МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Информационные системы и технологии

Специальность 1–40 01 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОЙ РАБОТЕ НА ТЕМУ:**

«Реализация базы данных аренды автомобилей с применением технологии Spatial and Graph в БД»

Выполнил студент Коршун Никита Игоревич

(Ф.И.О.)

Руководитель работы асс. Нистюк О.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

И.о. зав. кафедрой ст. преп. Блинова Е.А.

(учен. степень, звание, должность, Ф.И.О., подпись)

Курсовая работа защищена с оценкой

Минск 2023

**Содержание**

[Введение 3](#_Toc153977653)

[1 Постановка задачи 4](#_Toc153977654)

[1.1 Аналитический обзор аналогов и литературных решений 4](#_Toc153977655)

[1.2 Анализ и разработка функциональных требований 6](#_Toc153977656)

[2 Проектирование базы данных 8](#_Toc153977657)

[3 Разработка необходимых объектов 11](#_Toc153977658)

[3.1 Таблицы 11](#_Toc153977659)

[3.2 Пользователи и роли 13](#_Toc153977660)

[3.3 Процедуры и функции 13](#_Toc153977661)

[4 Описание процедур импорта и экспорта 14](#_Toc153977662)

[4.1 Процедура импорта данных в БД 14](#_Toc153977663)

[4.2 Процедура экспорта данных из БД 14](#_Toc153977664)

[5 Тестирование производительности 16](#_Toc153977665)

[6 Описание технологии применяемой в БД 19](#_Toc153977666)

[6.1 Spatial 19](#_Toc153977667)

[6.2 Graphs 20](#_Toc153977668)

[7 Краткое описание приложения 21](#_Toc153977669)

[8 Руководство пользователя 22](#_Toc153977670)

[Заключение 25](#_Toc153977671)

[Список используемых источников 26](#_Toc153977672)

[Приложение А 27](#_Toc153977673)

[Приложение Б 30](#_Toc153977674)

[Приложение В 38](#_Toc153977675)



# Введение

Цель данной работы заключается в создании реляционной базы данных для веб-приложения по бронированию авто, которое позволит клиентам брать в аренду транспорт на определенный срок.

База данных – это организованное собрание данных, которое обычно хранится в электронном виде в компьютерной системе. БД используются для хранения, организации и управления большим объемом структурированных и неструктурированных данных. Реляционная база данных является наиболее распространенной формой организации данных, в которой данные представлены в виде таблиц, состоящих из строк и столбцов, где каждый столбец представляет атрибут, а каждая строка представляет кортеж или запись. В данной работе для управления базой данных была выбрана СУБД Oracle, поскольку эта система обладает большим функционалом, что позволяет обеспечить эффективное хранение, обработку и управление данными.

Также необходимо разработать приложение для демонстрации функциональности базы данных и взаимодействия с ней. Приложение было реализовано с использованием фреймворка Django Framework на языке Python и HTML с использованием JavaScript.

Для сохранения данных местоположения автомобилей в курсовой работе применяется технология Spatial. Так же будут использоваться графы для получения связей между элементами.

Основные требования к приложению:

* Реализация ролей администратора, пользователя;
* Добавление и управление автомобилями;
* Резервирование автомобилей с указанием дат начала и окончания аренды;
* Управление заказами и генерация квитанций;
* Анализ данных (статистика, доход, популярные автомобили);
* Управление пользователями.

Содержание данной пояснительной записки отражает этапы выполнения курсового проекта.

1. Постановка задачи

Задачи проекта: разработать архитектуру базы данных с использованием технологий spatial and graph, создать процедуры и функции для работы с базой данных, взаимодействие с которыми будет понятно любому пользователю. Разработать веб-приложение на Python Django, используя встроенные функции Django для работы с базой данных.

Цели приложения состоит в демонстрации возможностей базы данных и предоставление удобного инструмента для бронирования автомобилей. Приложение должно обеспечивать эффективное и надежное бронирование автомобилей, обеспечивая при этом высокую производительность и безопасность данных.

В соответствии с заданием курсового проекта следует не только создать базу данных, но и разработать программное средство, которое должно в полной мере демонстрировать возможности базы данных.

Для того, чтобы сформировать окончательные требования к проектируемому программному средству сначала рассмотрим прототипы из той же области.

* 1. Аналитический обзор аналогов и литературных решений

Немаловажным этапом в разработке программного продукта является аналитический обзор прототипов и литературных источников.

На сегодняшний день существует множество аналогичных приложений или веб-сервисов для бронирования автомобилей. Поэтому было рассмотрено несколько аналогов.

Мобильное приложение «CarBooking» – приложение, предназначенное для бронирования автомобилей, а также помогает в выборе автомобиля и предлагает оптимальные решения для проблем, связанных с бронированием.

Интерфейс приведён на рисунке 1.1.

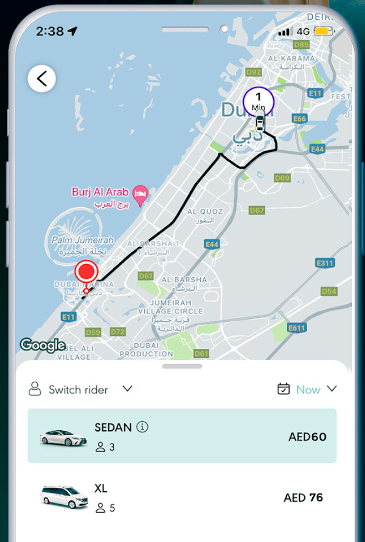


Рисунок 1.1 – Интерфейс приложения «CarBooking»

Проанализировав «CarBooking», можно выделить основные минусы и плюсы данного программного средства.

Основные плюсы:

* удобный интерфейс;
* бесплатный;
* имеется чат для общения с оператором;
* имеется карта стоянок.

Основные минусы:

* навязчивая реклама;

Мобильное приложение «CarRental» – приложение является мобильным сервисом для бронирования автомобилей с встроенным калькулятором стоимости.

Интерфейс приведён на рисунке 1.2.

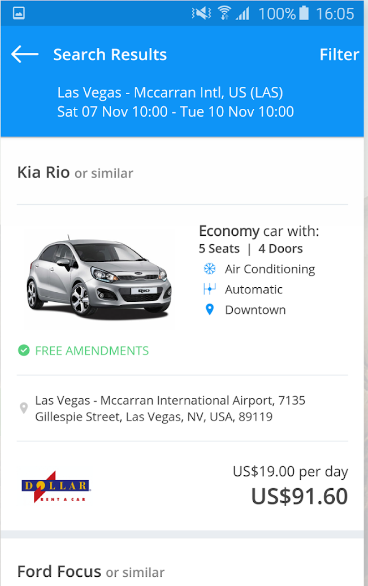


Рисунок 1.2 ­– интерфейс приложения «CarRental»

Проанализировав «CarRental», можно выделить основные минусы и плюсы данного программного средства.

Основные плюсы:

* есть калькулятор стоимости;
* большой выбор автомобилей.

Основные минусы:

* неудобный интерфейс;

Таким образом был выполнен анализ необходимого функционала, предоставленный аналогами разрабатываемого приложения.

* 1. Анализ и разработка функциональных требований

Анализ требований – процесс сбора требований к программному обеспечению, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки ПО.

Цель анализа требований в проектах – получить максимум информации о заказчике и специфике его задач, уточнить рамки проекта, оценить возможные сопутствующие риски.

На этом этапе происходит идентификация принципиальных требований методологического и технологического характера, формулируются цели и задачи создаваемого проекта.

Определение и описание требований – шаги, которые во многом определяют успех всего проекта, поскольку именно они влияют на все остальные этапы.

Различают три уровня требований к проекту:

* бизнес-требования;
* пользовательские требования;
* функциональные требования.

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга.

Курсовой проект не подразумевает наличие заказчика, который мог бы выдвинуть бизнес-требования, поэтому в качестве таких высокоуровневых требований будут рассматривать общие требования к разрабатываемому курсовому проекту.

К их этим требованиям относятся:

* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД);
* доступ к данным должен осуществляться только через соответствующие процедуры.

Следующими требованиями являются требования пользователей приложения, а именно администратора и пользователя.

Данные требования описывают цели и задачи, которые пользователям позволит решить система. Таким образом, в пользовательских требованиях указано, что клиенты смогут делать с помощью системы.

Пользователь данного программного решения должен иметь возможности, соответствующие его роли.

Возможности администратора:

* управление базой данных автомобилей;
* управление базой данных бронирований;
* управление базой данных пользователей;
* управление базой данных местоположений.

Возможности пользователя:

* бронирование автомобиля;
* просмотр своих бронирований;
* просмотр доступных автомобилей;
* просмотр местоположений автомобилей

После проведения анализа были выявлены следующие функциональные требования программного средства:

* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем различных данных;
* приложение должно корректным образом обрабатывать возникающие исключительные ситуации: отображать понятное для пользователя сообщение о возникшей ошибке;
* приложение должно предоставлять пользователям возможность создания нового аккаунта в виде регистрационной формы;
* приложение должно предоставлять возможность пользователям проходить аутентификацию и входить в систему под соответствующим введенным данным пользовательским именем.

Таким образом, был проведен тщательный анализ требований к программному средству, который позволил разработать список функциональных требований. Разработка данной программной системы должна проводиться в соответствии с сформированными списком.

1. Проектирование базы данных

Проектирование баз данных представляет собой процесс создания структуры базы данных и определения необходимых ограничений целостности.

Основные задачи проектирования базы данных:

* обеспечение хранения в БД всей необходимой информации;
* обеспечение возможности получения данных из таблиц по всем необходимым запросам;
* сокращение избыточности и дублирования данных;
* обеспечение целостности базы данных.

Проектирование базы данных проходит через два основных этапа: концептуальное и логическое проектирование.

Концептуальное проектирование представляет собой создание семантической модели предметной области, то есть информационной модели на высоком уровне абстракции.

Концептуально на данном этапе было выделено 6 сущностей:

* пользователь;
* местоположение;
* автомобиль;
* бронирование;
* узел;
* ребро.

Логическое проектирование – создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных.

Для реляционной модели данных логическая модель – набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи.

На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

Логическая модель базы со структурой связей представлена на рисунке 2.1. В ней показано как будут взаимодействовать таблицы между собой.

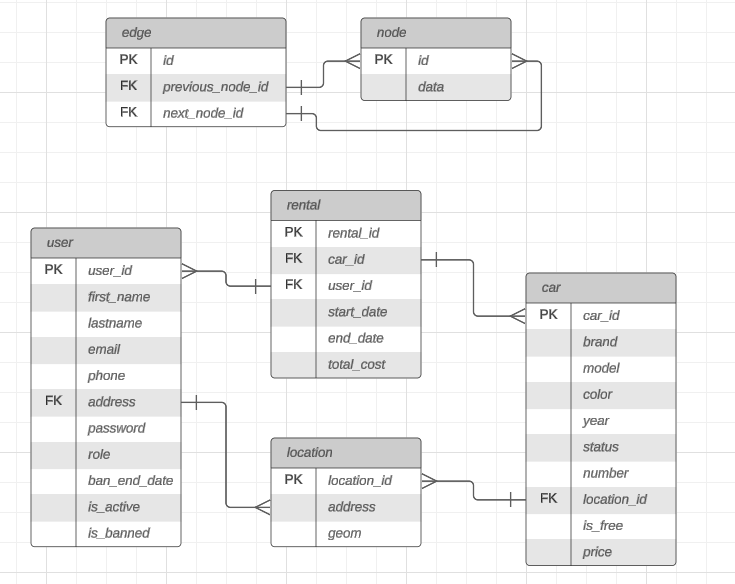


Рисунок 2.1 ­– Логическая модель базы данных

Всего в базе данных содержится 6 таблиц.

Таблица User предназначена для хранения данных зарегистрированных пользователей в базе данный.

Таблица Location предназначена для хранения данных о местоположениях.

Таблица Car предназначена для хранения данных об автомобилях.

Таблица Rental предназначена для хранения данных о бронированиях.

Таблица Node предназначена для хранения данных о узлах.

Таблица Edge предназначена для хранения данных о ребрах.

Так же была разработана use-case диаграмма, предоставленная на рисунке 2.2.

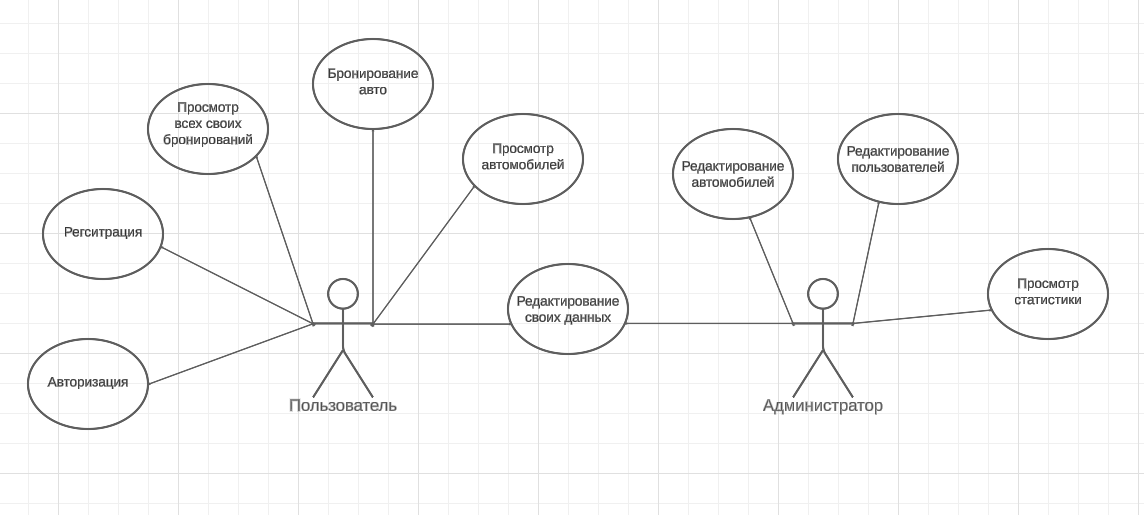


Рисунок 2.2 – Use-case диаграмма

Всего в базе данных 4 пользователя:

Пользователь ADMIN создан для представления администратора в базе данных. Данный пользователь имеет возможность управлять базой данных автомобилей, бронирований и пользователей, просмотра статистики.

Пользователь USER создан для представления пользователя в базе данных. Данный пользователь имеет возможность бронировать автомобиль, просматривать свои бронирования и просматривать доступные автомобили.

1. Разработка необходимых объектов
   1. Таблицы

Таблица – это совокупность связанных данных, хранящихся в структурированном виде в базе данных. Она состоит из столбцов и строк.

Столбцы таблицы называют полями; каждое поле характеризуется своим именем (названием соответствующего свойства) и типом данных, отражающих значения данного свойства. Каждое поле обладает определенным набором свойств (размер, формат и др.).

Поле базы данных – это столбец таблицы, включающий в себя значения определенного свойства.

В каждой таблице должно быть, по крайней мере, одно ключевое поле, содержимое которого уникально для любой записи в этой таблице.

Значения ключевого поля однозначно определяют каждую запись в таблице.

Для реализации базы данных «Реализация базы данных аренды автомобилей с применением технологии Spatial and Graph в БД» было разработано 6 таблиц: User, Car, Location, Rental, Node, Edge. Они будут реализованы в СУБД Postgresql.

Таблица Users представляет список всех зарегистрированных пользователей в базе данных (Таблица 3.1).

Таблица 3.1 – Столбцы таблицы User

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| user\_id | Идентификатор пользователя | int |
| first\_name | Имя пользователя | varchar |
| last\_name | Фамилия пользователя | varchar |
| email | Почта пользователя | varchar |
| phone | Телефон пользователя | varchar |
| password | Пароль пользователя | varchar |
| role | Дата рождения пользователя | varchar |
| ban\_end\_date | Электронная почта пользователя | date |
| is\_active | Активна ли учетная запись | boolean |
| address\_id | Идентификатор адреса пользователя | int |
| is\_banned | Заблокирована ли учетная запись | boolean |

Таблица Cars представляет список всех автомобилей (таблица 3.2).

Таблица 3.2 – Столбцы таблицы Car

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| car\_id | Идентификатор автомобиля | int |
| brand | Производитель автомобиля | varchar |
| model | Модель автомобиля | varchar |
| color | Цвет автомобиля | varchar |
| year | Год выпуска автомобиля | int |
| status | Состояние автомобиля | varchar |
| number | Номер автомобиля | varchar |
| is\_free | Свободна ли машина | boolean |
| price | Цена автомобиля за день | numeric |
| location\_id | Номер местоположения автомобиля | int |

Таблица Location представляет список всех местоположений(таблица 3.3).

Таблица 3.3 – Столбцы таблицы Location

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| location\_id | Идентификатор местоположения | int |
| address | Адрес | varchar |
| Geom | Координаты | Geometry |

Таблица Rental представляет список всех бронирований (таблица 3.4).

Таблица 3.4 – Столбцы таблицы Rental

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| rental\_id | Идентификатор бронирования | int |
| start\_date | Дата начала бронирования | date |
| end\_date | Дата конца бронирования | date |
| total\_cost | Сумма заказа | numeric |
| car\_id | Идентификатор автомобиля | int |
| user\_id | Идентификатор пользователя | int |

Таблица Node представляет список всех узлов (таблица 3.5).

Таблица 3.5 – Столбцы таблицы Node

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | Идентификатор узла | int |
| data | Данные узла | jsonb |

Таблица Edge представляет список всех ребер (таблица 3.6).

Таблица 3.6 – Столбцы таблицы Edge

|  |  |  |
| --- | --- | --- |
| Наименование | Описание | Тип |
| id | Идентификатор ребра | bigint |
| next\_node\_id | Идентификатор предыдущего узла | int |
| previous\_node\_id | Идентификатор следующего узла | int |

* 1. Пользователи и роли

В базе данных приложения есть одна основная таблица пользователей: User. В этой таблице есть поле `role`, которое определяет роль пользователя в системе. Всего в системе предусмотрено две роли: 'admin' и 'user'.

Администратор имеет полный доступ ко всем операциям в приложении. Он может управлять всеми данными в базе данных, включая автомобили, бронирования, местоположения и пользователей.

Пользователь имеет ограниченный доступ к операциям в приложении. Он может просматривать доступные автомобили, бронировать автомобили, просматривать свои бронирования и отменять свои бронирования.

Важно правильно управлять доступом к данным на основе роли пользователя, чтобы обеспечить безопасность и целостность вашей базы данных. Это также помогает обеспечить хороший пользовательский опыт, поскольку пользователи получают доступ только к тем функциям, которые им нужны.

* 1. Процедуры и функции

Вместо традиционного подхода с использованием пакетов и процедур в базе данных, приложение использует Django, фреймворк Python, для управления данными. Django предоставляет высокоуровневый API для взаимодействия с базой данных, который позволяет выполнять все необходимые операции без написания SQL-запросов.

Например, для получения всех пользователей используется User.objects.all(). Этот метод Django возвращает коллекцию объектов пользователей. Все эти операции выполняются внутри Django.

Таким образом, управление данными в приложении полностью осуществляется с помощью Django, что обеспечивает гибкость и удобство при работе с данными.

1. Описание процедур импорта и экспорта
   1. Процедура импорта данных в БД

Чтобы импортировать данный в БД был выбран формат XML.

Вместо традиционного подхода с использованием SQL и JSON для импорта данных, приложение использует Python и XML. Была разработана функция import\_streets\_from\_xml, которая считывает данные о улицах из XML-файла и сохраняет их в базе данных.

Код функции import\_streets\_from\_xml представлен в листинге 4.1.

def import\_streets\_from\_xml():

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'streetsWithCoordinates.xml')

tree = ET.parse(file\_path)

root = tree.getroot()

for street in root.findall('street'):

lat = street.get('lat')

lon = street.get('lon')

address = street.text

location = Location(address=address)

location.geom = f'POINT({lon} {lat})'

location.save()

Листинг 4.1 – Функция import\_streets\_from\_xml

* 1. Процедура экспорта данных из БД

Для экспорта данных из базы данных также используется Python и Django. Была разработана функция export\_data\_to\_xml, которая извлекает все объекты из таблиц базы данных, преобразует их в XML-представление и сохраняет их в отдельные файлы.

Код функции export\_data\_to\_xml представлен в листинге 4.2.

def export\_data\_to\_xml():

# Получаем все объекты из таблиц

locations = Location.objects.all()

cars = Car.objects.all()

users = User.objects.all()

rentals = Rental.objects.all()

# Преобразуем объекты в XML-представление и сохраняем их в отдельные файлы

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

locations\_file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'locations.xml')

cars\_file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'cars.xml')

users\_file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'users.xml')

rentals\_file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'rentals.xml')

with codecs.open(locations\_file\_path, 'w', encoding='utf-8') as locations\_file:

locations\_xml = serialize('xml', locations)

locations\_file.write(locations\_xml)

with codecs.open(cars\_file\_path, 'w', encoding='utf-8') as cars\_file:

cars\_xml = serialize('xml', cars)

cars\_file.write(cars\_xml)

with codecs.open(users\_file\_path, 'w', encoding='utf-8') as users\_file:

users\_xml = serialize('xml', users)

users\_file.write(users\_xml)

with codecs.open(rentals\_file\_path, 'w', encoding='utf-8') as rentals\_file:

rentals\_xml = serialize('xml', rentals)

rentals\_file.write(rentals\_xml)

print('Data exported successfully')

Листинг 4.2 – Функция export\_data\_to\_xml

1. Тестирование производительности

Для тестирования производительности БД в таблицу Rental было добавлено 100000 строк при помощи функции генерации. Для того что бы провести тесты цены запроса и времени его выполнения были разработаны три select запроса затрагивающие выборку с ключевыми полями и без них. Тестирование производиться при помощи инструментов pgAdmin.

Первый запрос, подремонтированный в листинге 5.1, необходим для теста стоимости и времени запроса при поиске строк с указанной суммой заказа.

SELECT \* FROM rental where total\_cost > 500;

Листинг 5.1 – Первый тестовый запрос

Результат первого тестового запроса показал, что время запроса составляет около 16ms, а среднее значение цены 993 что является не плохим результатом для поиска без ключевых или же индексированных полей в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.1.

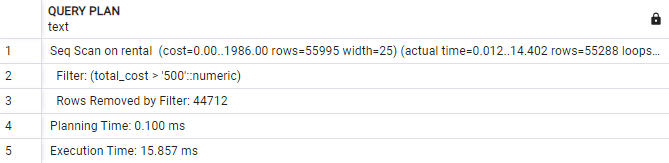


Рисунок 5.1 – Результат выполнения первого тестового запроса

Второй запрос, подремонтированный в листинге 5.2, необходим для теста стоимости и времени запроса при поиске строки с указанной почтой.

SELECT \* FROM main\_rental WHERE end\_date > '2023-12-25';

Листинг 5.2 – Второй тестовый запрос

Результат второго тестового запроса показал, что время запроса составляет около 12ms, что является хорошим результатом для поиска с ключевым или же индексированным полем в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.2.

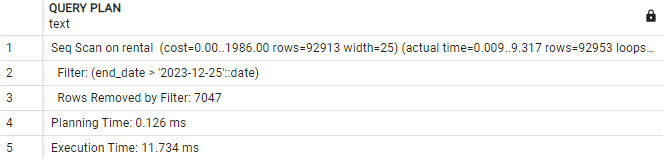


Рисунок 5.2 – Результат выполнения второго тестового запроса

Третий запрос, подремонтированный в листинге 5.3, необходим для теста стоимости и времени запроса при поиске строк с указанным идентификатором заказа. Данный тест является важным ведь в приложении часто используется подобный поиск.

SELECT \* FROM rental WHERE rental\_id = 56112;

Листинг 5.3 – Третий тестовый запрос

Результат третьего тестового запроса показал, что время запроса составляет около 0.036ms, что является не плохим результатом для поиска без ключевых или же индексированных полей в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.3.



Рисунок 5.3 – Результат выполнения третьего тестового запроса

Однако было принято решение оптимизировать именно этот тип запросов так как он является самым частым по отношению к данной таблице. Был разработан простой индекс код, которого предоставлен в листинге 5.4.

CREATE INDEX RENTALID\_INDEX ON rental (rental\_id);

Листинг 5.4 – Индекс для таблицы Rental

Повторной результат третьего тестового запроса показал, что время запроса практически не снизилось и составляет 0.036ms, что является хорошим результатом для поиска с ключевым или же индексированным полем в БД с 100000 строк. Результат выполнения предоставлен на рисунке 5.4.

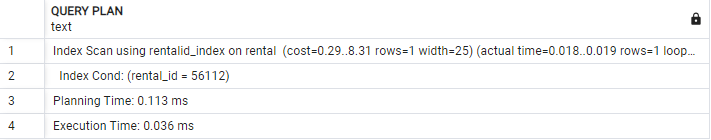


Рисунок 5.4 – Повторный результат выполнения третьего тестового запроса

По результатам проведённых в рамках тестирования производительности базы данных были проведены некоторые работы, направленные оптимизировать производительность. После проведения повторных тестов можно заявлять, что база данных является достаточно производительной что бы не вызывать проблем у пользователя, а также что база данных выдерживает 100000 строк без сильных падений в производительности.

1. Описание технологии применяемой в БД

В качестве основных технологий в БД были выбраны spatial и graphs. Spatial - это технология, которая позволяет работать с геометрическими данными. Spatial основана на геометрических объектах, таких как точки (координаты), линии (ребра) и площади (многоугольники). В приложении используются только точки для сохранения местоположения машин и пользователя. Графы - это математическая модель, которая описывает объекты (узлы) и связи между ними (ребра).

* 1. Spatial

Для реализации пространственных данных в курсовом проекте был использован тип данных geometry в PostgreSQL. Этот тип данных позволяет работать с геометрическими объектами, такими как точки, линии и полигоны. В контексте нашего веб-приложения для бронирования автомобилей, geometry используется для хранения и обработки координат, таких как долгота и широта.. Пример использования пространственных данных предоставлен в листинге 6.1.

locations = Location.objects.all()

for location in locations:

if str(location.address) == str(current\_user['address']):

user\_coordinates = {

'longitude': location.geom.x,

'latitude': location.geom.y,

}

for street in root.findall('street'):

lat = street.get('lat')

lon = street.get('lon')

address = street.text

location = Location(address=address)

location.geom = f'POINT({lon} {lat})'

location.save()

Листинг 6.1 – Пример использования пространственных данных.

Использование типа данных geometry в PostgreSQL, как видно из листинга 6.1, было выбрано для хранения и обработки геометрических данных, таких как долгота и широта. Эти данные используются для определения местоположения пользователей и автомобилей в веб-приложении.

* 1. Graphs

Для реализации графов в вашем веб-приложении для бронирования автомобилей были использованы две таблицы в PostgreSQL: node и edge. В таблице node сохраняются данные, а в таблице edge сохраняются связи этих данных.

В контексте вашего приложения, узлы (node) представляют собой клиентов и совершенные арендные сделки, а ребра (edge) - связи между ними (например, бронирование). Эта структура позволяет эффективно управлять связями между автомобилями и клиентами, а также предоставлять администраторам статистику по общему доходу, самым популярным автомобилям и т.п.

Процесс использования графов предоставлен в листинге 6.2.

def popular\_cars(self, obj):

with connection.cursor() as cursor:

cursor.execute("""

SELECT data->>'car' AS car, COUNT(\*) AS popularity

FROM main\_node

GROUP BY data->>'car'

ORDER BY popularity DESC;""")

return cursor.fetchall()

def get\_user\_outcome(self, obj):

with connection.cursor() as cursor:

query = """

SELECT next\_node\_id

FROM edge

WHERE previous\_node\_id = %s;

"""

cursor.execute(query, [obj.previous\_node\_id])

rows = cursor.fetchall()

car\_ids = [row[0] for row in rows]

user\_outcome = 0

for car\_id in car\_ids:

car\_node = Node.objects.get(id=car\_id)

car\_data = json.loads(car\_node.data['income'])

outcome = int(car\_data)

user\_outcome += outcome

Листинг 6.2 – Процесс маскирования телефонного номера пользователя

Маскирование используется для частичного сокрытия телефонного номера. Выбирается первые 8 символов, а остальное заполняется звёздами.

1. Краткое описание приложения

Для демонстрации работы БД было разработано веб-приложение, написанное на языке Python при помощи фреймворка Django с использованием JavaScript. Это веб-приложение с использованием архитектуры MVT.

Приложение состоит из 3 основных элементов:

Model (Модель) – представление базы данных. Это включает в себя важные поля и поведение данных, которые хранятся в базе данных.

View (Представление) отвечает за то, что видит пользователь. Представление извлекает данные из модели и передает их в шаблон.

Template (Шаблон) отвечает за то, как что-то должно быть отображено на веб-странице. Они описывают, как данные, полученные из представлений, вставляются в HTML для отображения на веб-странице.

Код методов представлений, который реализует всю обработку данных перед передачей в шаблон предоставлен в листинге 7.1.

def index(request):

def login(request):

def signup(request):

def account(request):

def update\_account(request):

def admin\_login(request):

def logout(request):

def delete\_account(request):

def car\_rent(request):

def addRent(request):

def generate\_payment\_receipt(request):

def validate\_email(email):

def addNewCars(num\_rows):

def addNewLocations():

def import\_streets\_from\_xml():

def get\_coordinates\_by\_street(street\_name):

def fillUsersTable(num\_rows):

def encrypt\_decrypt\_password(password, key):

def generate\_orders(num\_rows):

def export\_data\_to\_xml():

Листинг 7.1 – Код методов представлений

Данные методы вызываются при загрузке или отправке данных на веб страницу.

1. Руководство пользователя

Разработанное в рамках курсового проекта приложение поддерживает две основные роли: администратор и пользователь. Заходя на главную страницу пользователь видит карту с маркерами, обозначающими автомобили, также в верхней части приложения имеются две кнопки «Войти» и «Зарегистрироваться». Скриншот главной страницы предоставлен на рисунке 8.1.

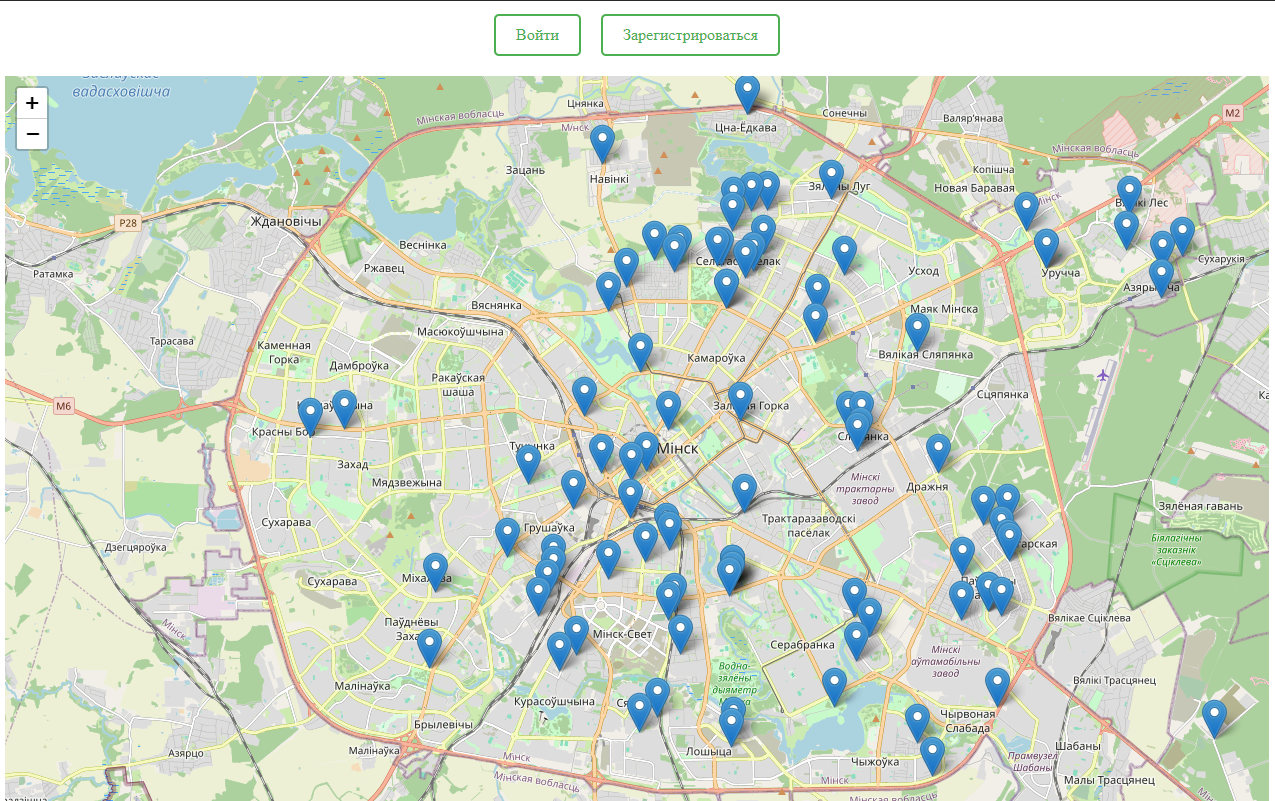


Рисунок 8.1 – Главная страница

При нажатии на кнопку «Зарегистрироваться» пользователь будет перенаправлен на страницу регистрации. Скриншот страницы регистрации приведен на рисунке 8.2

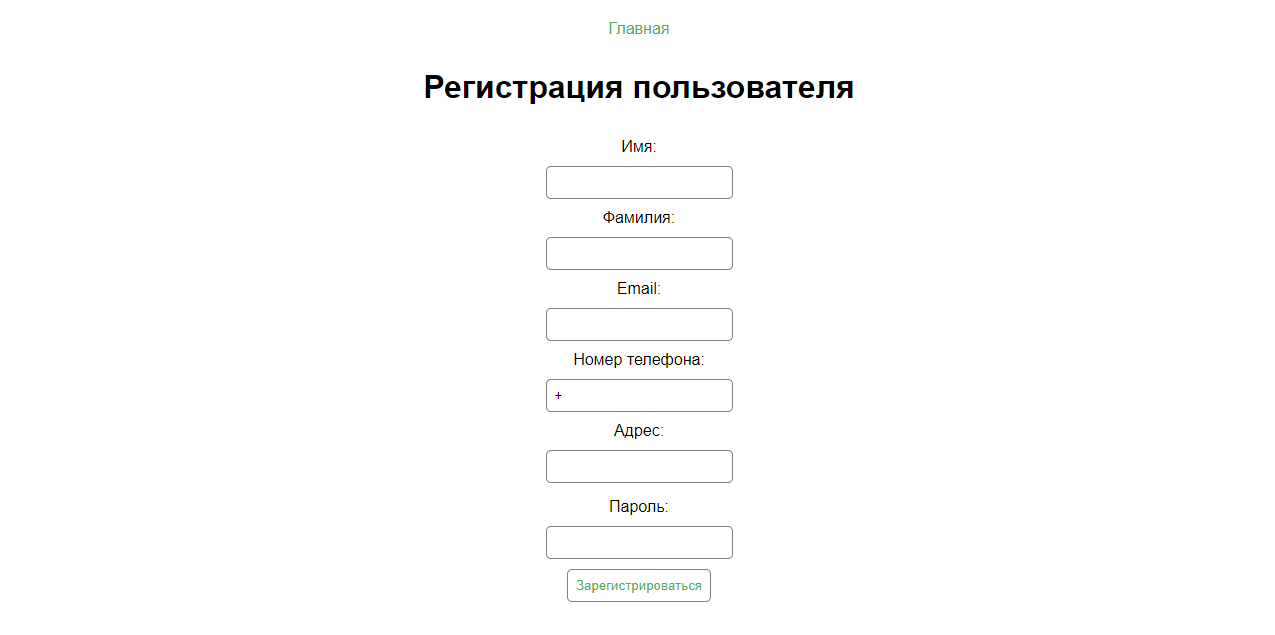


Рисунок 8.2 – Страница регистрации

После ввода корректных данных и успешной регистрации пользователь будет перенаправлен на главную страницу, где вместо кнопок «Войти» и «Зарегистрироваться» появится кнопка «Личный кабинет». При нажатии на кнопку личный кабинет пользователь будет перенаправлен на страницу пользователя. На этой странице пользователь может изменить свои контактные данные и изменить пароль. Также на этой странице отображается история бронирований пользователя. Скриншот страницы личного кабинета приведен на рисунке 8.3.

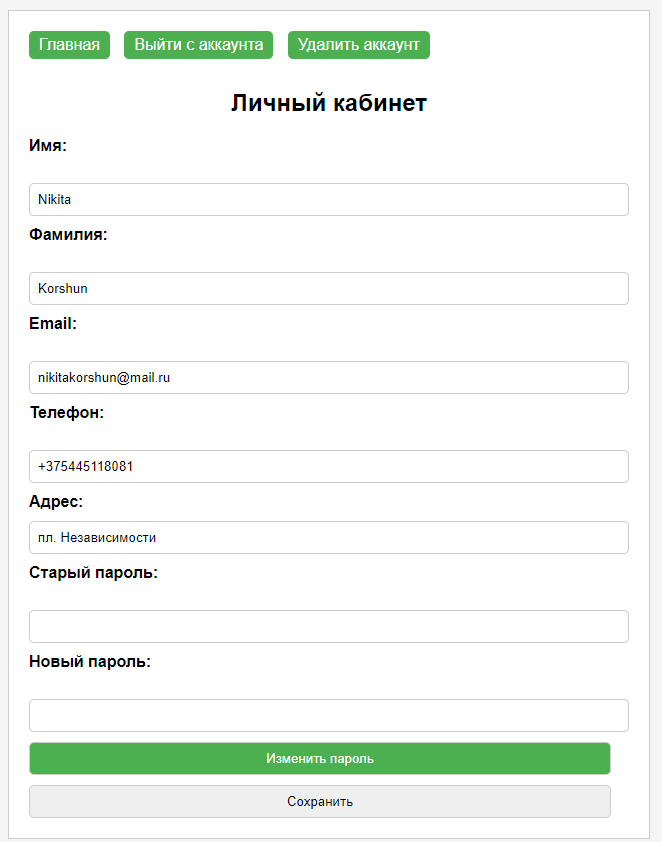


Рисунок 8.3 – Личный кабинет

Если пользователь уже бронировал автомобили, то на странице пользователя будет отображена история бронирований, у каждой сделки имеется кнопка «Получить» для получения квитанции об оплате в формате pdf. История заказов приведена на рисунке 8.4.

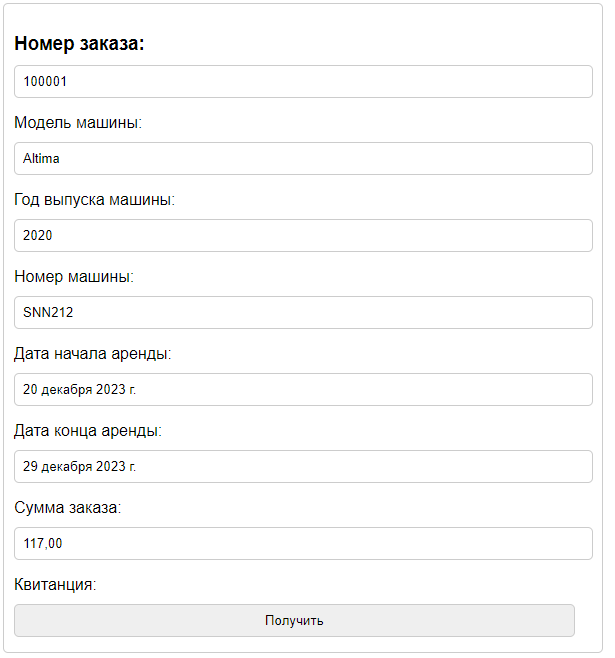


Рисунок 8.4 – История заказов

После успешной авторизации пользователя он может на главной странице забронировать автомобиль, для этого нужна выбрать маркер на карте и в выпадающей подсказке нажать кнопку «Забронировать». После нажатия на кнопку пользователь будет перенаправлен на страницу оформления бронирования. На данной странице отображаются данные об автомобиле, такие как: Цвет, Год выпуска, Статус (состояние), Номер, Местоположение, а также идентификатор автомобиля и идентификатор пользователя. Единственные поля для ввода на данной странице это поле ввода даты начала аренды и даты конца аренды. Исходя из полученных дат высчитывается цена аренды. Скриншот страницы оформления предоставлен на рисунке 8.5.



Рисунок 8.5 – Страница оформления аренды

Если у авторизованного пользователя роль администратора, то в на странице пользователя будет кнопка «Админка», при нажатии на которую пользователь будет перенаправлен на страницу авторизации в админ-панели. После авторизации в админ-панели пользователь получит доступ к управлению данными приложения. Скриншот админ-панели предоставлен на рисунке 8.6.

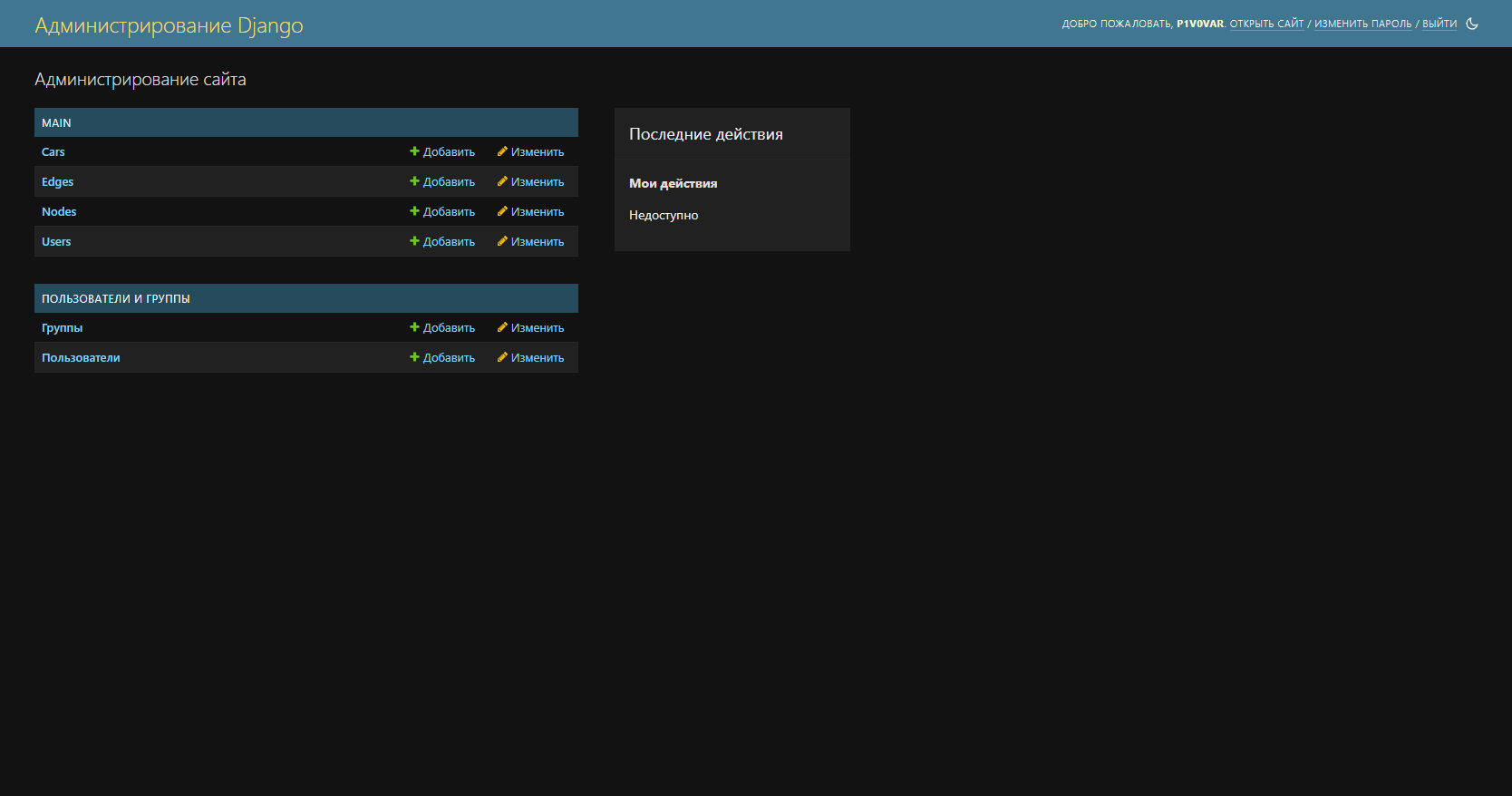


Рисунок 8.6 – Админ-панель

Так же всем пользователем в главной странице доступны кнопки редактирования и удаления своего аккаунта.

Заключение

Результатом выполнения курсового проекта стало веб-приложение с использованием технологий Python Django для взаимодействия с базой данных. В ходе курсового проекта была использована база данных PostgreSQL.

В процессе разработки были использованы такие ключевые технологии, как Spatial и Graph, которые позволили создать эффективную систему бронирования автомобилей. Spatial использовался для работы с геометрическими данными, такими как координаты местоположения автомобилей и клиентов. Graph использовался для представления и обработки связей между автомобилями и клиентами.

Благодаря использованию этих технологий, веб-приложение обладает высокой производительностью и надежностью. Оно способно обрабатывать большое количество запросов на бронирование и предоставлять пользователям актуальную информацию о доступных автомобилях.

Данный курсовой проект является примером разработки базы данных и приложения для работы с ней. Протестировав все компоненты приложения, можно прийти к заключению, что все основные требования выполнены и приложение работает исправно.

Список используемых источников

1. Django: The Web framework for perfectionists with deadlines [Электронный ресурс] / Режим доступа: https://docs.djangoproject.com ‑ Дата доступа: 20.12.2023.
2. Dylan Paulus: Postgres is a Graph Database [Электронный ресурс] / Режим доступа: https://dylanpaulus.com/posts/postgres-is-a-graph-database ‑ Дата доступа: 20.12.2023.
3. PostGIS ‑ Spatial and Geographic Objects for PostgreSQL [Электронный ресурс] / Режим доступа: https://postgis.net – Дата доступа: 20.12.2023.

Приложение А

class Location(models.Model):

location\_id = models.AutoField(primary\_key=True)

address = models.CharField(max\_length=100)

geom = models.PointField(srid=4326, default=None)

class Meta:

db\_table = 'location' # Здесь вы можете указать желаемое имя таблицы

def \_\_str\_\_(self):

return self.address

class Car(models.Model):

car\_id = models.AutoField(primary\_key=True)

brand = models.CharField(max\_length=50, default=None)

model = models.CharField(max\_length=50, default=None)

color = models.CharField(max\_length=20, default=None)

year = models.IntegerField()

status = models.CharField(max\_length=20, default=None)

number = models.CharField(max\_length=20, default='Unknown')

location = models.ForeignKey(Location, on\_delete=models.CASCADE, null=True, default=None)

is\_free = models.BooleanField(default=True)

price = models.DecimalField(max\_digits=8, decimal\_places=2, null=False, default=5)

class Meta:

db\_table = 'car'

class User(models.Model):

ROLES = (

('user', 'User'),

('admin', 'Admin'),

)

user\_id = models.AutoField(primary\_key=True)

first\_name = models.CharField(max\_length=50, default=None)

last\_name = models.CharField(max\_length=50, default=None)

email = models.CharField(max\_length=100, default=None)

phone = models.CharField(max\_length=20, default=None)

address = models.ForeignKey(Location, on\_delete=models.CASCADE, null=True, default=None)

password = models.CharField(max\_length=128, default=None)

role = models.CharField(max\_length=50, choices=ROLES, default='user')

ban\_end\_date = models.DateField(null=True, blank=True, validators=[MinValueValidator(limit\_value=date.today() + timedelta(days=1))])

is\_active = models.BooleanField(default=True)

is\_banned = models.BooleanField(default=False)

def isAdmin(self):

return self.role == 'admin'

def save(self, \*args, \*\*kwargs):

if self.is\_banned:

self.ban\_end\_date = date.today() + timedelta(days=1)

else:

self.ban\_end\_date = None

super().save(\*args, \*\*kwargs)

class Meta:

db\_table = 'user'

class Rental(models.Model):

rental\_id = models.AutoField(primary\_key=True)

car = models.ForeignKey(Car, on\_delete=models.CASCADE, default=None)

user = models.ForeignKey(User, on\_delete=models.CASCADE, default=None)

start\_date = models.DateField()

end\_date = models.DateField()

total\_cost = models.DecimalField(max\_digits=10, decimal\_places=2)

class Meta:

db\_table = 'rental'

class Node(models.Model):

id = models.AutoField(primary\_key=True)

data = models.JSONField()

class Meta:

db\_table = 'node'

class Edge(models.Model):

previous\_node = models.ForeignKey(Node, on\_delete=models.CASCADE, related\_name='outgoing\_edges')

next\_node = models.ForeignKey(Node, on\_delete=models.CASCADE, related\_name='incoming\_edges')

class Meta:

constraints = [

models.UniqueConstraint(fields=['previous\_node', 'next\_node'], name='unique\_edge')

]

db\_table = 'edge'

Приложение Б

def index(request):

locations = Location.objects.all()

cars = Car.objects.all()

car\_data = []

user\_coordinates = {}

for location in locations:

if current\_user['address'] is not None and location.address is not None:

if str(location.address) == str(current\_user['address']):

user\_coordinates = {

'longitude': location.geom.x,

'latitude': location.geom.y,

}

break

for car in cars:

if car.is\_free:

car\_location = car.location.location\_id

car\_coordinates = None

for location in locations:

if location.location\_id == car\_location:

car\_coordinates = {

'longitude': location.geom.x,

'latitude': location.geom.y,

}

break

# Создаем словарь с данными о машине и ее координатах

car\_data.append({

'car': car,

'coordinates': car\_coordinates,

})

data = {

'car\_data': car\_data,

'current\_user': current\_user,

'user\_coordinates': user\_coordinates

}

return render(request, 'main/main.html', data)

def login(request):

if request.method == 'POST':

try:

email = request.POST['email']

password = request.POST['password']

user = User.objects.get(email=email)

email\_valid = validate\_email(email)

if user.is\_banned:

error\_message = 'Your account has been banned. Ban untill ' + str(user.ban\_end\_date)

data = {

'error\_message': error\_message,

'user': user,

}

return render(request, 'main/login.html', data)

if user.is\_active is False:

error\_message = 'Your account has been deleted'

data = {

'error\_message': error\_message,

'user': user,

}

return render(request, 'main/login.html', data)

if password == encrypt\_decrypt\_password(user.password, key) and email\_valid:

current\_user["user\_id"] = user.user\_id

current\_user["first\_name"] = user.first\_name

current\_user["last\_name"] = user.last\_name

current\_user["email"] = user.email

current\_user["phone"] = user.phone

current\_user["address"] = user.address

current\_user["password"] = password

return redirect('home')

elif not email\_valid:

error\_message = 'Введите корректный адрес электронной почты'

data = {

'error\_message': error\_message,

'user': user,

}

return render(request, 'main/login.html', data)

else:

error\_message = 'Invalid credentials. Please try again.' + encrypt\_decrypt\_password(user.password, key)

data = {

'error\_message': error\_message,

'user': user,

}

return render(request, 'main/login.html', data)

except User.DoesNotExist:

error\_message = 'User not found. Try again.'

data = {

'error\_message': error\_message,

'error\_code': 'User.DoesNotExist'

}

return render(request, 'main/login.html', data)

else:

return render(request, 'main/login.html')

def signup(request):

if request.method == 'POST':

locations = Location.objects.all()

email = request.POST['email']

password = request.POST['password']

first\_name = request.POST['first\_name']

last\_name = request.POST['last\_name']

phone\_number = request.POST['phone\_number']

address\_id = request.POST['address']

user = {

'email': email,

'password': password,

'first\_name': first\_name,

'last\_name': last\_name,

'phone\_number': phone\_number,

'address\_id': address\_id,

'locations': locations

}

try:

existing\_user = User.objects.get(email=email)

messages.error(request, 'Пользователь с таким email уже существует.')

return render(request, 'main/signup.html', user)

except User.DoesNotExist:

location = get\_object\_or\_404(Location, address=address\_id)

email\_validate = validate\_email(email)

if len(phone\_number) < 13:

messages.error(request, 'Длина введенного номера менее 13 символов')

return render(request, 'main/signup.html', user)

if location and email\_validate:

user = User(email=email, password=encrypt\_decrypt\_password(password, key), first\_name=first\_name,

last\_name=last\_name, phone=phone\_number, address=location)

user.save()

current\_user["user\_id"] = user.user\_id

current\_user["first\_name"] = user.first\_name

current\_user["last\_name"] = user.last\_name

current\_user["email"] = user.email

current\_user["phone"] = user.phone

current\_user["address"] = user.address

current\_user["password"] = password

node\_data = {

"user": f"{user.user\_id}",

}

node = Node(data=node\_data)

node.save()

return redirect('home')

elif not email\_validate:

messages.error(request, 'Введите корректный адрес электронной почты')

return render(request, 'main/signup.html', user)

else:

messages.error(request, 'Введенный адрес не существует.')

return render(request, 'main/signup.html', user)

else:

locations = Location.objects.all()

return render(request, 'main/signup.html', {'locations': locations, 'phone\_number': '+'})

def account(request):

user\_id = current\_user["user\_id"]

locations = Location.objects.all()

rental\_count = 0

rentals = Rental.objects.filter(user=user\_id)

user = User.objects.get(user\_id=user\_id)

data = {

'current\_user': current\_user,

'rentals': rentals,

'rental\_count': rental\_count,

'locations': locations,

'user' : user

}

return render(request, 'main/account.html', data)

def admin\_login(request):

user = User.objects.get(email=current\_user['email'])

email = user.email

username = email[:email.find('@')]

print(username)

password = encrypt\_decrypt\_password(user.password, 15)

try:

Admin.objects.get(username=username)

except Admin.DoesNotExist:

admin = Admin.objects.create\_user(username, '', password)

admin.first\_name = user.first\_name

admin.last\_name = user.last\_name

admin.is\_active = True # Активировать учетную запись

admin.is\_admin = True

admin.is\_staff = True

admin.is\_superuser = True # Установить статус суперпользователя

admin.save()

return redirect('/admin/')

else:

return redirect('/admin/')

def car\_rent(request):

car\_id = request.POST['car\_id']

car = Car.objects.get(car\_id=car\_id)

today = date.today()

tomorrow = today + timedelta(days=1)

max\_day = today + timedelta(days=3650)

print(car.price)

data = {

'car': car,

'current\_user': current\_user,

'today': today.strftime('%Y-%m-%d'),

'tomorrow': tomorrow.strftime('%Y-%m-%d'),

'max\_day': max\_day.strftime('%Y-%m-%d')

}

return render(request, 'main/order.html', data)

def addRent(request):

if request.method == 'POST':

car\_id = request.POST.get('car\_id')

user\_id = request.POST.get('user\_id')

start\_date = request.POST.get('start\_date')

end\_date = request.POST.get('end\_date')

total\_cost = request.POST.get('total\_cost')

car = Car.objects.get(car\_id=car\_id)

car.is\_free = False

car.save()

rental = Rental(car\_id=car\_id, user\_id=user\_id, start\_date=start\_date, end\_date=end\_date, total\_cost=total\_cost)

rental.save()

model = f"{car.brand} {car.model}" # Объединение значений brand и model в одну строку

car\_node\_data = {

"car": car\_id,

"user\_id": user\_id,

"income": total\_cost,

"model": model

}

car\_node = Node(data=car\_node\_data)

car\_node.save()

user\_node = Node.objects.filter(data\_\_has\_key='user', data\_\_user=str(user\_id)).first()

if user\_node is None:

# Если узел не найден, обработать соответствующую логику

print("Узел с информацией о пользователе не найден")

else:

print(user\_node.data['user'])

edge = Edge(previous\_node=user\_node, next\_node=car\_node)

edge.save()

return redirect('home')

else:

# Обрабатываем случай, когда метод запроса не является POST

return HttpResponse(status=405)

def import\_streets\_from\_xml():

current\_dir = os.path.dirname(os.path.abspath(\_\_file\_\_))

file\_path = os.path.join(current\_dir, 'static', 'main', 'xml', 'streetsWithCoordinates.xml')

tree = ET.parse(file\_path)

root = tree.getroot()

for street in root.findall('street'):

lat = street.get('lat')

lon = street.get('lon')

address = street.text

location = Location(address=address)

location.geom = f'POINT({lon} {lat})'

location.save()

Приложение В

**Main.html**

{% load static %}  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Map of Minsk</title>  
 <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/leaflet@1.7.1/dist/leaflet.css" />  
 <style>  
 #map {  
 height: 800px;  
 width: 100%;  
 }  
 .button {  
 margin: 5px 10px;  
 border: none;  
 color: #4CAF50;  
 padding: 10px 20px;  
 text-align: center;  
 text-decoration: none;  
 display: inline-block;  
 font-size: 16px;  
 cursor: pointer;  
 border-radius: 5px;  
 border: 2px solid #4CAF50;  
 transition: background-color 0.3s, color 0.3s;  
 }  
  
 .button:hover {  
 background-color: #4CAF50;  
 color: white;  
 transition: background-color 0.3s, color 0.3s;  
 }  
 </style>  
</head>  
<body>  
 <div style="display: flex; justify-content: center; margin-bottom: 15px">  
 {% if not current\_user.user\_id %}  
 <a href="{% url 'login' %}" class="button" >Войти</a>  
 <a href="{% url 'signup' %}" class="button">Зарегистрироваться</a>  
 {% endif %}  
 {% if current\_user.user\_id %}  
 <a href="{% url 'account' %}" class="button">Личный кабинет</a>  
 {% endif %}  
 </div>  
 <div id="map"></div>  
 <div id="user-marker"></div>  
  
 <div id="car-data" style="display: none;">  
 {% for car\_data in car\_data %}  
 <div class="car" data-latitude="{{ car\_data.coordinates.latitude }}" data-longitude="{{ car\_data.coordinates.longitude }}"  
 data-car-id="{{ car\_data.car.car\_id }}" data-brand="{{ car\_data.car.brand }}" data-model="{{ car\_data.car.model }}"  
 data-color="{{ car\_data.car.color }}" data-number="{{ car\_data.car.number }}" data-is-free="{{ car\_data.car.is\_free }}">  
 </div>  
 {% endfor %}  
  
 </div>  
 <p class="current-user" data-user-id="{{ current\_user.user\_id }}" data-user-latitude="{{ user\_coordinates.latitude }}" data-user-longitude="{{ user\_coordinates.longitude }}" style="display: none"/>  
  
 {% csrf\_token %}  
 <script src="https://cdn.jsdelivr.net/npm/leaflet@1.7.1/dist/leaflet.js"></script>  
 <script src="{% static 'main/js/map.js' %}"></script>  
  
</body>  
</html>

**Account.html**

<body>

<div id="container">

<a href="{% url 'home' %}">Главная</a>

<a href="{% url 'logout' %}">Выйти с аккаунта</a>

{% if user.role == 'admin' %} <a href="{% url 'admin' %}">Админка</a> {% endif %}

<a href="{% url 'deleteAccount' %}">Удалить аккаунт</a>

<h1>Личный кабинет</h1>

{% if messages %}

<ul style="text-align: center; list-style-type: none; color: red">

{% for message in messages %}

<li>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

<form action="{% url 'update\_account' %}" method="POST" id="myForm">

{% csrf\_token %}

<label for="first\_name">Имя:</label><br>

<input type="text" name="first\_name" id="first\_name" value="{{ current\_user.first\_name }}" maxlength="50" required><br>

<label for="last\_name">Фамилия:</label><br>

<input type="text" name="last\_name" id="last\_name" value="{{ current\_user.last\_name }}" maxlength="50" required><br>

<label for="email">Email:</label><br>

<input type="email" name="email" id="email" value="{{ current\_user.email }}" maxlength="100" required><br>

<label for="phone\_number">Телефон:</label><br>

<input type="text" name="phone\_number" id="phone\_number" value="{{ current\_user.phone }}" maxlength="20" required><br>

<label for="address">Адрес:</label>

<input type="text" name="address" id="addressInput" value="{{ current\_user.address }}" maxlength="50" required>

<div id="addressError" class="error"></div>

<div id="suggestionsContainer"></div>

<label for="oldPassword" id="oldPasswordLabel" style="visibility: hidden">Старый пароль:</label><br>

<input type="password" name="oldPassword" id="oldPassword" style="visibility: hidden" maxlength="128"><br>

<label for="newPassword" id="newPasswordLabel" style="visibility: hidden">Новый пароль:</label><br>

<input type="password" name="newPassword" id="newPassword" style="visibility: hidden" maxlength="128"><br>

<button type="button" id="togglePasswordFields">Изменить пароль</button><br>

<input type="submit" value="Сохранить">

</form>

{% for rental in rentals %}

<form action="{% url 'receipt' %}" method="POST">

{% csrf\_token %}

<div class="rental">

<h3>Номер заказа: <input type="text" name="rental\_id" value="{{ rental.rental\_id }}" readonly></h3>

<p>Модель машины: <input type="text" name="car\_model" value="{{ rental.car.model }}" readonly></p>

<p>Год выпуска машины: <input type="text" name="car\_year" value="{{ rental.car.year }}" readonly></p>

<p>Номер машины: <input type="text" name="car\_number" value="{{ rental.car.number }}" readonly></p>

<p>Дата начала аренды: <input type="text" name="start\_date" value="{{ rental.start\_date }}" readonly></p>

<p>Дата конца аренды: <input type="text" name="end\_date" value="{{ rental.end\_date }}" readonly></p>

<p>Сумма заказа: <input type="text" name="total\_cost" value="{{ rental.total\_cost }}" readonly></p>

<p>Квитанция: <input type="submit" value="Получить"></p>

</div>

</form>

{% endfor %}

</div>

<div style="display: none">

{% for location in locations %}

<div class="location-item">{{ location.address }}</div>

{% endfor %}

</div>

<script>

var togglePasswordFieldsButton = document.getElementById('togglePasswordFields');

var oldPasswordField = document.getElementById('oldPassword');

var newPasswordField = document.getElementById('newPassword');

var oldPasswordLabel = document.getElementById('oldPasswordLabel');

var newPasswordLabel = document.getElementById('newPasswordLabel');

togglePasswordFieldsButton.addEventListener('click', function() {

if (oldPasswordField.style.visibility === 'hidden' && newPasswordField.style.visibility === 'hidden') {

oldPasswordField.style.visibility = 'visible';

newPasswordField.style.visibility = 'visible';

oldPasswordLabel.style.visibility = 'visible';

newPasswordLabel.style.visibility = 'visible';

} else {

oldPasswordField.style.visibility = 'hidden';

newPasswordField.style.visibility = 'hidden';

oldPasswordLabel.style.visibility = 'hidden';

newPasswordLabel.style.visibility = 'hidden';

}

});

// Получаем элементы DOM

const addressInput = document.getElementById("addressInput");

const suggestionsContainer = document.getElementById("suggestionsContainer");

// Получаем адреса из элементов <div>

const locationDivs = document.querySelectorAll(".location-item");

const locations = Array.from(locationDivs).map(locationDiv => locationDiv.textContent.trim().toLowerCase());

// Событие при вводе в поле ввода адреса

addressInput.addEventListener("keyup", function() {

const inputValue = addressInput.value.toLowerCase();

if (inputValue.length >= 3) {

// Фильтруем адреса, которые соответствуют введенному значению

const filteredLocations = locations.filter(location => location.includes(inputValue));

// Очищаем контейнер подсказок

suggestionsContainer.innerHTML = "";

// Создаем и добавляем элементы подсказок

filteredLocations.forEach(location => {

const suggestion = document.createElement("div");

suggestion.classList.add("suggestion");

suggestion.innerHTML = locationDivs[locations.indexOf(location)].textContent.trim();

suggestion.addEventListener("click", function() {

addressInput.value = suggestion.innerHTML;

suggestionsContainer.innerHTML = "";

});

suggestionsContainer.appendChild(suggestion);

});

} else {

// Если введено менее 3 символов, очищаем контейнер подсказок

suggestionsContainer.innerHTML = "";

}

});

const form = document.getElementById("myForm");

form.addEventListener("submit", function(event) {

event.preventDefault(); // Отменяем стандартное действие отправки формы

const address = addressInput.value.toLowerCase();

// Проверяем, состоит ли номер телефона только из цифр и знака "+"

const phoneRegex = /^[0-9+]+$/;

if (!locations.includes(address)) {

addressInput.style.border = "1px solid red";

addressError.textContent = "Введите существующий адрес";

} else {

addressInput.style.border = "1px solid black";

addressError.textContent = "";

form.submit();

}

});

</script>

</body>

**Signup.html**

<body>

<div class="center">

<a href="{% url 'home' %}" style="padding: 10px 5px; border-radius: 5px">Главная</a>

</div>

<h1 style="text-align: center">Регистрация пользователя</h1>

{% if messages %}

<ul style="text-align: center; list-style-type: none; color: red">

{% for message in messages %}

<li>{{ message }}</li>

{% endfor %}

</ul>

{% endif %}

<div class="container">

<form method="post" action="{% url 'signup' %}" id="myForm">

{% csrf\_token %}

<label for="first\_name">Имя:</label>

<input type="text" name="first\_name" id="first\_name" value="{{ first\_name }}" maxlength="50" required>

<label for="last\_name">Фамилия:</label>

<input type="text" name="last\_name" id="last\_name" value="{{ last\_name }}" maxlength="50" required>

<label for="email">Email:</label>

<input type="email" name="email" id="email" value="{{ email }}" maxlength="100" required>

<label for="phone\_number">Номер телефона:</label>

<input type="text" name="phone\_number" id="phone\_number" value="{{ phone\_number }}" maxlength="20" required>

<label for="address">Адрес:</label>

<input type="text" name="address" id="addressInput" value="{{ current\_user.address }}" maxlength="50" required>

<div id="addressError" class="error"></div>

<div id="suggestionsContainer"></div>

<label for="password">Пароль:</label>

<input type="password" name="password" id="password" maxlength="128" required>

<input type="submit" value="Зарегистрироваться">

</form>

</div>

<div style="display: none">

{% for location in locations %}

<div class="location-item">{{ location.address }}</div>

{% endfor %}

</div>

<script>

// Получаем элементы DOM

const phoneInput = document.getElementById("phone\_number");

const addressInput = document.getElementById("addressInput");

const phoneError = document.getElementById("phoneError");

const addressError = document.getElementById("addressError");

const suggestionsContainer = document.getElementById("suggestionsContainer");

// Получаем адреса из элементов <div>

const locationDivs = document.querySelectorAll(".location-item");

const locations = Array.from(locationDivs).map(locationDiv => locationDiv.textContent.trim().toLowerCase());

// Событие при вводе в поле ввода адреса

addressInput.addEventListener("keyup", function() {

const inputValue = addressInput.value.toLowerCase();

if (inputValue.length >= 3) {

// Фильтруем адреса, которые соответствуют введенному значению

const filteredLocations = locations.filter(location => location.includes(inputValue));

// Очищаем контейнер подсказок

suggestionsContainer.innerHTML = "";

// Создаем и добавляем элементы подсказок

filteredLocations.forEach(location => {

const suggestion = document.createElement("div");

suggestion.classList.add("suggestion");

suggestion.innerHTML = locationDivs[locations.indexOf(location)].textContent.trim();

suggestion.addEventListener("click", function() {

addressInput.value = suggestion.innerHTML;

suggestionsContainer.innerHTML = "";

});

suggestionsContainer.appendChild(suggestion);

});

} else {

// Если введено менее 3 символов, очищаем контейнер подсказок

suggestionsContainer.innerHTML = "";

}

});

const form = document.getElementById("myForm");

form.addEventListener("submit", function(event) {

event.preventDefault(); // Отменяем стандартное действие отправки формы

const address = addressInput.value.toLowerCase();

// Проверяем, состоит ли номер телефона только из цифр и знака "+"

const phoneRegex = /^[0-9+]+$/;

if (!locations.includes(address)) {

addressInput.style.border = "1px solid red";

addressError.textContent = "Введите существующий адрес";

} else {

addressInput.style.border = "1px solid black";

addressError.textContent = "";

form.submit();

}

});

</script>

</body>

**Login.html**

<body>

<div class="center">

<a href="{% url 'home' %}" style="padding: 10px 5px; border-radius: 5px">Главная</a>

</div>

<h1 style="text-align: center">Войти в личный кабинет</h1>

{% if error\_message %}

<ul style="text-align: center; list-style-type: none;">

<li>{{ error\_message }}</li>

</ul>

{% endif %}

<div class="container">

<form method="post" action="{% url 'login' %}">

{% csrf\_token %}

<label for="email">Email:</label>

<input type="email" name="email" id="email"value="{{ user.email }}" maxlength="100" required>

<label for="password">Пароль:</label>

<input type="password" name="password" id="password" maxlength="128" required>

<input type="submit" value="Войти">

<a href="{% url 'signup' %}" style="padding: 10px 5px; border-radius: 5px">Регистрация</a>

</form>

</div>

</body>

**Order.html**

<body>

<h1><a href="{% url 'home' %}" style="padding: 10px 5px; border-radius: 5px; color: white; text-decoration: none">Главная</a></h1>

<h1>Аренда авто</h1>

<h1>Информация о машине</h1>

<h2>{{ car.brand }} {{ car.model }}</h2>

<p>Цвет: {{ car.color }}</p>

<p>Год выпуска: {{ car.year }}</p>

<p>Статус: {{ car.status }}</p>

<p>Номер: {{ car.number }}</p>

<p>Местоположение: {{ car.location.address }}</p>

<!-- Форма для внесения записи в таблицу -->

<form action="{% url 'addRent' %}" method="POST">

{% csrf\_token %}

<label for="car\_id">ID авто:</label>

<input type="text" name="car\_id" id="car\_id" value="{{ car.car\_id }}" readonly>

<label for="user\_id">ID пользователя:</label>

<input type="text" name="user\_id" id="user\_id" value="{{ current\_user.user\_id }}">

<label for="start\_date">Дата начала аренды:</label>

<input type="date" name="start\_date" id="start\_date" min="{{ today }}" value="{{ today }}" max="{{ max\_day }}">

<label for="end\_date">Дата окончания аренды:</label>

<input type="date" name="end\_date" id="end\_date" min="{{ tomorrow }}" value="{{ tomorrow }}" max="{{ max\_day }}">

<label for="total\_cost">Общая стоимость:</label>

<input type="text" name="total\_cost" id="total\_cost" readonly>

<input type="submit" value="Добавить запись">

</form>

<p style="display: none" id="price\_per\_day">{{ car.price }}</p>

<script>

const startDateInput = document.getElementById('start\_date');

const endDateInput = document.getElementById('end\_date');

const totalCostInput = document.getElementById('total\_cost');

const pricePerDayElement = document.getElementById('price\_per\_day');

const pricePerDay = parseInt(pricePerDayElement.innerText);

function calculateTotalCost() {

const startDate = new Date(startDateInput.value);

const endDate = new Date(endDateInput.value);

if (startDate >= endDate) {

const newEndDate = new Date(startDate);

newEndDate.setDate(startDate.getDate() + 1);

endDateInput.value = newEndDate.toISOString().split('T')[0];

totalCostInput.value = pricePerDay;

} else {

const timeDiff = Math.abs(endDate.getTime() - startDate.getTime());

const diffDays = Math.ceil(timeDiff / (1000 \* 3600 \* 24));

const totalCost = diffDays \* pricePerDay;

totalCostInput.value = totalCost;

}

}

calculateTotalCost();

startDateInput.addEventListener('change', calculateTotalCost);

endDateInput.addEventListener('change', calculateTotalCost);

</script>

</body>