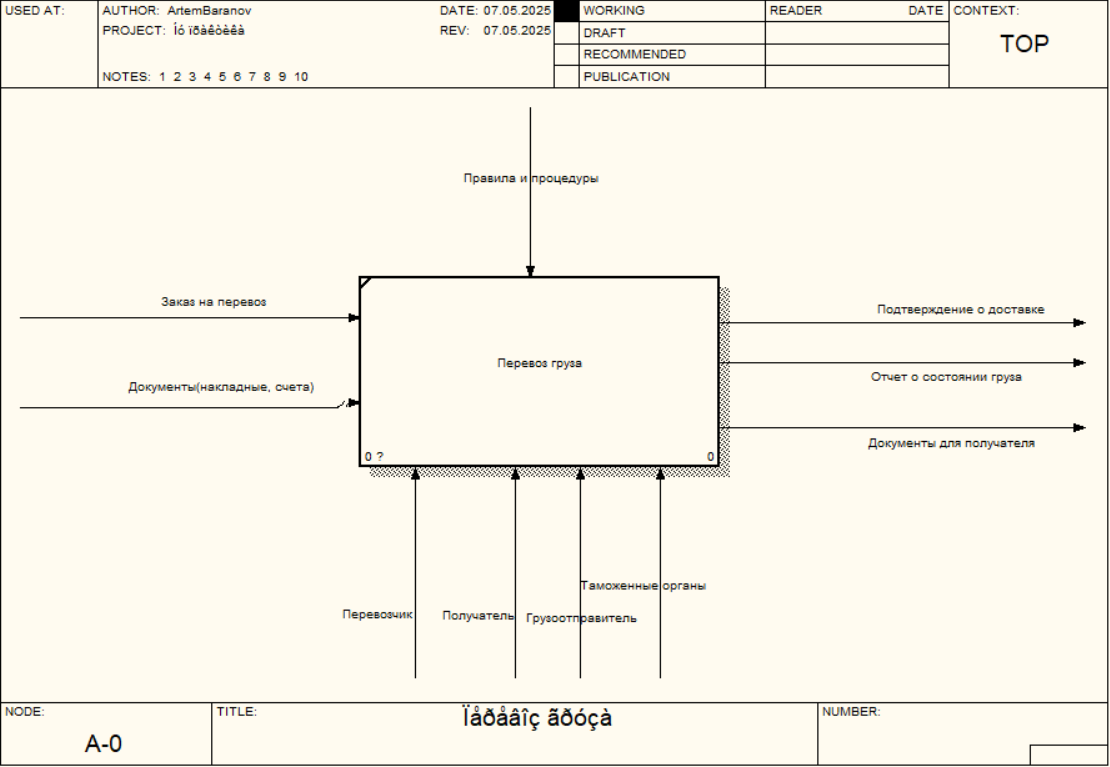


Рис. 1.1 Контекстная диаграмма «Перевоз груза»

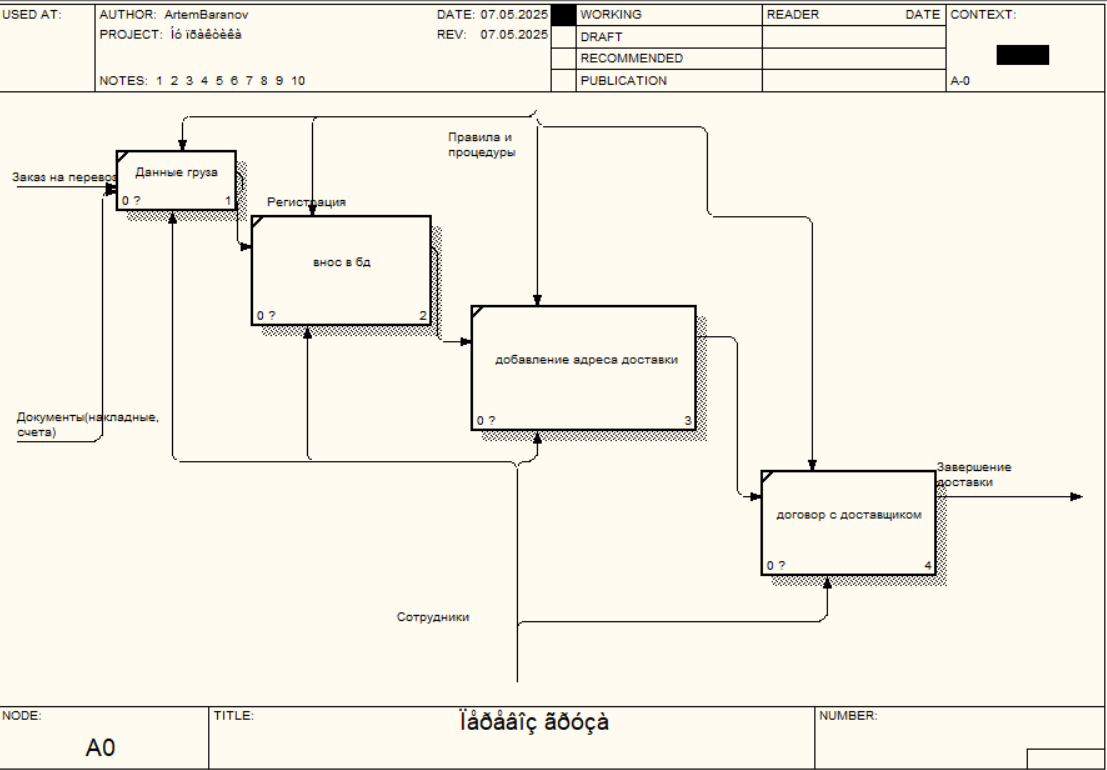


Рис 1.2 Диаграмма декомпозиции «Перевоз груза». Уровень А0.

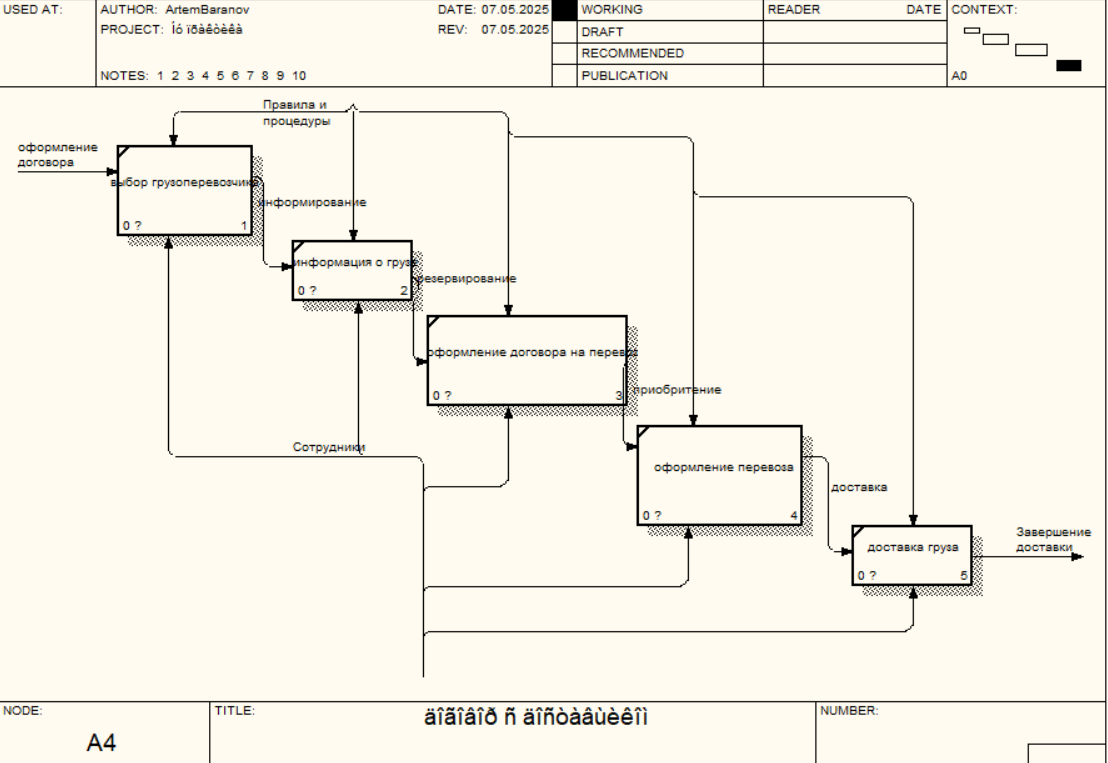


Рис 1.3 Диаграмма декомпозиции «Перевоз груза». Уровень А2.

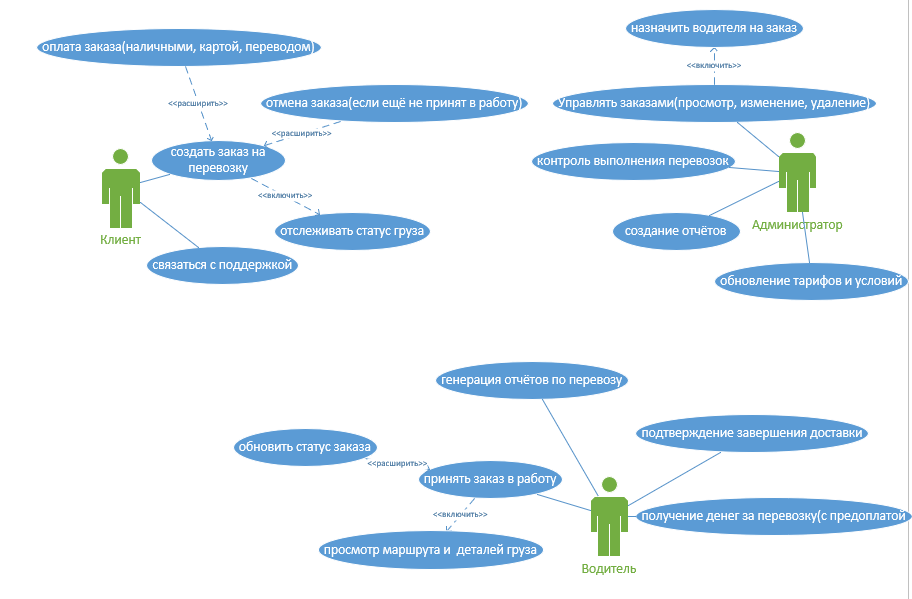


Рис 2 Диаграмма вариантов использования темы «Перевоз груза».

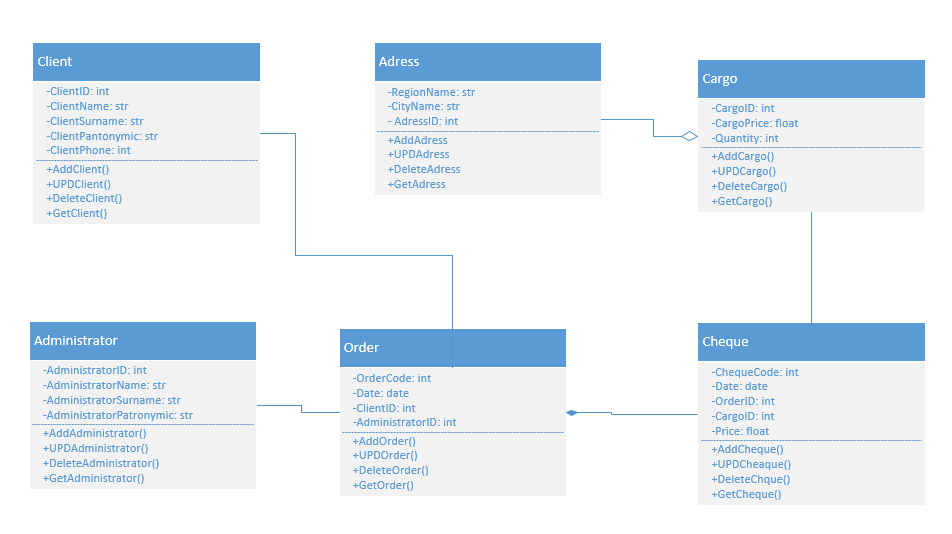


Рис 3 Диаграмма классов на тему «Перевоз груза».

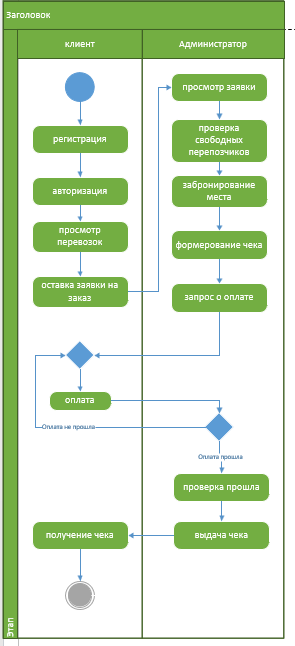


Рис 4 Диаграмма деятельности на тему «Перевоз груза».

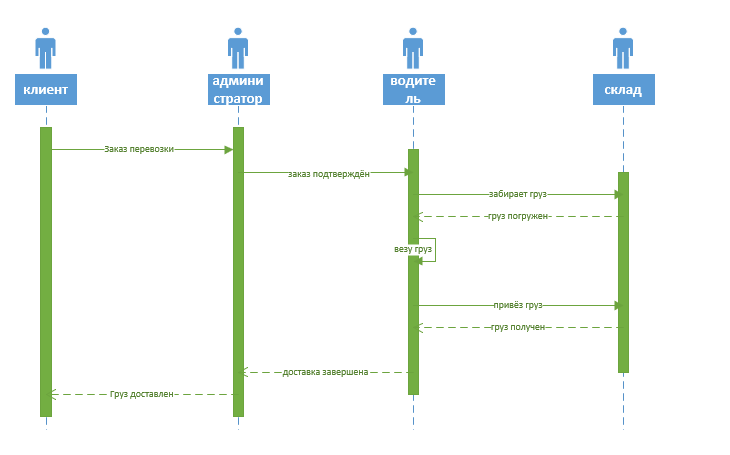


Рис 5 Диаграмма последовательности на тему «Перевоз груза».

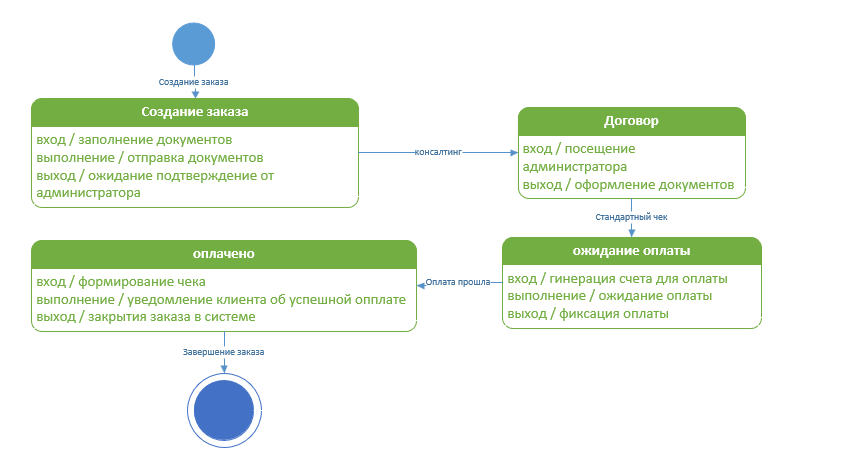


Рис 6 Диаграмма состояний на тему «Перевоз груза».

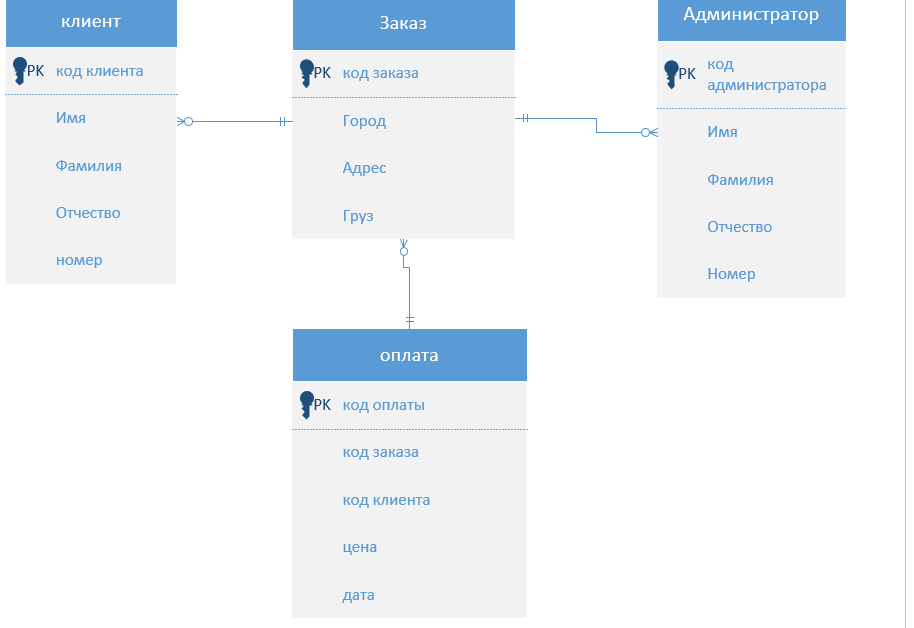


Рис 7 Логическая модель базы данных на тему «Перевоз груза»

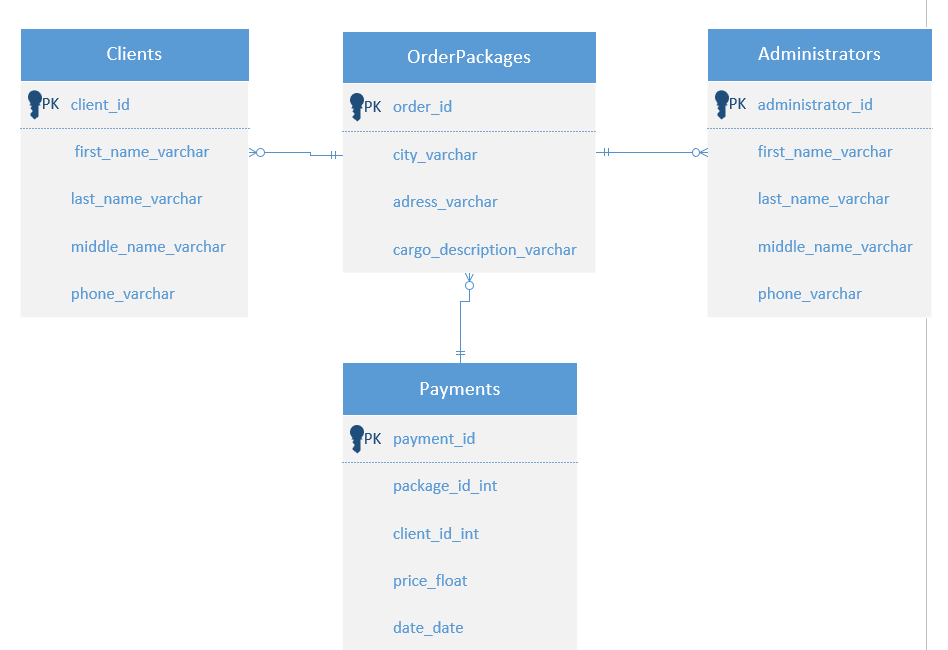


Рис 8 Физическая модель базы данных на тему «Перевоз груза».

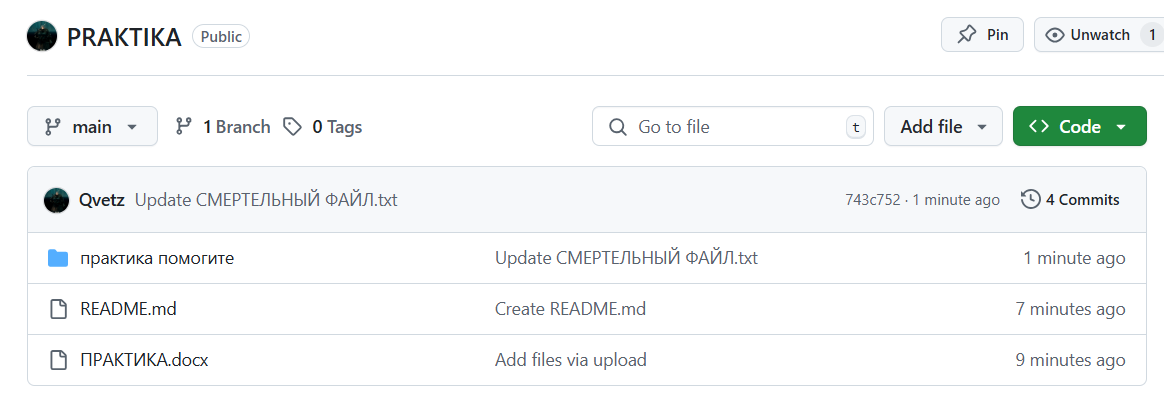


Рис 9 Репозиторий темы «Перевоз груза» на платформе GitHub.

Код программы:

config.py

DB\_NAME = "transportation.db"

database.py

import sqlite3

from config import DB\_NAME

def get\_connection():

    return sqlite3.connect(DB\_NAME)

def transportation\_db():

    conn = get\_connection()

    cursor = conn.cursor()

    # Города

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS City (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        name TEXT NOT NULL,

        region TEXT NOT NULL

    )

    ''')

    # Адреса

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS Adress (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        address TEXT,

        city\_id INTEGER,

        FOREIGN KEY (city\_id) REFERENCES cities(id)

    )

    ''')

    # Клиенты

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS clients (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        surname TEXT,

        name TEXT,

        patronymic TEXT,

        address TEXT,

        phone TEXT

    )

    ''')

    # Грузы

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS Gruz (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        duration INTEGER,

        base\_price REAL,

        adress\_id INTEGER,

        FOREIGN KEY (adress\_id) REFERENCES adresses(id)

    )

    ''')

    # Продажи

    cursor.execute('''

    CREATE TABLE IF NOT EXISTS sales (

        id INTEGER PRIMARY KEY AUTOINCREMENT,

        client\_id INTEGER,

        gruz\_id INTEGER,

        sale\_date DATE,

        discount REAL DEFAULT 0,

        total\_price REAL,

        departure\_date DATE,

        FOREIGN KEY (client\_id) REFERENCES clients(id),

        FOREIGN KEY (gruz\_id) REFERENCES gruzs(id)

    )

    ''')

    conn.commit()

    conn.close()

main.py

from Models.client import Client

from database import transportation\_db

from Models.sale import Sale

from Models.gruz import Gruz

from Models.city import City

from Models.adress import Adress

def show\_main\_menu():

    while True:

        print("\n Грузовые перевозки 'Фурри компани'")

        print("1. Работа с клиентами")

        print("2. Работа с грузами")

        print("3. Заказы")

        print("0. Выход")

        choice = input("Выберите действие: ")

        if choice == "1":

            client\_menu()

        elif choice == "2":

            gruz\_menu()

        elif choice == "3":

            sale\_menu()

        elif choice == "0":

            break

        else:

            print("Неверный выбор")

def client\_menu():

    while True:

        print("\n Работа с клиентами")

        print("1. Добавить клиента")

        print("2. Просмотреть клиентов")

        print("0. Назад")

        choice = input("Выберите действие: ")

        if choice == "1":

            surname = input("Фамилия: ")

            name = input("Имя: ")

            patronymic = input("Отчество: ")

            address = input("Адрес: ")

            phone = input("Телефон: ")

            client = Client(surname=surname, name=name, patronymic=patronymic, address=address, phone=phone)

            client.save()

            print("Клиент сохранен!")

        elif choice == "2":

            clients = Client.get\_all()

            print("\nСписок клиентов:")

            for c in clients:

                print(f"{c.id}. {c.surname} {c.name} {c.patronymic}")

        elif choice == "0":

            break

def gruz\_menu():

    while True:

        print("\n Работа с грузами")

        print("1. Добавить город")

        print("2. Добавить адрес")

        print("3. Добавить груз")

        print("4. Просмотреть грузы")

        print("0. Назад")

        choice = input("Выберите действие: ")

        if choice == "1":

            name = input("Название города: ")

            region = input("Регион: ")

            city = City(name=name, region=region)

            city.save()

            print("Город добавлен!")

        elif choice == "2":

            street\_address = input("Адрес: ")

            cities = City.get\_all()

            for c in cities:

                print(f"{c.id}. {c.name}")

            city\_id = int(input("ID города: "))

            address = Adress(address=street\_address, city\_id=city\_id)

            address.save()

            print("Адрес добавлен!")

        elif choice == "3":

            duration = int(input("Длительность (дни): "))

            price = float(input("Базовая цена: "))

            addresses = Adress.get\_all()

            for a in addresses:

                city = City.get\_by\_id(a.city\_id)

                if city:

                    print(f"{a.id}. {city.name}, {a.address}")

                else:

                    print(f"{a.id}. [НЕИЗВЕСТНЫЙ ГОРОД], {a.address}")

            address\_id = int(input("ID адреса: "))

            gruz = Gruz(duration=duration, base\_price=price, adress\_id=address\_id)

            gruz.save()

            print("Груз добавлен!")

        elif choice == "4":

            gruzs = Gruz.get\_all()

            for g in gruzs:

                address = Adress.get\_by\_id(g.adress\_id)

                city = City.get\_by\_id(address.city\_id)

                print(f"{g.id}. {city.name} - {address.address} ({g.duration} дней, {g.base\_price} руб.)")

        elif choice == "0":

            break

def sale\_menu():

    while True:

        print("\n Заказы")

        print("1. Оформить заказ")

        print("2. Просмотреть заказы")

        print("0. Назад")

        choice = input("Выберите действие: ")

        if choice == "1":

            # Выбор клиента

            clients = Client.get\_all()

            for c in clients:

                print(f"{c.id}. {c.surname} {c.name} {c.patronymic}")

            client\_id = int(input("ID клиента: "))

            # Выбор груза

            gruzs = Gruz.get\_all()

            for g in gruzs:

                address = Adress.get\_by\_id(g.adress\_id)

                city = City.get\_by\_id(address.city\_id)

                print(f"{g.id}. {city.name} - {address.address} ({g.duration} дней, {g.base\_price} руб.)")

            gruz\_id = int(input("ID груза: "))

            # Скидки

            print("Доступные скидки:")

            print("- 5% как за постоянного клиента")

            discount = float(input("Общий процент скидки: "))

            departure = input("Дата выезда (ГГГГ-ММ-ДД): ")

            sale = Sale(

                client\_id=client\_id,

                gruz\_id=gruz\_id,

                discount=discount,

                departure\_date=departure

            )

            sale.save()

            print(f"Заказ оформлен! Итоговая цена: {sale.total\_price} руб.")

        elif choice == "2":

            sales = Sale.get\_all()

            for s in sales:

                client = Client.get\_by\_id(s.client\_id)

                gruz = Gruz.get\_by\_id(s.gruz\_id)

                address = Adress.get\_by\_id(gruz.adress\_id)

                city = City.get\_by\_id(address.city\_id)

                print(f"{s.id}. {client.surname} {client.name} - {city.name} ({gruz.duration} дней), {s.total\_price} руб.")

        elif choice == "0":

            break

if \_\_name\_\_ == '\_\_main\_\_':

    transportation\_db()

    show\_main\_menu()

Папка Models:

adress.py

from database import get\_connection

from Models.city import City

import sqlite3

class Adress:

    def \_\_init\_\_(self, id=None, name=None, address=None, city\_id=None):

        self.id = id

        self.name = name

        self.address = address

        self.city\_id = city\_id

    def save(self):

        conn = get\_connection()

        cursor = conn.cursor()

        if self.id is None:

            cursor.execute("""

            INSERT INTO Adress (address, city\_id)

            VALUES (?, ?)

            """, (self.address, self.city\_id))

            self.id = cursor.lastrowid

        else:

            cursor.execute("""

            UPDATE Adress SET address=?, city=?

            WHERE id=?

            """, (self.address, self.city\_id, self.id))

        conn.commit()

        conn.close()

    def delete(self):

        if self.id:

            conn = get\_connection()

            cursor = conn.cursor()

            cursor.execute("DELETE FROM Adress WHERE id=?", (self.id,))

            conn.commit()

            conn.close()

    @staticmethod

    def get\_all():

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM Adress")

        rows = cursor.fetchall()

        conn.close()

        print("\n Адреса доставки:")

        return [Adress(\*row) for row in rows]

    @staticmethod

    def get\_by\_id(Adress\_id):

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM Adress WHERE id=?", (Adress\_id,))

        row = cursor.fetchone()

        conn.close()

        return Adress(\*row) if row else None

    @staticmethod

    def get\_by\_country(city):

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM Adress WHERE city=?", (city,))

        rows = cursor.fetchall()

        conn.close()

        return [Adress(\*row) for row in rows]

city.py

from database import get\_connection

class City:

    def \_\_init\_\_(self, id=None, name=None, region=None):

        self.id = id

        self.name = name

        self.region = region

    def save(self):

        conn = get\_connection()

        cursor = conn.cursor()

        if self.id is None:

            cursor.execute("""

            INSERT INTO City (name, region)

            VALUES (?, ?)

            """, (self.name, self.region))

            self.id = cursor.lastrowid

        else:

            cursor.execute("""

            UPDATE City SET name=?, region=?

            WHERE id=?

            """, (self.name, self.region, self.id))

        conn.commit()

        conn.close()

    def delete(self):

        if self.id:

            conn = get\_connection()

            cursor = conn.cursor()

            cursor.execute("DELETE FROM City WHERE id=?", (self.id,))

            conn.commit()

            conn.close()

    @staticmethod

    def get\_all():

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM City")

        rows = cursor.fetchall()

        conn.close()

        print("\n Список городов:")

        return [City(\*row) for row in rows]

    @staticmethod

    def get\_by\_id(city\_id):

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM City WHERE id=?", (city\_id,))

        row = cursor.fetchone()

        conn.close()

        return City(\*row) if row else None

client.py

from database import get\_connection

class Client:

    def \_\_init\_\_(self, id=None, surname=None, name=None, patronymic=None, address=None, phone=None):

        self.id = id

        self.surname = surname

        self.name = name

        self.patronymic = patronymic

        self.address = address

        self.phone = phone

    def save(self):

        conn = get\_connection()

        cursor = conn.cursor()

        if self.id is None:

            cursor.execute("""

            INSERT INTO clients (surname, name, patronymic, address, phone)

            VALUES (?, ?, ?, ?, ?)

            """, (self.surname, self.name, self.patronymic, self.address, self.phone))

            self.id = cursor.lastrowid

        else:

            cursor.execute("""

            UPDATE clients SET surname=?, name=?, patronymic=?, address=?, phone=?

            WHERE id=?

            """, (self.surname, self.name, self.patronymic, self.address, self.phone, self.id))

        conn.commit()

        conn.close()

    def delete(self):

        if self.id:

            conn = get\_connection()

            cursor = conn.cursor()

            cursor.execute("DELETE FROM clients WHERE id=?", (self.id,))

            conn.commit()

            conn.close()

    @staticmethod

    def get\_all():

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM clients")

        rows = cursor.fetchall()

        conn.close()

        return [Client(\*row) for row in rows]

    @staticmethod

    def get\_by\_id(client\_id):

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM clients WHERE id=?", (client\_id,))

        row = cursor.fetchone()

        conn.close()

        return Client(\*row) if row else None

gruz.py

from database import get\_connection

from Models.adress import Adress

class Gruz:

    def \_\_init\_\_(self, id=None, base\_price=0, duration=0, adress\_id=None):

        self.id = id

        self.base\_price = base\_price

        self.duration = duration

        self.adress\_id= adress\_id

    def save(self):

        conn = get\_connection()

        cursor = conn.cursor()

        if self.id is None:

            cursor.execute("""

            INSERT INTO Gruz (duration, base\_price, adress\_id)

            VALUES (?, ?, ?)

            """, (self.duration, self.base\_price, self.adress\_id))

            self.id = cursor.lastrowid

        else:

            cursor.execute("""

            UPDATE Gruz SET duration=?, base\_price=?, adress\_id=?

            WHERE id=?

            """, (self.duration, self.base\_price, self.adress\_id, self.id))

        conn.commit()

        conn.close()

    def delete(self):

        if self.id:

            conn = get\_connection()

            cursor = conn.cursor()

            cursor.execute("DELETE FROM Gruz WHERE id=?", (self.id,))

            conn.commit()

            conn.close()

    @staticmethod

    def get\_all():

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM Gruz")

        rows = cursor.fetchall()

        conn.close()

        print("\n Список грузов:")

        return [Gruz(\*row) for row in rows]

    @staticmethod

    def get\_by\_id(Gruz\_id):

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM Gruz WHERE id=?", (Gruz\_id,))

        row = cursor.fetchone()

        conn.close()

        return Gruz(\*row) if row else None

    @staticmethod

    def get\_by\_hotel(city):

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM Gruz WHERE city=?", (city,))

        rows = cursor.fetchall()

        conn.close()

        return [Gruz(\*row) for row in rows]

sale.py

from database import get\_connection

from Models.client import Client

from Models.city import City

from Models.adress import Adress

from Models.gruz import Gruz

from datetime import date

class Sale:

    def \_\_init\_\_(self, id=None, client\_id=None, Gruz\_id=None, sale\_date=None,

                 discount=0, total\_price=0, departure\_date=None):

        self.id = id

        self.client\_id = client\_id

        self.Gruz\_id = Gruz\_id

        self.sale\_date = sale\_date or date.today().strftime("%Y-%m-%d")

        self.discount = discount

        self.total\_price = total\_price

        self.departure\_date = departure\_date

    def save(self):

        conn = get\_connection()

        cursor = conn.cursor()

        # Рассчитываем цену с учетом скидок

        Gruz = Gruz.get\_by\_id(self.Gruz\_id)

        City = City.get\_by\_id(Adress.get\_by\_id(Gruz.Adress\_id).City\_id)

        # Базовая цена

        base\_price = Gruz.base\_price

        # Применяем скидки (суммируются)

        total\_discount = self.discount

        # Итоговая цена

        self.total\_price = base\_price \* (1 - total\_discount/100)

        if self.id is None:

            cursor.execute("""

            INSERT INTO sales (client\_id, Gruz\_id, sale\_date, discount, total\_price, departure\_date)

            VALUES (?, ?, ?, ?, ?, ?)

            """, (self.client\_id, self.Gruz\_id, self.sale\_date, self.discount,

                  self.total\_price, self.departure\_date))

            self.id = cursor.lastrowid

        else:

            cursor.execute("""

            UPDATE sales SET client\_id=?, Gruz\_id=?, sale\_date=?,

                            discount=?, total\_price=?, departure\_date=?

            WHERE id=?

            """, (self.client\_id, self.Gruz\_id, self.sale\_date, self.discount,

                  self.total\_price, self.departure\_date, self.id))

        conn.commit()

        conn.close()

    def delete(self):

        if self.id:

            conn = get\_connection()

            cursor = conn.cursor()

            cursor.execute("DELETE FROM sales WHERE id=?", (self.id,))

            conn.commit()

            conn.close()

    @staticmethod

    def get\_all():

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM sales")

        rows = cursor.fetchall()

        conn.close()

        print("\n Список заказов:")

        return [Sale(\*row) for row in rows]

    @staticmethod

    def get\_by\_id(sale\_id):

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM sales WHERE id=?", (sale\_id,))

        row = cursor.fetchone()

        conn.close()

        return Sale(\*row) if row else None

    @staticmethod

    def get\_by\_client(client\_id):

        conn = get\_connection()

        cursor = conn.cursor()

        cursor.execute("SELECT \* FROM sales WHERE client\_id=?", (client\_id,))

        rows = cursor.fetchall()

        conn.close()

        return [Sale(\*row) for row in rows]