Федеральное государственное бюджетное образовательное учреждение   
высшего образования

«Российский государственный педагогический университет им. А. И. ГЕРЦЕНА»

Кафедра информационных систем

Курсовая работа

по дисциплине «Методы и средства проектирования информационных систем и технологий»

на тему

«Проектирование базы данных «Пиццерии»»

Выполнил: студент 4 курса бакалавриата направления 09.03.02 – "Информационные системы и технологии"

Старцев Д.В.

Проверил:

**Содержание**

[**Введение 3**](#_Toc185179636)

[**Обзор литературы 4**](#_Toc185179637)

[**1 . Аналитический раздел 5**](#_Toc185179638)

[**1.1. Обзор предметной области 5**](#_Toc185179639)

[**1.2. Определение и обоснование требований к системе 6**](#_Toc185179640)

[**2 . Проектный раздел 7**](#_Toc185179641)

[**2.1. Разработка архитектуры системы 7**](#_Toc185179642)

[**2.2. Проектирование базы данных. 9**](#_Toc185179643)

[**3 . Технологический раздел 11**](#_Toc185179644)

[**3.1. Выбор средств разработки и инфраструктуры 11**](#_Toc185179645)

[**3.1.1. Реализация и описание фреймворка FastAPI 11**](#_Toc185179646)

[**3.1.2. Реализация в файле main.py: 11**](#_Toc185179647)

[**Список литературы 13**](#_Toc185179648)

Введение

Современные реалии заставляют бизнес активно искать пути оптимизации и повышения эффективности своей работы. Одной из таких востребованных сфер является организация проката автомобилей. Клиенты ожидают удобства и прозрачности, а компании — точности в учёте и удобства в управлении автопарком. Именно поэтому разработка программных решений для автоматизации процессов бронирования, контроля состояния транспортных средств и взаимодействия с клиентами приобретает первостепенное значение.

Цель данной работы — спроектировать такую систему проката автомобилей, которая не только облегчит жизнь менеджерам и операторам, но и улучшит клиентский опыт. Мы стремимся создать гибкий программный продукт, который можно будет без труда адаптировать под разные масштабы бизнеса: от небольших местных фирм до крупных сетевых операторов. Расширяемые модули, понятный интерфейс и удобные инструменты аналитики должны позволить компании чётко отслеживать состояние автопарка, оперативно реагировать на спрос и повышать качество обслуживания.

Конечным результатом станут упрощённые, прозрачные и удобные процессы аренды, обеспечивающие компании больший контроль, а клиентам — положительные впечатления от взаимодействия с сервисом.

**Обзор литературы**

В процессе проектирования системы проката автомобилей были изучены различные литературные источники, документация и материалы, связанные с разработкой баз данных, веб-сервисов и инструментов для построения пользовательских интерфейсов.

В источниках [1]-[3] представлены основные подходы к проектированию и организации баз данных с использованием PostgreSQL, включая создание таблиц, выполнение запросов и управление транзакциями. Это стало основой для реализации функциональности базы данных в рамках данной работы.

Документация [4]-[6] посвящена библиотекам Python, применяемым в проекте. В частности, [4] охватывает основные возможности FastAPI для построения веб-API, [5] объясняет использование SQLModel для взаимодействия с базой данных, а [6] предоставляет информацию о Pydantic, который применялся для валидации и сериализации данных.

Для разработки скриптов взаимодействия с базой данных и наполнения её тестовыми данными использовались материалы [7]-[9], в которых рассмотрены подходы к автоматизации работы с SQL-запросами и настройке серверной части.

В процессе работы над проектом также были изучены [10]-[11], где представлены лучшие практики проектирования пользовательских интерфейсов и маршрутизации в веб-приложениях.

1. Аналитический раздел
   1. Обзор предметной области

В современном секторе аренды автомобилей все чаще востребованы удобные, прозрачные и технологически продвинутые решения. Предметная область охватывает процессы, связанные с предоставлением клиенту временного доступа к автотранспорту при условии соблюдения определенных правил и оплаты соответствующих услуг. Основными участниками данных процессов являются:

* **Клиент** (пользователь системы) — лицо, желающее взять автомобиль в аренду на определенный срок. Он формирует запросы, просматривает доступные варианты, производит отбор по разным параметрам (марка, модель, доступность, место получения) и совершает бронирование.
* **Система учета бронирования** — центральный компонент, отвечающий за логику взаимодействия между клиентом, базой данных и внешними сервисами. Система осуществляет операции по созданию и обновлению заказов, хранит и обрабатывает информацию о наличии автомобилей, местах их получения, временных интервалах аренды и платежных статусах.
* **База данных** — хранилище структурированной информации об автопарке (марки, модели, характеристики авто), местах получения автомобилей (адреса станций проката или конкретных точек выдачи), а также данных о заказах, клиентах и транзакциях.
* **Интерфейс (веб-приложение или мобильное приложение)** — обеспечивает удобный доступ пользователя к функционалу системы. Через интерфейс клиент формирует запросы, просматривает доступные варианты, совершает бронирование и переходит к оплате.
* **Платежная система** — внешний сервис, интегрированный с системой, обеспечивающий проведение финансовых транзакций. Она осуществляет проверку реквизитов, обработку платежей по банковским картам, оповещает систему учета бронирования о статусе транзакции.

Основной сценарий взаимодействия в предметной области выглядит так: клиент входит в систему, запрашивает доступные автомобили по заданным критериям, получает перечень вариантов, выбирает подходящий автомобиль и место его получения. Далее система генерирует заказ и выставляет счет для оплаты. После успешной оплаты через платежный сервис статус бронирования обновляется, что подтверждает готовность автомобиля к выдаче в назначенное время.

Важными аспектами предметной области являются:

* Вариативность автопарка (разнообразие марок и моделей).
* Прозрачность и удобство процесса бронирования для клиента.
* Интеграция с платежными сервисами, гарантирующая безопасность и надежность финансовых операций.
* Гибкость в расширении функциональности (добавление новых пунктов выдачи, автомобилей, тарифных планов).
* Обеспечение корректного учета и обновления данных о заказах и платежах.
  1. Определение и обоснование требований к системе

Требования к системе могут быть разделены на функциональные и нефункциональные, при этом каждое требование должно быть обосновано с точки зрения практической необходимости и удобства для конечного пользователя, а также целесообразности для компании-оператора.

**Функциональные требования:**

1. **Предоставление актуального списка автомобилей с фильтрами (марка, модель, доступность):**  
   Обоснование: Клиенты должны иметь возможность оперативно находить подходящие автомобили согласно своим предпочтениям. Это ускоряет процесс выбора и повышает удовлетворенность клиента.
2. **Выбор места получения автомобиля и времени аренды:**  
   Обоснование: Гибкая система бронирования с указанием конкретного места и времени повышает точность планирования для клиента и помогает оптимизировать логистику компании.
3. **Создание и хранение заказа в базе данных:**  
   Обоснование: Необходимо надежно фиксировать факт бронирования, включая информацию о клиенте, автомобиле, адресе выдачи и сроках аренды. Хранение данных критично для последующей обработки, статистики и отчетности.
4. **Выставление счета и интеграция с платежными сервисами:**  
   Обоснование: Упрощает клиенту процесс оплаты, повышает доверие к сервису, делает взаимодействие более комфортным. Интеграция с надежной платежной системой гарантирует безопасность проведения транзакций.
5. **Обновление статуса заказа после оплаты:**  
   Обоснование: Автоматизация обновления статуса (оплачен, готов к выдаче) сокращает ручные операции, улучшает точность данных и ускоряет обслуживание.
6. Проектный раздел
   1. Разработка архитектуры системы

Для решения задачи была разработана следующая структура системы (Приложение 1).

Рис. 1 UML диаграмма

**Основные компоненты архитектуры:**

**Клиентский слой (Интерфейс)**

Этот слой — это то, что видят пользователи. Он представлен в виде веб-приложения или мобильного приложения, через которое люди могут искать и бронировать автомобили. Пользователи могут легко просматривать доступные машины, выбирать марку и модель, указывать, где они хотят забрать автомобиль, и оформлять бронирование. Интерфейс простой и интуитивно понятный, так что даже те, кто не очень разбирается в технологиях, смогут без труда ввести свои данные для оплаты и завершить процесс.

**Серверный слой (Система учета бронирования)**

Этот слой — обрабатывает все запросы от клиентского слоя. Здесь находится логика, благодаря которой система работает. Она выполняет несколько важных функций:

* Получает список автомобилей и применяет заданные пользователем фильтры.
* Определяет, доступность автомобилей, опираясь на данные из базы.
* Создает заказ и регистрирует его в системе.
* Обрабатывает платежи и взаимодействует с платежными сервисами.
* Обновляет статус заказа, когда приходит подтверждение от платёжной системы.

Этот серверный слой также формирует ответы, которые возвращаются обратно в интерфейс, чтобы пользователи могли видеть актуальную информацию.

**Уровень данных (База данных)**

Здесь хранится вся информация об автомобилях, их характеристиках и доступности. База обеспечивает целостность данных и поддерживает операции, такие как выбор, добавление, обновление и удаление записей. Это как большой архив, который всегда под рукой.

**Интеграционный слой (Платежная система)\***

Этот компонент отвечает за связку с внешними платежными сервисами. Когда пользователь вводит свои данные для оплаты, система отправляет их на проверку. Если всё в порядке, информация возвращается, и заказ обновляется. Это позволяет пользователям быстро и безопасно совершать платежи.

**Логика работы системы:**

1. Пользователь открывает приложение и начинает искать доступные автомобили, используя фильтры по марке и модели. Информация отправляется на сервер, который возвращает список подходящих вариантов.
2. После выбора автомобиля и указания места получения, а также дат аренды, приложение отправляет на сервер запрос на создание заказа. Сервер обращается к базе данных и создает новый заказ.
3. \* На этапе оплаты интерфейс передает данные на внешний платежный сервис. Как только платеж завершен, система обновляет статус заказа в базе данных.
4. \* Пользователь получает сообщение об успешном бронировании.

\* - не реализуется в курсовой работе

* 1. Проектирование базы данных.

При разработке системы проката автомобилей важно обеспечить целостность, структурированность и логическую связность данных. Проектируемая база данных должна позволять хранить информацию об автопарке, клиентах, заказах, а также справочниках статусов и справочной информации (модели, бренды, офисы выдачи).

**Основные принципы проектирования:**

1. **Нормализация данных:**  
   Структура таблиц приведена к нормальной форме, исключая дублирование и нарушение целостности данных. Ключевые сущности — автомобили, заказы, пользователи, справочники статусов, модели, бренды и офисы — разнесены по отдельным таблицам.  
   Данный подход упрощает обновление и модификацию данных, а также улучшает читаемость и масштабируемость базы.
2. **Использование первичных и внешних ключей:**  
   Каждая сущность имеет уникальный первичный ключ (PRIMARY KEY), упрощающий идентификацию и доступ к записям. Внешние ключи (FOREIGN KEY) обеспечивают связь между таблицами и гарантируют целостность данных, не позволяя сохранить запись, противоречащую логике предметной области.
3. **Именование полей и таблиц:**  
   Названия таблиц и столбцов отражают смысл хранимых данных (например, “users”, “cars”, “orders”, “models”). Для удобочитаемости столбцы названы в соответствии с их назначением (например, “vin\_code”, “car\_reg\_plate”).  
   Такой подход улучшает разборчивость схемы данных и облегчает дальнейшую поддержку.
4. **Использование справочных таблиц (status\_table\_...)**:  
   Для статусов автомобилей, заказов и офисов созданы отдельные таблицы, например:
   * **status\_table\_car** для статусов автомобиля (доступен, в ремонте, забронирован)
   * **status\_table\_order** для статусов заказа (новый, оплачен, отменен)
   * **status\_table\_office** для статусов офисов (активен, временно закрыт)

Это позволяет централизованно управлять справочными данными, легко добавлять новые статусы или менять логику, не затрагивая основную структуру.

1. **Разделение справочников по направлениям:**  
   Для марок и моделей автомобилей созданы отдельные таблицы “brands” и “models”, что обеспечивает гибкость в расширении ассортимента автопарка. Добавление новой марки или модели автомобиля не требует изменения структуры базовых таблиц.
2. **Работа с заказами и связью с другими сущностями:**  
   Таблица “orders” связана с “cars”, “users”, “office” и “status\_table\_order”.  
   Это отражает логику предметной области: каждый заказ принадлежит конкретному автомобилю (car\_id), оформлен определенным пользователем (user\_id), имеет статус (orders\_status\_id) и привязан к месту выдачи (office\_id). Поля “date\_start” и “date\_end” ограничены CHECK-условием, чтобы избежать некорректных временных интервалов.
3. **Хранение контактных данных клиентов:**  
   Таблица “users” содержит информацию о номере телефона (tel\_number), который служит уникальным идентификатором клиента. Это упрощает поиск клиентов и обеспечивает однозначную идентификацию без избыточных полей.
4. **Атрибуты, отражающие бизнес-логику:**  
   В таблице “cars” поля “date\_available” и “cost\_day” позволяют гибко управлять сроками доступности автомобиля для аренды и стоимостью аренды в сутки. Это обеспечивает гибкость в управлении прокатом и упрощает расчет итоговой суммы в заказах.

**Пример взаимодействия сущностей:**

* При создании нового заказа в таблицу “orders” вносятся данные о выбранном автомобиле, пользователе, месте получения автомобиля и временных параметрах аренды. Ссылка на “cars” (car\_id) обеспечивает связку с конкретным транспортным средством, а связь с “office” (office\_id) определяет пункт выдачи. Через внешние ключи обеспечивается недопущение некорректных записей (например, заказ не может быть создан для несуществующего автомобиля или пользователя).
* Статусы заказов динамически контролируются через статусные таблицы. При изменении состояния заказа (например, после оплаты или отмены) обновляется поле “orders\_status\_id”, что упрощает мониторинг состояния заказов и прозрачность операций.

1. Технологический раздел
   1. Выбор средств разработки и инфраструктуры
      1. **Реализация и описание фреймворка FastAPI**

При создании веб-сервиса для управления системой проката автомобилей ключевую роль играет серверная часть приложения. В данном проекте в качестве основы для реализации RESTful API используется фреймворк **FastAPI**, один из наиболее производительных и удобных инструментов на языке Python. FastAPI позволяет быстро и эффективно определять маршруты (эндпойнты), обрабатывать входящие запросы, возвращать ответы в удобном формате и легко интегрироваться с базами данных и внешними сервисами.

**Основные особенности и преимущества FastAPI:**

1. **Высокая производительность:**  
   FastAPI базируется на асинхронном веб-сервере Uvicorn/Hypercorn и использует концепции asyncio, что обеспечивает высокую скорость обработки запросов и масштабируемость. Это особенно актуально для системы проката автомобилей, где оперативная реакция на действия пользователя (просмотр доступных автомобилей, бронирование, оплата) критически важна.
2. **Аннотации типов и автоматическая документация:**  
   Благодаря использованию аннотаций типов в Python (typing) и интеграции со спецификацией OpenAPI (Swagger) FastAPI автоматически генерирует документацию по всем доступным эндпойнтам. Разработчикам и администраторам проще ориентироваться в сервисе, понимать формат ожидаемых данных и ответы, что упрощает поддержку и развитие проекта.
3. **Простота интеграции с базой данных и внешними сервисами:**  
   FastAPI легко сочетается с библиотеками SQLModel, SQLAlchemy или другими инструментами для работы с базами данных. В данном случае применена SQLModel и библиотека для соединения с PostgreSQL (create\_engine). Это упрощает написание запросов, выполнение операций CRUD и обработку данных.  
   Также предусмотрена интеграция с внешними API, например, для обработки платежей или получения дополнительных данных.
4. **Удобная работа с запросами и ответами:**  
   FastAPI использует модели данных на основе Pydantic для валидации входных и выходных данных. Это позволяет убедиться, что на сервер передаются корректные значения, а клиент получает стандартизированный, понятный ответ. В случае несоответствия данных или возникновения ошибок удобно возвращать HTTP-статусы и детальные описания проблем.
   * 1. **Реализация в файле main.py**
5. **Инициализация приложения и настройка CORS:**  
   В начале файла создается экземпляр приложения app = FastAPI(). Это основной объект, через который регистрируются маршруты и middleware.  
   Для обеспечения возможности обращений к API из различных доменов (например, при использовании SPA-фронтендов или мобильных приложений) добавляется CORSMiddleware, разрешающий запросы со стороны любых источников, что упрощает интеграцию с клиентскими интерфейсами.
6. **Определение маршрутов (эндпойнтов):**  
   В файле main.py используются декораторы @app.get(...) и @app.post(...) для определения маршрутов, обрабатывающих соответствующие HTTP-запросы (GET, POST и т.д.). Каждый эндпойнт возвращает либо словарь с данными, либо вызывает функцию, работающую с базой данных, и возвращает результат в формате JSON.  
   Например, эндпойнт /selects/cars возвращает список доступных автомобилей, вызывая соответствующую функцию select\_cars() из модуля selects. Аналогичным образом, /create\_order/ принимает данные заказа от клиента, создает запись в базе и отвечает сообщением об успехе операции.
7. **Валидация и обработка данных:**  
   Прием данных для создания заказа реализован через модель Pydantic (например, OrderCreate), которая описывает структуру входных данных и позволяет автоматически проверить корректность формата. Если отсутствуют обязательные поля или их тип некорректен, FastAPI вернет пользователю понятную ошибку.
8. **Возврат ответов и коды статусов HTTP:**  
   При нормальном выполнении операции FastAPI возвращает код 200 (OK) или 201 (Created) вместе с JSON-ответом. В случае ошибок можно выбрасывать исключение HTTPException с конкретным кодом статуса (например, 400, 404 или 500) и дополнительными данными, поясняющими причину ошибки.  
   Такой подход облегчает клиентам (приложениям или другим сервисам) корректное реагирование на различные ситуации.
9. **Расширяемость и поддержка:**  
   Опираясь на FastAPI, систему легко масштабировать и дополнять новыми функциями. Добавление новых маршрутов, интеграция с дополнительными сервисами, расширение схем данных — все это можно осуществить быстро и прозрачно для существующих пользователей API. Автоматическая документация (доступная по адресу /docs и /redoc) упрощает знакомство с новым функционалом.
10. Инициализация и наполнение базы данных начальными данными (файл gener\_bd.py)

В данном разделе рассматривается модуль, отвечающий за первоначальную загрузку данных в базу для отладки и тестирования работы системы проката автомобилей. Скрипт **gener\_bd.py** служит удобным инструментом для массового добавления записей в таблицы: он создает тестовые записи о брендах, моделях автомобилей, офисах выдачи, пользователях, заказах и статусах. Таким образом, система сразу получает необходимый объем актуальной информации, позволяющей проверить корректную работу всех модулей и эндпойнтов

* 1. Основные задачи и функциональность файла gener\_bd.py

1. **Установка соединения с базой данных:**  
   В начале скрипта создается подключение к PostgreSQL при помощи create\_engine и инициируется базовая структура данных, определенная в модели (SQLModel).
2. **Загрузка справочной информации:**  
   Скрипт добавляет в базу основные справочные данные:
   * **Марки и модели автомобилей:** Формируется база доступных к аренде марок и моделей машин, что позволяет проверить фильтрацию и подбор автомобилей под нужды клиента.
   * **Статусы объектов:** Вносятся статусы для автомобилей, офисов и заказов (например, «Available», «In Service», «Pending», «Confirmed», «Closed» и т.д.). Это обеспечивает реалистичные сценарии использования, когда часть автомобилей может находиться в ремонте или заказ может быть ожидающим оплаты.
3. **Добавление тестовых автомобилей, офисов и пользователей:**  
   В таблицы вносятся несколько тестовых автомобилей с разными характеристиками и стоимостью аренды, офисы с различным статусом, а также пользователи с уникальными телефонными номерами. Такой подход позволяет проверить корректность работы запросов по выбору свободных автомобилей или выбору офисов выдачи.
4. **Создание тестовых заказов:**  
   Генерируются тестовые заказы, содержащие ссылки на конкретные автомобили, пользователей, офисы, а также даты аренды и суммы. Это дает возможность убедиться, что запросы на создание и выбор заказов работают корректно, статусы заказов меняются ожидаемо, а данные о сроках аренды и суммах корректно сохраняются в базе.
5. **Автоматизация и удобство тестирования:**  
   Благодаря использованию скрипта **gener\_bd.py**, при необходимости можно быстро очистить и заново наполнить базу актуальными тестовыми данными. Это упрощает процесс отладки и тестирования новых функций, особенно на этапе активной разработки и внесения изменений.

**Итог:**  
Модуль **gener\_bd.py** является полезным инструментом для быстрой и удобной инициализации и заполнения базы данных начальными данными. Он обеспечивает подготовку инфраструктуры для полноценного тестирования системы проката автомобилей, включая проверку бизнес-логики, функционала интерфейса и корректной интеграции с внешними сервисами.

1. Реализация функций выборки и изменения данных (файл selects.py)

В данном разделе рассматривается функциональность, сосредоточенная на осуществлении запросов к базе данных и выполнении операций, связанных с чтением, фильтрацией, обновлением и созданием новых записей. Файл **selects.py** объединяет набор функций, которые предоставляют доступ к информации об автомобилях, заказах, офисах и пользователях, а также позволяют изменять статусы и создавать новые заказы.

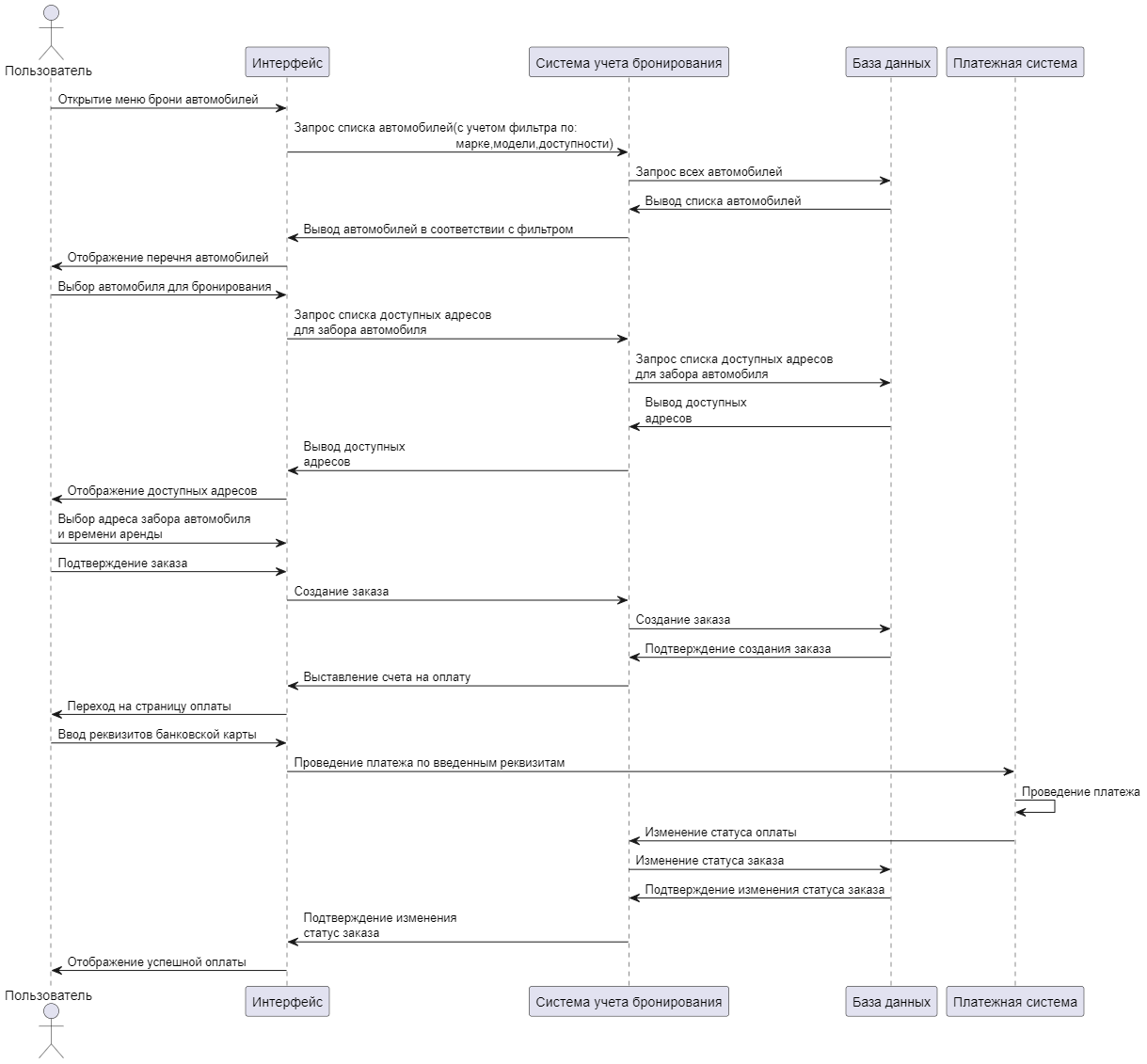
* 1. Основные возможности файла selects.py:

1. **Выборка данных о наличии и характеристиках автомобилей:**  
   Функции, такие как select\_cars(), select\_available\_cars() и select\_available\_cars\_and\_rent\_end\_cars(), осуществляют сложные запросы к связанным таблицам (cars, models, brands, status\_table\_car). Они возвращают набор данных с полными сведениями о каждом автомобиле, его статусе, марке, модели, стоимости аренды и доступности. Благодаря этому, клиентская часть системы может динамически формировать предложения для пользователей, подбирая подходящие автомобили по требованиям.
2. **Получение информации о заказах и офисах:**  
   Методы, такие как select\_orders() и select\_adress\_pickup(), предоставляют доступ к текущим заказам и точкам выдачи автомобилей. Так, система способна отображать историю заказов для аналитики и контроля, а также предоставить клиенту актуальный список офисов, работающих в данный момент.
3. **Операции изменения статусов и создание заказов:**  
   Функции bron\_rent\_car() и create\_order() служат для изменения состояний в базе данных. Первая переводит автомобиль в статус «Rented», фиксируя факт бронирования. Вторая создает полноценный заказ на аренду автомобиля, автоматически рассчитывает сумму, проверяет и при необходимости регистрирует пользователя, а также обновляет статус автомобиля. Это упрощает всю бизнес-логику: от обработки заявок до ведения учета автопарка.
4. **Гибкое расширение функционала:**  
   В процессе дальнейшей разработки можно добавлять новые фильтры, расширять логику существующих функций или внедрять новые методы работы с данными. Использование SQLModel и ORM-оберток упрощает модификации структуры базы данных и внедрение новых функций без кардинальных изменений в коде запросов.

**Результат:**  
Файл **selects.py** обеспечивает удобный, надежный и гибкий механизм взаимодействия с базой данных, поддерживая как чтение и анализ данных, так и обновление информации о заказах, автомобилях и пользователях. Это служит фундаментом для реализации богатого функционала системы проката, позволяя быстро и эффективно масштабировать возможности приложения.

Список литературы

1. PostgreSQL Documentation. Official website: [Электронный ресурс]. – Режим доступа: <https://www.postgresql.org/docs/> (дата обращения: 15.12.2024).
2. Книга "PostgreSQL для профессионалов" / Издательство: Piter. – 2020. – 352 с.
3. Электронный ресурс: "PostgreSQL для начинающих". – Режим доступа: https://habr.com/ru/companies/tensor/articles/779698/ (дата обращения: 15.12.2024).
4. FastAPI Documentation. Official website: [Электронный ресурс]. – Режим доступа: <https://fastapi.tiangolo.com/> (дата обращения: 15.12.2024).
5. SQLModel Documentation. Official website: [Электронный ресурс]. – Режим доступа: <https://sqlmodel.tiangolo.com/> (дата обращения: 15.12.2024).
6. Pydantic Documentation. Official website: [Электронный ресурс]. – Режим доступа: <https://docs.pydantic.dev/> (дата обращения: 15.12.2024).
7. Официальная документация библиотеки psycopg2: [Электронный ресурс]. – Режим доступа: <https://www.psycopg.org/docs/> (дата обращения: 15.12.2024).
8. Python документация: Модуль datetime: [Электронный ресурс]. – Режим доступа: <https://docs.python.org/3/library/datetime.html> (дата обращения: 15.12.2024).
9. Документация SQLAlchemy: [Электронный ресурс]. – Режим доступа: <https://docs.sqlalchemy.org/> (дата обращения: 15.12.2024).
10. "Современные веб-приложения на Python: от простого к сложному". – Издательство: DMK Press, 2021. – 420 с.
11. Руководство по проектированию интерфейсов от Nielsen Norman Group: [Электронный ресурс]. – Режим доступа: <https://www.nngroup.com/articles/> (дата обращения: 15.12.2024).
12. UML диаграмма



1. Ссылка на проект на GitHub

https://github.com/Jantisss/kurs\_project\_rent