

Санкт-Петербургский политехнический университет Петра Великого
Физико-механический институт

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 3
«Генерация диаграммы»
по дисциплине «Автоматизация научных исследований»

Выполнил
студент гр. № 5040102/50201

Степанов А.А.

Преподаватель: Новиков Ф.А.

Санкт-Петербург
2025 г.

ЗАДАНИЕ

1. Часть 1. Диаграмма вариантов использования

- Выбрать предметную область для проектирования информационной системы
- Определить основных действующих лиц системы
- Выявить основные функциональные требования
- Построить диаграмму вариантов использования с учетом следующих требований:
 1. Отобразить все действующие лица
 2. Определить основные и вспомогательные варианты использования
 3. Установить отношения между вариантами использования (include, extend)
 4. Добавить границу системы
 5. Обеспечить логическую целостность диаграммы

2. Часть 2. Диаграмма классов

- На основе выбранной предметной области разработать диаграмму классов
- Определить основные классы системы
- Установить иерархию наследования
- Определить типы связей между классами
- Указать атрибуты классов с типами данных
- Добавить кратности ассоциаций

Требования к выполнению

- Использовать PlantUML для генерации диаграмм
- Соблюдать стандарты UML
- Обеспечить логическую целостность диаграмм
- Предоставить комментарии к ключевым элементам диаграмм
- Проверить работоспособность сгенерированного кода
- Провести исследование для 1-2 моделей ИИ

Критерии оценки

- Корректность синтаксиса PlantUML
- Соответствие стандартам UML
- Полнота отображения функциональных требований
- Качество визуализации
- Логическая целостность диаграмм
- Обоснованность выбора предметной области

Форма отчёта

1. Исходный код PlantUML для обеих диаграмм
2. Сгенерированные диаграммы
3. Пояснительная записка с описанием:
 1. Выбранной предметной области
 2. Обоснования выбора действующих лиц
 3. Функциональных требований
 4. Структуры классов
4. Анализ результатов проектирования

В качестве нейросети был использован Qwen 3.5. Реализация кода PlantUML производилась в онлайн-сервисе: <https://editor.plantuml.com/>

Промт: Информационная система управления службой такси. Построить диаграмму вариантов использования с учетом следующих требований:

1. Отобразить все действующие лица
2. Определить основные и вспомогательные варианты использования
3. Установить отношения между вариантами использования (include, extend)
4. Добавить границу системы
5. Обеспечить логическую целостность диаграммы

Сделай

в

PlantUML

Вывод:

```
@startuml
left to right direction
skinparam usecase {
    BackgroundColor White
    BorderColor Black
    ArrowColor Black
}

rectangle "Служба такси" {
    usecase "Авторизоваться" as UC1
    usecase "Зарегистрироваться" as UC2
    usecase "Оформить заказ" as UC3
    usecase "Отменить заказ" as UC4
    usecase "Оценить заказ" as UC5
    usecase "Просмотреть историю заказов" as UC6
    usecase "Управлять профилем" as UC7
    usecase "Назначить водителя" as UC8
    usecase "Управлять справочниками" as UC9
    usecase "Проверить авторизацию" as UC10
    usecase "Рассчитать стоимость поездки" as UC11
    usecase "Уведомить пользователя" as UC12
}

actor "Клиент" as Client
actor "Водитель" as Driver
actor "Диспетчер" as Dispatcher
actor "Администратор" as Admin

Client --> UC1
Client --> UC2
Client --> UC3
Client --> UC4
Client --> UC5
Client --> UC6
Client --> UC7

Driver --> UC1
Driver --> UC6
Driver --> UC7
```

Dispatcher --> UC1

Dispatcher --> UC8

Dispatcher --> UC6

Admin --> UC1

Admin --> UC9

UC3 ..> UC10 : <<include>>

UC3 ..> UC11 : <<include>>

UC8 ..> UC12 : <<include>>

UC4 ..> UC3 : <<extend>>

UC5 ..> UC6 : <<extend>>

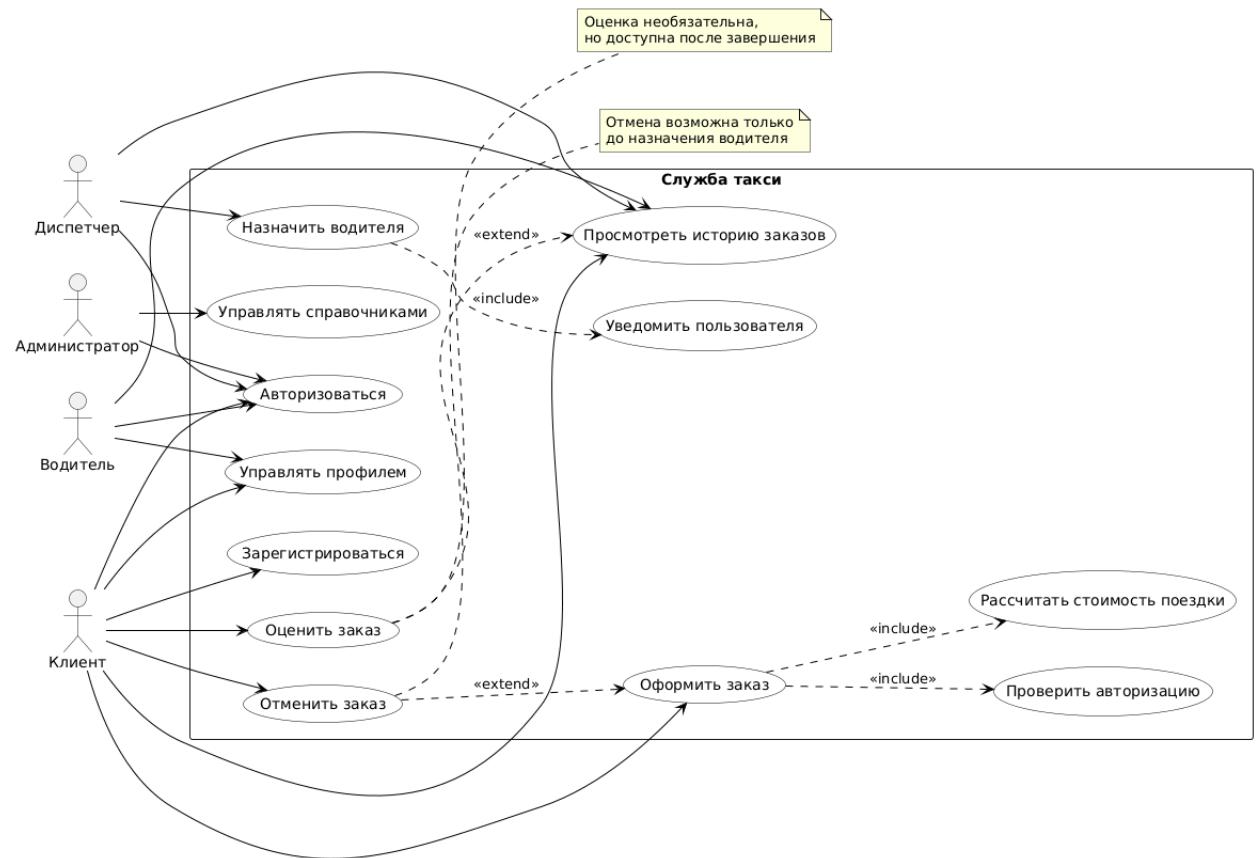
note right of UC4

Отмена возможна только
до назначения водителя
end note

note right of UC5

Оценка необязательна,
но доступна после завершения
end note

@enduml



Пояснение к диаграмме:

Действующие лица:

- Клиент
- Водитель
- Диспетчер

- Администратор

Основные варианты использования:

Оформить заказ, Отменить заказ, Оценить заказ, Назначить водителя, Управлять справочниками
Вспомогательные варианты использования:

Проверить авторизацию, Рассчитать стоимость поездки, Уведомить пользователя
Отношения:

<<include>> — обязательные подпроцессы (например, авторизация при оформлении заказа)

<<extend>> — условные/опциональные расширения (отмена — расширение заказа; оценка — расширение просмотра истории)

Граница системы: прямоугольник с названием «Служба такси»

Логическая целостность:

- Нет избыточных связей
- Каждый актор взаимодействует только с релевантными функциями
- Вспомогательные UC используются повторно, как положено по UML

ПРОМТ 2

Нужна пояснительная записка с описанием:

1. Выбранной предметной области
2. Обоснования выбора действующих лиц
3. Функциональных требований
4. Структуры классов

ВЫВОД:

Пояснительная записка к проектированию информационной системы «Служба такси»

1. Выбранная предметная область

Предметной областью проектирования выбрана **деятельность службы такси** — организация, обеспечивающая транспортные услуги по перевозке пассажиров по запросу.

Система направлена на автоматизацию ключевых бизнес-процессов: подача и обработка заказов, распределение водителей, расчёт стоимости, обратная связь и администрирование.

Актуальность выбора обусловлена широким распространением подобных систем, необходимостью повышения эффективности диспетчерской работы, а также ростом требований к прозрачности и удобству взаимодействия для конечных пользователей.

2. Обоснование выбора действующих лиц

В системе выделены следующие **акторы** (действующие лица), соответствующие реальным ролям в бизнес-процессе:

Актор	Обоснование
Клиент	Основной инициатор заказа на транспортные услуги. Требует возможность оформлять, отслеживать и оценивать заказы.
Водитель	Исполнитель услуги. Должен получать информацию о назначенных заказах и управлять своим профилем (статус доступности, данные авто).
Диспетчер	Координатор взаимодействия между клиентами и водителями. Отвечает за оперативное назначение водителей и контроль выполнения.
Администратор	Обеспечивает техническую и справочную поддержку системы: управление тарифами, типами автомобилей, статусами заказов, учётными записями.

Выбор акторов соответствует принципу **полноты охвата** функциональных ролей и исключает избыточные или дублирующие сущности.

3. Функциональные требования

Функциональные требования определены на основе анализа бизнес-процессов и ролей акторов. Ключевые требования включают:

Требование	Описание
Аутентификация и авторизация	Все пользователи должны проходить проверку подлинности перед доступом к функциям системы.
Регистрация клиентов	Новые клиенты могут создавать учётные записи с указанием

Требование	Описание
	контактных данных.
Оформление заказа	Клиент указывает точки отправления и назначения; система фиксирует заказ и инициирует его обработку.
Назначение водителя	Диспетчер (или автоматическая система) выбирает доступного водителя и назначает его на заказ.
Расчёт стоимости	Стоимость поездки определяется на основе тарифа, расстояния и дополнительных условий (например, время суток).
Отмена заказа	Клиент может отменить заказ до момента назначения водителя.
Оценка заказа	После завершения поездки клиент может оставить числовую оценку и комментарий.
Просмотр истории	Клиенты и водители могут просматривать список завершённых заказов.
Управление профилем	Пользователи могут редактировать личные данные (ФИО, телефон, данные автомобиля — для водителей).
Администрирование справочников	Администратор управляет тарифами, статусами заказов («ожидает», «в пути», «завершён»), типами автомобилей и др.
Уведомления	Система оповещает участников о ключевых событиях (назначение водителя, отмена, завершение).

Все требования направлены на обеспечение **полного жизненного цикла заказа и гибкости системы** при изменении бизнес-правил.

4. Структура классов (концептуальная модель)

Для отражения статической структуры системы определены следующие ключевые классы:

Класс	Атрибуты	Основные методы	Комментарий
User (абстрактный)	user_id, login, password_hash, full_name, phone	authenticate(), update_profile()	Базовый класс для всех пользователей.
Client	—	place_order(), cancel_order(), rate_ride()	Наследует User.
Driver	car_model, license_plate, is_available	accept_assignment(), set_availability()	Наследует User.
Dispatcher	—	assign_driver(), monitor_orders()	Наследует User.
Admin	—	manage_reference_data()	Наследует User.
Order	order_id, pickup, destination, status, client_id, driver_id, created_at, completed_at, fare	calculate_fare(), cancel(), complete()	Центральный класс бизнес-логики.
Tariff	tariff_id, name, base_price, price_per_km, is_active	apply_to_distance()	Используется при расчёте стоимости.
Notification	notification_id, recipient_id, message, sent_at, type	send()	Поддержка каналов уведомлений.
ReferenceData (абстрактный)	id, name, description	—	Базовый класс для справочников (тарифы, статусы и др.).

Ассоциации и связи:

- Client → 1.. Order
- Driver → 0.. Order
- Order — 1 → Tariff
- Dispatcher → 1.. Order (назначение)
- Order → 0..1 Rating (в случае оценки)

Модель обеспечивает **модульность, повторное использование** (через наследование User) и **простоту расширения** (например, добавление новых типов уведомлений или тарифов).

ПРОМТ: Сделай диаграмму классов в plantuml

Вывод:

```
@startuml
skinparam classAttributeIconSize 0
skinparam class {
    BackgroundColor White
    BorderColor Black
    ArrowColor Black
    AttributeFontSize 12
    MethodFontSize 12
}

' ===== Абстрактный базовый класс =====
abstract class User {
    + Integer user_id
    + String login
    + String password_hash
    + String full_name
    + String phone
    + {abstract} void update_profile()
    + Boolean authenticate(String password)
}

' ===== Конкретные роли =====
class Client {
    + void place_order(Location pickup, Location destination)
    + void cancel_order(Order order)
    + void rate_ride(Order order, Integer rating, String comment)
}

class Driver {
    + String car_model
    + String license_plate
    + Boolean is_available
    + void set_availability(Boolean status)
    + void accept_assignment(Order order)
}

class Dispatcher {
    + void assign_driver(Order order, Driver driver)
    + List<Order> monitor_orders()
}

class Admin {
    + void manage_reference_data(String entity, Map<String, Object> data)
}

' ===== Бизнес-классы =====
class Order {
    + Integer order_id
    + Location pickup
    + Location destination
    + String status
    + LocalDateTime created_at
    + LocalDateTime completed_at
    + BigDecimal fare
}
```

```

+ void calculate_fare(Tariff tariff, Double distance_km)
+ void cancel()
+ void complete()
}

class Tariff {
+ Integer tariff_id
+ String name
+ BigDecimal base_price
+ BigDecimal price_per_km
+ Boolean is_active
+ BigDecimal apply_to_distance(Double distance_km)
}

class Notification {
+ Integer notification_id
+ String message
+ LocalDateTime sent_at
+ String type
+ void send(User recipient)
}

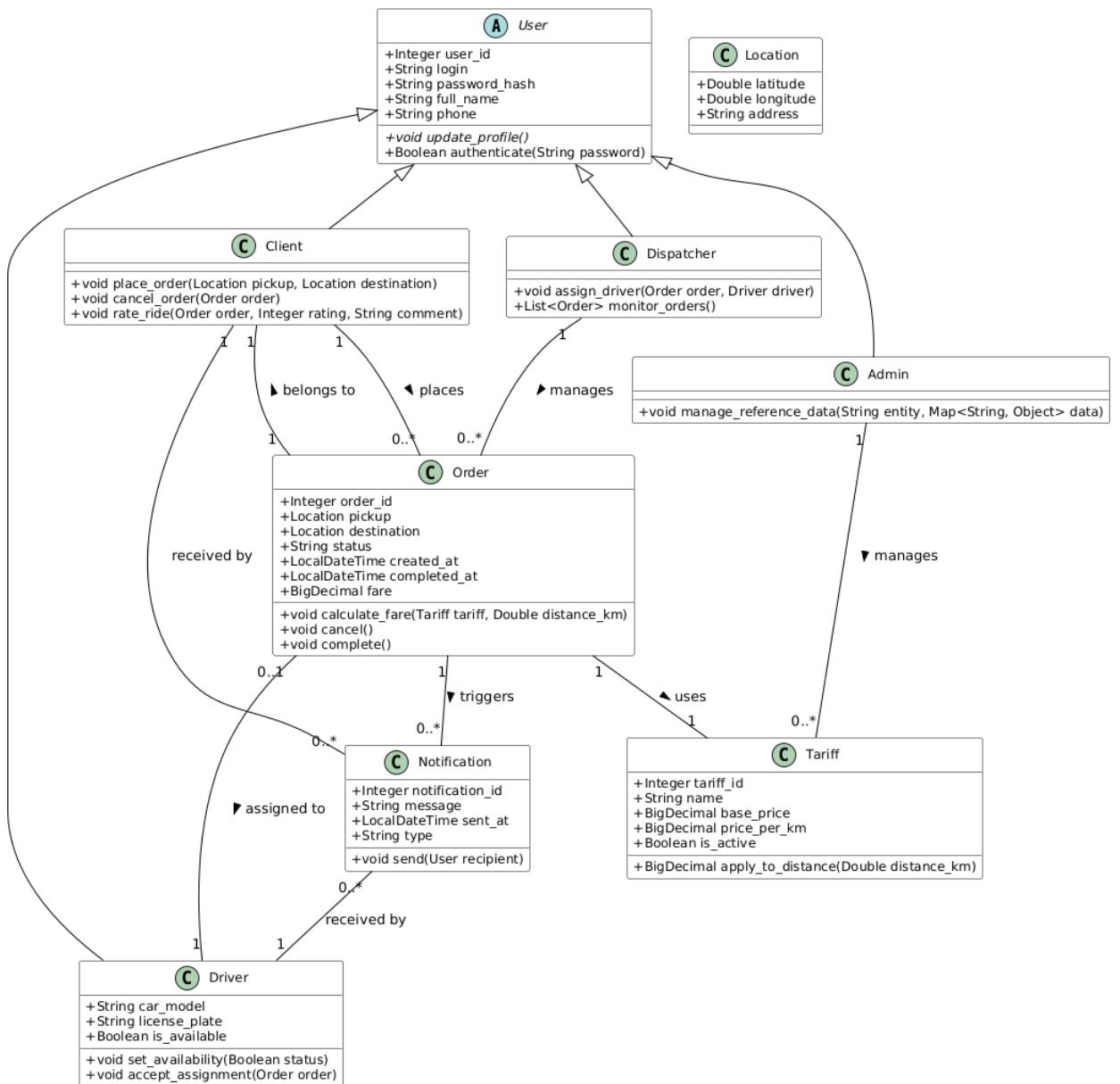
class Location {
+ Double latitude
+ Double longitude
+ String address
}

' ===== Наследование =====
User <|-- Client
User <|-- Driver
User <|-- Dispatcher
User <|-- Admin

' ===== Ассоциации с кратностями =====
Client "1" -- "0..*" Order : places >
Order "1" -- "1" Tariff : uses >
Order "0..1" -- "1" Driver : assigned to >
Order "1" -- "1" Client : belongs to >
Dispatcher "1" -- "0..*" Order : manages >
Admin "1" -- "0..*" Tariff : manages >
Order "1" -- "0..*" Notification : triggers >
Notification "0..*" -- "1" Client : received by
Notification "0..*" -- "1" Driver : received by

```

@enduml



В качестве нейросети был использован Deepseek. Реализация кода PlantUML производилась в онлайн-сервисе: <https://editor.plantuml.com/>

ПРОМТ: Промт: Информационная система управления службой такси. Построить диаграмму вариантов использования с учетом следующих требований:

1. Отобразить все действующие лица
2. Определить основные и вспомогательные варианты использования
3. Установить отношения между вариантами использования (include, extend)
4. Добавить границу системы
5. Обеспечить логическую целостность диаграммы

Сделай

в

PlantUML

Вывод:

```
@startuml
title Диаграмма классов системы управления такси

' ===== ПАКЕТЫ =====

package "Бизнес-сущности" {
    class User <<abstract>> {
        -id: UUID
        -name: String
        -phone: String
        -email: String
        -registrationDate: Date
        +authenticate(): Boolean
        +updateProfile(): void
    }

    class Customer {
        -rating: Double
        -paymentMethods: List<PaymentMethod>
        -favoriteAddresses: List<Address>
        +createOrder(): Order
        +rateTrip(): void
        +addPaymentMethod(): void
    }

    class Driver {
        -licenseNumber: String
    }
}
```

```
-car: Car  
-status: DriverStatus  
-rating: Double  
-currentLocation: Location  
+acceptOrder(): void  
+updateLocation(): void  
+changeStatus(): void  
}
```

```
class Order {  
-id: UUID  
-status: OrderStatus  
-pickupLocation: Location  
-destination: Location  
-createdAt: DateTime  
-estimatedCost: Double  
-actualCost: Double  
+calculateCost(): Double  
+updateStatus(): void  
+assignDriver(): void  
}
```

```
class Car {  
-model: String  
-licensePlate: String  
-color: String  
-type: CarType  
-year: Integer  
+validate(): Boolean  
}
```

```
class Payment {  
-id: UUID  
-amount: Double  
-method: PaymentMethod  
-status: PaymentStatus  
-transactionId: String  
+process(): Boolean
```

```
+refund(): Boolean
}

}

package "Управление" {
    class Dispatcher {
        -assignedOrders: List<Order>
        +assignOrder(): void
        +monitorOrders(): List<Order>
        +resolveIssue(): void
    }
}

class Administrator {
    -permissions: List<Permission>
    +generateReport(): Report
    +manageTariffs(): void
    +manageDrivers(): void
    +viewStatistics(): Statistics
}
}

package "Сервисы" {
    interface LocationService {
        +getCurrentLocation(): Location
        +calculateRoute(): Route
        +estimateTime(): Duration
    }
}

interface PaymentService {
    +authorizePayment(): Boolean
    +capturePayment(): Boolean
    +refundPayment(): Boolean
}

interface NotificationService {
    +sendSMS(): void
    +sendPush(): void
    +sendEmail(): void
}
```

```
}
```

```
class GeoLocationService implements LocationService {  
    +getCurrentLocation(): Location  
    +calculateRoute(): Route  
    +estimateTime(): Duration  
    +findNearestDrivers(): List<Driver>  
}
```

```
class PaymentGateway implements PaymentService {  
    +authorizePayment(): Boolean  
    +capturePayment(): Boolean  
    +refundPayment(): Boolean  
    +generateReceipt(): Receipt  
}  
}
```

```
package "Значения и перечисления" {  
enum OrderStatus {  
    PENDING  
    ASSIGNED  
    IN_PROGRESS  
    COMPLETED  
    CANCELLED  
}
```

```
enum DriverStatus {  
    AVAILABLE  
    BUSY  
    OFFLINE  
    ON_BREAK  
}
```

```
enum PaymentMethod {  
    CARD  
    CASH  
    ELECTRONIC  
    CORPORATE
```

}

```
enum CarType {  
    ECONOMY  
    COMFORT  
    BUSINESS  
    VAN  
}
```

' ===== СВЯЗИ =====

User <|-- Customer

User <|-- Driver

User <|-- Dispatcher

User <|-- Administrator

Customer "1" -- "*" Order : создает

Driver "1" -- "*" Order : выполняет

Order "1" -- "1" Payment : содержит

Order "1" -- "1" Car : использует

Driver "1" -- "1" Car : управляет

Dispatcher "1" -- "*" Order : управляет

Administrator "1" -- "*" Driver : контролирует

Administrator "1" -- "*" Order : анализирует

Order "1" -- "1" OrderStatus : имеет

Driver "1" -- "1" DriverStatus : имеет

Payment "1" -- "1" PaymentMethod : использует

Car "1" -- "1" CarType : относится к

Order ..> LocationService : использует

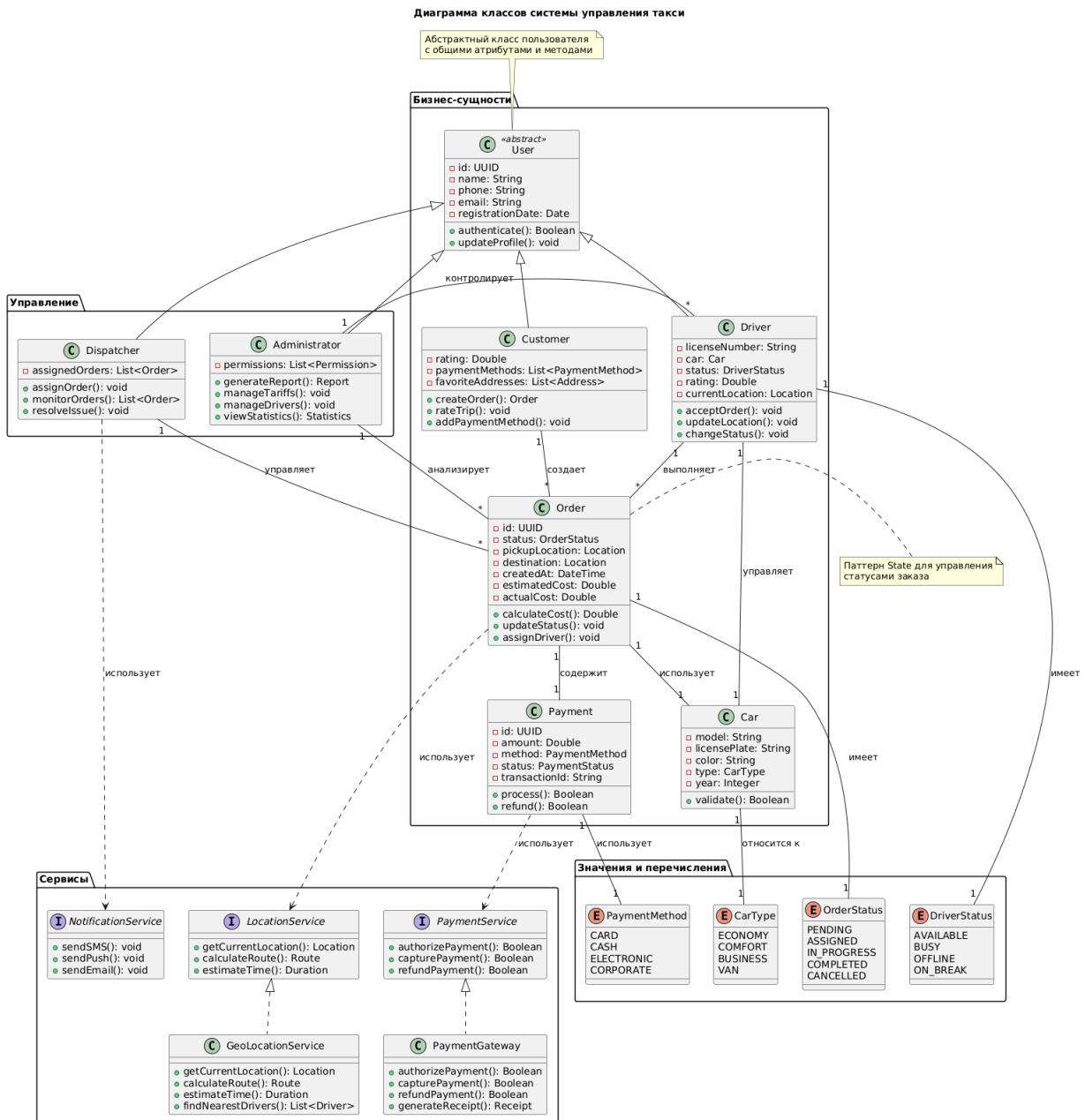
Payment ..> PaymentService : использует

Dispatcher ..> NotificationService : использует

note top of User : Абстрактный класс пользователя\nc общими атрибутами и методами

note right of Order : Паттерн State для управления\ncстатусами заказа

@enduml



ПРОМТ 2

Нужна пояснительная записка с описанием:

1. Выбранной предметной области
 2. Обоснования выбора действующих лиц
 3. Функциональных требований
 4. Структуры классов

ВЫВОД:

Пояснительная записка к диаграмме вариантов использования

Информационная система управления службой такси

1. Выбранная предметная область

1.1. Описание предметной области

Предметной областью является **система управления службой такси**, которая представляет собой комплексное программное решение для автоматизации процессов заказа, диспетчеризации, мониторинга и анализа деятельности такси-сервиса. Система охватывает полный цикл обслуживания клиентов от момента заказа до завершения поездки и финансовых расчетов.

1.2. Ключевые бизнес-процессы:

1. **Прием и обработка заказов** - регистрация заявок от клиентов
2. **Диспетчеризация** - распределение заказов между водителями
3. **Мониторинг поездок** - отслеживание выполнения заказов в реальном времени
4. **Финансовые расчеты** - обработка платежей и формирование отчетности
5. **Управление ресурсами** - контроль за автопарком и персоналом
6. **Аналитика и отчетность** - анализ эффективности работы службы

1.3. Технологический контекст:

- Интеграция с картографическими сервисами (Яндекс.Карты, Google Maps)
- Подключение к платежным системам (банковские карты, электронные кошельки)
- Мобильные приложения для клиентов и водителей
- Web-интерфейс для диспетчеров и администраторов

2. Обоснование выбора действующих лиц

2.1. Основные пользователи системы:

Действующее лицо	Роль в системе	Обоснование включения
Клиент	Потребитель услуг	Основной источник заказов, взаимодействует через мобильное приложение
Водитель	Исполнитель услуг	Непосредственно выполняет заказы, требует инструментов для работы
Диспетчер	Координатор	Управляет распределением заказов, решает конфликтные ситуации
Администратор	Управляющий	Контролирует работу системы, анализирует показатели эффективности

2.2. Внешние системы:

Внешняя система	Назначение	Обоснование интеграции
Платежная система	Обработка транзакций	Безопасная обработка финансовых операций, соответствует стандартам PCI DSS
Картографический сервис	Геолокация и навигация	Точное определение местоположения, построение маршрутов, расчет расстояний

2.3. Исключенные роли и обоснование:

- **Бухгалтер** - его функции включены в роль Администратора
- **Технический специалист** - рассматривается как часть команды поддержки, не взаимодействует напрямую с бизнес-логикой
- **Маркетолог** - использует аналитические отчеты, но не управляет операционной деятельностью

3. Функциональные требования

3.1. Требования высокого уровня:

F1. Управление заказами:

- **F1.1:** Система должна позволять клиентам создавать заказы такси
- **F1.2:** Система должна автоматически рассчитывать стоимость поездки

- **F1.3:** Система должна предоставлять выбор типа автомобиля
- **F1.4:** Система должна отслеживать статус выполнения заказа

F2. Управление профилями:

- **F2.1:** Система должна обеспечивать регистрацию и аутентификацию пользователей
- **F2.2:** Система должна хранить и управлять профилями клиентов
- **F2.3:** Система должна хранить и управлять профилями водителей
- **F2.4:** Система должна вести историю поездок и рейтинги

F3. Оплата и финансы:

- **F3.1:** Система должна интегрироваться с платежными системами
- **F3.2:** Система должна формировать электронные квитанции
- **F3.3:** Система должна обрабатывать возвраты средств
- **F3.4:** Система должна вести финансовую отчетность

F4. Администрирование:

- **F4.1:** Система должна предоставлять аналитические отчеты
- **F4.2:** Система должна позволять управлять тарифами
- **F4.3:** Система должна обеспечивать мониторинг операций в реальном времени
- **F4.4:** Система должна управлять составом водителей

3.2. Детальные требования:

Для варианта использования "Оформить заказ такси":

1. Определение местоположения клиента (автоматически или вручную)
2. Указание пункта назначения
3. Выбор типа обслуживания (эконом, комфорт, бизнес)
4. Расчет предварительной стоимости
5. Поиск и назначение подходящего водителя
6. Подтверждение заказа клиентом
7. Уведомление водителя о новом заказе

Для варианта использования "Обработать платеж":

1. Поддержка multiple payment methods (карта, электронный кошелек, наличные)
2. Авторизация платежа
3. Генерация транзакционного кода
4. Отправка подтверждения на email/SMS
5. Обновление финансовой отчетности

3.3. Нефункциональные требования:

Производительность:

- Время отклика системы ≤ 2 секунд
- Поддержка ≥ 1000 одновременных пользователей
- Обработка ≥ 100 заказов в минуту

Надежность:

- Доступность системы 99.9%
- Резервное копирование данных ежедневно
- Восстановление после сбоя ≤ 15 минут

Безопасность:

- Шифрование персональных данных
- Двухфакторная аутентификация для администраторов
- Соответствие GDPR для обработки персональных данных

4.2. Описание ключевых классов:

Базовый класс User:

- Абстрактный класс, содержащий общие атрибуты всех пользователей
- Инкапсулирует логику аутентификации и управления профилем
- Использует паттерн Наследование для специализации ролей

Класс Order:

- Центральная сущность системы
- Управляет жизненным циклом заказа через статусы (State pattern)
- Содержит бизнес-логику расчета стоимости
- Агрегирует информацию о поездке

Класс Driver:

- Управляет состоянием водителя (доступность, местоположение)

- Содержит информацию о транспортном средстве
- Ведет статистику и рейтинг

Класс Payment:

- Обрабатывает финансовые транзакции
- Интегрируется с внешними платежными системами через интерфейс
- Поддерживает различные методы оплаты

4.3. Принципы проектирования:

1. **Принцип единственной ответственности (SRP):**
 - Каждый класс имеет четко определенную зону ответственности
 - Сервисы выделены в отдельные компоненты
2. **Принцип открытости/закрытости (OCP):**
 - Интерфейсы для сервисов позволяют расширять функциональность
 - Добавление новых платежных систем не требует изменения существующего кода
3. **Принцип подстановки Барбары Лисков (LSP):**
 - Все конкретные пользователи могут быть использованы вместо абстрактного User
4. **Принцип разделения интерфейса (ISP):**
 - Интерфейсы сервисов узкоспециализированы
 - Клиенты зависят только от необходимых методов

4.4. Шаблоны проектирования:

1. **State Pattern:** Для управления статусами заказа (OrderStatus)
2. **Strategy Pattern:** Для различных алгоритмов расчета стоимости
3. **Observer Pattern:** Для уведомлений участников о изменениях статуса
4. **Factory Method:** Для создания различных типов пользователей
5. **Adapter Pattern:** Для интеграции с внешними сервисами

5. Заключение

Предложенная архитектура информационной системы управления службой такси обеспечивает:

1. **Масштабируемость:** Модульная структура позволяет добавлять новые функции
2. **Гибкость:** Использование интерфейсов и паттернов проектирования
3. **Надежность:** Четкое разделение ответственности между компонентами
4. **Безопасность:** Многоуровневая система аутентификации и авторизации
5. **Производительность:** Оптимизированная структура данных и алгоритмов

ПРОМТ: Сделай диаграмму диаграмму классов в plantuml

ВЫВОД:

```
@startuml
' ===== НАСТРОЙКИ =====
skinparam BackgroundColor #F8F9FA
skinparam ClassBackgroundColor #FFFFFF
skinparam ClassBorderColor #2C3E50
skinparam ClassFontColor #2C3E50
skinparam ArrowColor #2C3E50
skinparam ArrowFontColor #2C3E50
skinparam Shadowing false
skinparam RoundCorner 10
```

title Информационная система управления такси\nДиаграмма классов

```
' ===== ПЕРЕЧИСЛЕНИЯ =====
enum OrderStatus {
CREATED
SEARCHING_DRIVER
DRIVER_ASSIGNED
IN_PROGRESS
COMPLETED
```

```

    CANCELLED
}

enum DriverStatus {
    AVAILABLE
    BUSY
    OFFLINE
    ON_BREAK
    BLOCKED
}

enum PaymentStatus {
    PENDING
    SUCCESS
    FAILED
    REFUNDED
}

enum PaymentMethod {
    CASH
    CREDIT_CARD
    MOBILE_WALLET
    CORPORATE_ACCOUNT
}

enum CarType {
    ECONOMY
    COMFORT
    BUSINESS
    MINIVAN
    PREMIUM
}

' ===== АБСТРАКТНЫЕ КЛАССЫ И ИНТЕРФЕЙСЫ =====
abstract class User {
    {abstract} +id: Long
    {abstract} +name: String
    {abstract} +phoneNumber: String
    {abstract} +email: String
    {abstract} +registrationDate: Date
    {abstract} +passwordHash: String
    {abstract} +isActive: Boolean
    +authenticate(password: String): Boolean
    +updateProfile(data: UserData): void
    {abstract} +validate(): Boolean
}

interface LocationService {
    {abstract} +getCurrentLocation(userId: Long): GeoPoint
    {abstract} +calculateRoute(start: GeoPoint, end: GeoPoint): Route
    {abstract} +estimateTravelTime(route: Route, traffic: Boolean): Duration
    {abstract} +findNearestDrivers(location: GeoPoint, radius: Double): List<Driver>
}

interface PaymentProcessor {
    {abstract} +processPayment(orderId: Long, amount: Double, method: PaymentMethod): PaymentResult
    {abstract} +refundPayment(paymentId: Long): Boolean
}

```

```

    {abstract} +getPaymentStatus(paymentId: Long): PaymentStatus
}

interface NotificationService {
    {abstract} +sendSMS(phone: String, message: String): Boolean
    {abstract} +sendPushNotification(userId: Long, title: String, body: String): Boolean
    {abstract} +sendEmail(email: String, subject: String, content: String): Boolean
}

' ===== СУЩНОСТИ ПРЕДМЕТНОЙ ОБЛАСТИ =====
class Customer {
    -id: Long
    -name: String
    -phoneNumber: String
    -email: String
    -registrationDate: Date
    -passwordHash: String
    -isActive: Boolean
    -rating: Double
    -totalTrips: Integer
    -paymentMethods: List<PaymentMethod>
    -favoriteAddresses: List<Address>
    +createOrder(pickup: Address, destination: Address, carType: CarType): Order
    +rateTrip(orderId: Long, rating: Integer, comment: String): void
    +addPaymentMethod(method: PaymentMethod, details: String): void
    +getOrderHistory(limit: Integer): List<Order>
    +validate(): Boolean
}

class Driver {
    -id: Long
    -name: String
    -phoneNumber: String
    -email: String
    -registrationDate: Date
    -passwordHash: String
    -isActive: Boolean
    -licenseNumber: String
    -driverRating: Double
    -totalTrips: Integer
    -status: DriverStatus
    -currentLocation: GeoPoint
    -assignedCar: Car
    -workSchedule: Schedule
    +acceptOrder(orderId: Long): Boolean
    +declineOrder(orderId: Long): void
    +startTrip(orderId: Long): void
    +completeTrip(orderId: Long): void
    +updateLocation(location: GeoPoint): void
    +changeStatus(newStatus: DriverStatus): void
    +getEarnings(startDate: Date, endDate: Date): Double
    +validate(): Boolean
}

class Order {
    -id: Long
    -customer: Customer

```

```
-driver: Driver
-pickupAddress: Address
-destinationAddress: Address
-requestedCarType: CarType
-actualCar: Car
-status: OrderStatus
-createdAt: Timestamp
-acceptedAt: Timestamp
-startedAt: Timestamp
-completedAt: Timestamp
-estimatedCost: Double
-actualCost: Double
-distance: Double
-duration: Duration
-payment: Payment
-rating: Rating
+calculateEstimatedCost(): Double
+assignDriver(driver: Driver): Boolean
+updateStatus(newStatus: OrderStatus): void
+cancel(reason: String): Boolean
+addWaypoint(location: GeoPoint): void
+getRouteDetails(): Route
}
```

```
class Car {
    -id: Long
    -licensePlate: String
    -model: String
    -year: Integer
    -color: String
    -carType: CarType
    -vin: String
    -insuranceNumber: String
    -isActive: Boolean
    -lastMaintenance: Date
    -nextMaintenance: Date
    -features: List<String>
    +validate(): Boolean
    +isAvailable(): Boolean
    +scheduleMaintenance(date: Date): void
    +getAge(): Integer
}
```

```
class Payment {
    -id: Long
    -order: Order
    -amount: Double
    -paymentMethod: PaymentMethod
    -status: PaymentStatus
    -transactionId: String
    -createdAt: Timestamp
    -processedAt: Timestamp
    -receiptNumber: String
    +process(): PaymentResult
    +refund(reason: String): Boolean
    +generateReceipt(): Receipt
    +isSuccessful(): Boolean
}
```

```

}

class Rating {
    -id: Long
    -order: Order
    -fromCustomer: Boolean
    -score: Integer
    -comment: String
    -createdAt: Timestamp
    +validateScore(): Boolean
}

class Address {
    -id: Long
    -street: String
    -houseNumber: String
    -city: String
    -postalCode: String
    -country: String
    -geoPoint: GeoPoint
    -apartment: String
    -entrance: String
    -floor: String
    +getFullAddress(): String
    +validate(): Boolean
}

class GeoPoint {
    -latitude: Double
    -longitude: Double
    -accuracy: Double
    -timestamp: Timestamp
    +distanceTo(other: GeoPoint): Double
    +isValid(): Boolean
}

' ===== КЛАССЫ УПРАВЛЕНИЯ =====

class Dispatcher {
    -id: Long
    -name: String
    -phoneNumber: String
    -email: String
    -registrationDate: Date
    -passwordHash: String
    -isActive: Boolean
    -permissions: List<String>
    -shift: Shift
    +assignOrderToDriver(order: Order, driver: Driver): Boolean
    +findOptimalDriver(order: Order): Driver
    +monitorActiveOrders(): List<Order>
    +handleCustomerComplaint(orderId: Long, complaint: String): void
    +adjustOrder(orderId: Long, changes: OrderChanges): Boolean
    +validate(): Boolean
}

class Administrator {
    -id: Long

```

```

-name: String
-phoneNumber: String
-email: String
-registrationDate: Date
-passwordHash: String
-isActive: Boolean
-role: String
-permissions: List<String>
+manageDriver(driverId: Long, action: String, reason: String): Boolean
+manageCustomer(customerId: Long, action: String, reason: String): Boolean
+generateReport(type: String, startDate: Date, endDate: Date): Report
+updateTariff(tariffType: CarType, newPrice: Double): Boolean
+viewSystemStatistics(): SystemStats
+validate(): Boolean
}

class Tariff {
    -id: Long
    -carType: CarType
    -basePrice: Double
    -pricePerKm: Double
    -pricePerMinute: Double
    -minimumPrice: Double
    -isActive: Boolean
    -effectiveFrom: Date
    -effectiveTo: Date
    +calculatePrice(distance: Double, duration: Duration): Double
    +isApplicable(date: Date): Boolean
}

class Report {
    -id: Long
    -type: String
    -periodStart: Date
    -periodEnd: Date
    -generatedAt: Timestamp
    -data: JSON
    -format: String
    +generate(): void
    +export(format: String): File
    +getSummary(): String
}

' ===== СЕРВИСЫ =====
class GeoLocationServiceImpl implements LocationService {
    -apiKey: String
    -serviceUrl: String
    +getCurrentLocation(userId: Long): GeoPoint
    +calculateRoute(start: GeoPoint, end: GeoPoint): Route
    +estimateTravelTime(route: Route, traffic: Boolean): Duration
    +findNearestDrivers(location: GeoPoint, radius: Double): List<Driver>
    +getTrafficInfo(route: Route): TrafficData
}

class PaymentGateway implements PaymentProcessor {
    -merchantId: String
    -apiKey: String
}

```

```

-endpoint: String
+processPayment(orderId: Long, amount: Double, method: PaymentMethod): PaymentResult
+refundPayment(paymentId: Long): Boolean
+getPaymentStatus(paymentId: Long): PaymentStatus
+validateCard(cardNumber: String, cvv: String, expiry: Date): Boolean
}

class NotificationServiceImpl implements NotificationService {
    -smsProvider: String
    -pushService: String
    -emailServer: String
    +sendSMS(phone: String, message: String): Boolean
    +sendPushNotification(userId: Long, title: String, body: String): Boolean
    +sendEmail(email: String, subject: String, content: String): Boolean
    +scheduleNotification(time: Timestamp, notification: Notification): Boolean
}

' ===== СВЯЗИ НАСЛЕДОВАНИЯ =====
User <|-- Customer
User <|-- Driver
User <|-- Dispatcher
User <|-- Administrator

LocationService <.. GeoLocationServiceImpl
PaymentProcessor <.. PaymentGateway
NotificationService <.. NotificationServiceImpl

' ===== АССОЦИАЦИИ И АГРЕГАЦИИ =====
Customer "1" --> "*" Order : creates
Driver "1" --> "*" Order : completes
Order "1" --> "1" Payment : has
Order "1" --> "0..1" Rating : receives
Order "1" --> "1" Car : uses

Driver "1" --> "1" Car : drives
Car "1" --> "1" CarType : classified as
Order "1" --> "1" OrderStatus : has status
Driver "1" --> "1" DriverStatus : has status

Payment "1" --> "1" PaymentStatus : has status
Payment "1" --> "1" PaymentMethod : uses method

Dispatcher "1" --> "*" Order : manages
Administrator "1" --> "*" Driver : supervises
Administrator "1" --> "*" Customer : manages

Tariff "*" --> "1" CarType : defines price for
Order "1" --> "1" Tariff : applies

Customer "1" --> "*" Address : has favorites
Order "1" --> "1" Address : pickup from
Order "1" --> "1" Address : destination to

' ===== ЗАВИСИМОСТИ =====
Order ..> LocationService : uses for routing
Order ..> Tariff : calculates cost using
Payment ..> PaymentProcessor : processes via

```

Dispatcher ..> NotificationService : notifies via
Administrator ..> Report : generates

```
' ===== КОМПОЗИЦИИ =====
class Route {
    -points: List<GeoPoint>
    -totalDistance: Double
    -estimatedDuration: Duration
    +getInstructions(): List<String>
}
```

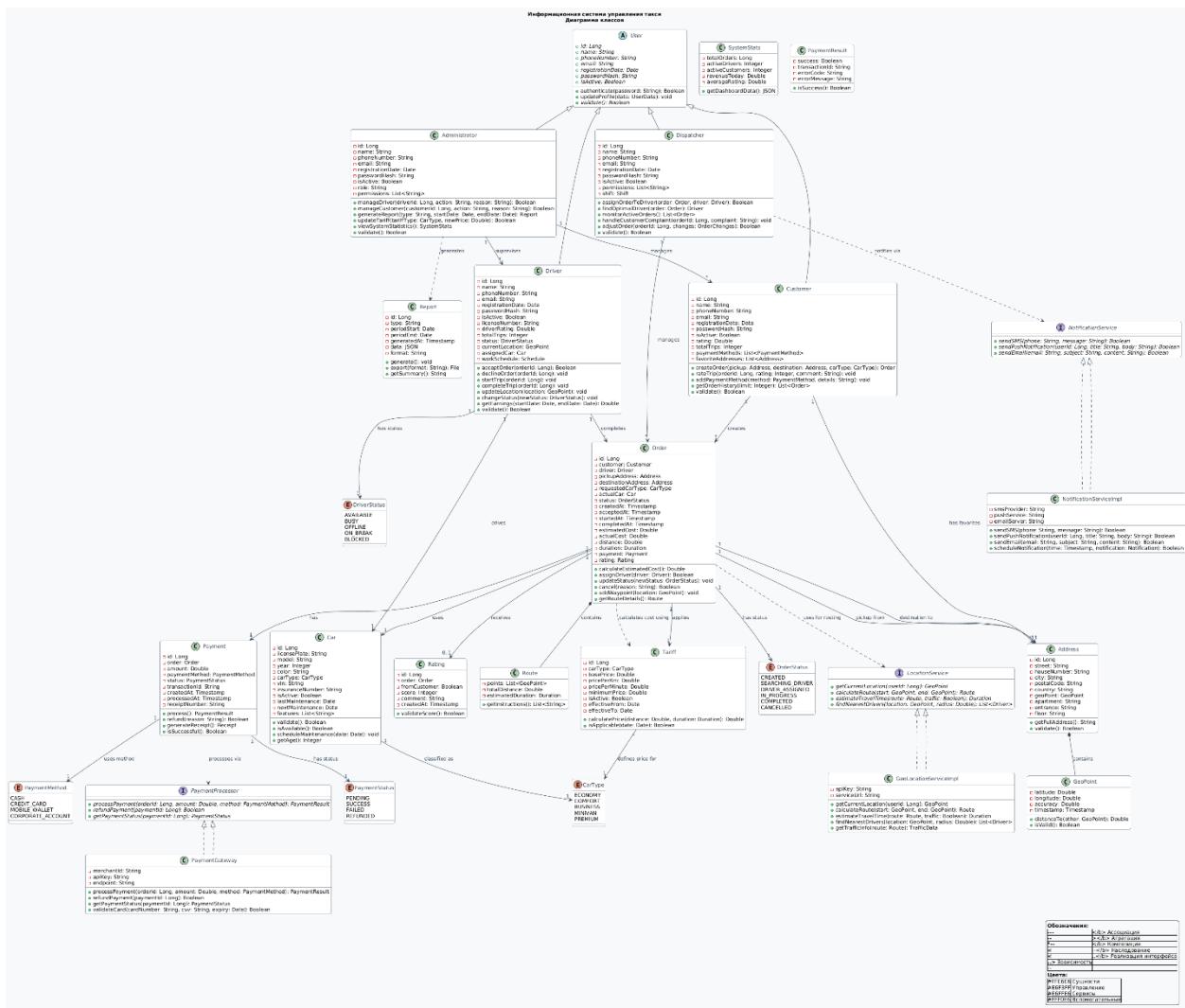
Order *-- Route : contains
Address *-- GeoPoint : contains

```
' ===== ВСПОМОГАТЕЛЬНЫЕ КЛАССЫ =====
class SystemStats {
    -totalOrders: Long
    -activeDrivers: Integer
    -activeCustomers: Integer
    -revenueToday: Double
    -averageRating: Double
    +getDashboardData(): JSON
}
```

```
class PaymentResult {
    -success: Boolean
    -transactionId: String
    -errorCode: String
    -errorMessage: String
    +isSuccess(): Boolean
}
```

```
' ===== ЛЕГЕНДА =====
legend right
<b>Обозначения:</b>
|<b>---|</b> Ассоциация|
|<b>--|></b> Агрегация|
|<b>*--|</b> Композиция|
|<b><|--</b> Наследование|
|<b><..</b> Реализация интерфейса|
|<b>..></b> Зависимость|
|--|
<b>Цвета:</b>
|#FFE6E6| Сущности|
|#E6F3FF| Управление|
|#E6FFE6| Сервисы|
|#FFF0E6| Вспомогательные|
endlegend
```

@enduml



ВЫВОД

1. Корректность синтаксиса PlantUML

Оценка: высокая

Плюсы:

- Все фрагменты кода корректно оформлены:
 - Используются @startuml / @enduml
 - Корректные объявления actor, usecase, class, interface, enum, package
 - Правильное применение стрелок (--, <--..>, *--)
 - Используются допустимые элементы PlantUML:
 - skinparam
 - note
 - legend
 - implements, abstract
 - Код явно проверялся в онлайн-редакторе PlantUML, что подтверждено в тексте отчёта.

Замечания (незначительные):

- В некоторых диаграммах классов наблюдается **избыточная детализация**, что не является ошибкой синтаксиса, но усложняет восприятие.
 - В одном из вариантов используется title Диаграмма классов системы управления такси при фактическом описании use-case — это **логическая, а не синтаксическая неточность**.

Синтаксис PlantUML полностью корректен.

2. Соответствие стандартам UML

Оценка: очень хорошая

Диаграммы вариантов использования:

- Корректно выделены:
 - Акторы
 - Граница системы
 - Основные и вспомогательные варианты использования
- Правильно применены отношения:
 - <<include>> — для обязательных подпроцессов
 - <<extend>> — для условных и optionalных сценариев
- Использование note соответствует UML для уточнения бизнес-ограничений.

Диаграммы классов:

- Соблюдаены ключевые элементы UML:
 - Наследование (User → Client, Driver, Dispatcher, Administrator)
 - Ассоциации с кратностями
 - Композиция и агрегация
 - Реализация интерфейсов
- Используются:
 - Абстрактные классы
 - Интерфейсы
 - Перечисления (enum)
- Видно осознанное применение шаблонов проектирования (State, Strategy, Observer).

Замечания:

- Некоторые диаграммы ближе к **архитектурной модели**, чем к классической учебной UML-диаграмме (допустимо, но требует комментария — он в отчёте есть).

Стандарты UML соблюдаены корректно и осмысленно.

3. Полнота отображения функциональных требований

Оценка: отличная

Покрытие требований:

- Все ключевые бизнес-процессы отражены:
 - Заказ поездки
 - Назначение водителя
 - Расчёт стоимости
 - Оплата
 - Уведомления
 - Администрирование
- В use-case диаграммах функциональные требования:
 - Чётко сопоставимы с актором
 - Логически сгруппированы
- В диаграммах классов:
 - Каждое требование имеет структурное отражение (класс + метод)

Сильная сторона отчёта:

- Наличие **пояснительной записи**, где функциональные требования:
 - Формализованы (F1, F2, F3...)
 - Связаны с диаграммами
 - Разделены на функциональные и нефункциональные

Возможное улучшение:

- Можно было явно связать требования с конкретными use-case (traceability), но это **не обязательно** для данной лабораторной.

Функциональные требования отражены полно и последовательно.

4. Качество визуализации

Оценка: хорошая / очень хорошая

Плюсы:

- Используются:
 - skinparam
 - Цветовые схемы
 - Группировка по пакетам

- Легенда обозначений
- Диаграммы визуально аккуратны и профессиональны
- Чётко различимы:
 - Сущности
 - Сервисы
 - Управляющие классы

Минусы:

- В крупных диаграммах классов:
 - Перегруженность элементами
 - Снижается читаемость без масштабирования
 - Для учебной работы иногда избыточна детализация (но это скорее плюс, чем минус).
- Качество визуализации высокое, но ближе к инженерному, чем учебному уровню.**

5. Логическая целостность диаграмм

Оценка: отличная

- Нет противоречий между:
 - Диаграммами вариантов использования
 - Диаграммами классов
 - Пояснительной запиской
- Роли акторов логически согласованы с их возможностями
- Связи между классами:
 - Обоснованы
 - Не дублируют друг друга
 - Соответствуют реальным бизнес-отношениям
- Видна единая концептуальная модель системы

Диаграммы логически непротиворечивы и целостны.

6. Обоснованность выбора предметной области

Оценка: отличная

- Предметная область:
 - Современная
 - Понятная
 - Богатая на бизнес-логику
- Хорошо подходит для:
 - Use-case диаграмм
 - Диаграмм классов
 - Демонстрации UML и PlantUML
- В пояснительной записке:
 - Обоснован выбор
 - Описаны бизнес-процессы
 - Указан технологический контекст

Выбор предметной области полностью оправдан и методически корректен.