

**Министерство науки и высшего образования Российской
Федерации**

федеральное государственное автономное

образовательное учреждение высшего образования

«Самарский национальный исследовательский университет

имени академика С.П. Королева»

Институт информатики и кибернетики

Кафедра технической кибернетики

Финальный отчёт

Дисциплина: «Технологии сетевого программирования»

Финализация приложения «Аренда автомобилей»

Выполнил: Бондарев Иван, Де Лео Дэвид

Группа: 6303-010302D

Самара, 2025

Описание проекта

Этот проект представляет собой веб-приложение для аренды автомобилей. Пользователи могут просматривать доступные автомобили, фильтровать их по характеристикам и оформлять бронирование.

Функционал

- Регистрация и авторизация пользователей
- Просмотр списка автомобилей с фильтрацией по параметрам
- Оформление бронирований
- Оплата аренды онлайн

Используемые технологии

Backend: Python — основной язык разработки; Django — веб-фреймворк для построения серверной части и API; Django REST Framework — для создания RESTful API;

База данных: PostgreSQL — основная реляционная база данных;

Frontend: HTML + Django Templates (Jinja2) — для создания пользовательского интерфейса и отображения страниц; CSS (Bootstrap) — для стилизации.

Контейнеризация: Docker — для упаковки приложения и базы данных; Docker Compose — для управления многоконтейнерной архитектурой.

Тестирование и отладка: Postman — для ручного тестирования API; Faker — генерация тестовых данных (пользователи, машины и пр.); Swagger — для документации API.

1. Проектирование приложения

1) Проектирование архитектуры

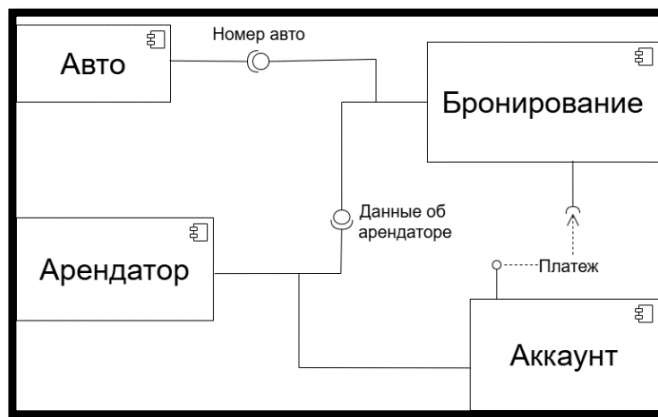
Проект — веб-приложение для аренды автомобилей, которое позволяет пользователям найти и арендовать интересный автомобиль через удобный веб-интерфейс. Приложение объединяет компанию, предоставляющих машины в аренду, и пользователей, которым нужен

автомобиль на какой-то срок. Вся работа проходит через онлайн-платформу, где можно выбрать авто, оформить бронирование и оплатить аренду.

Логика работы:

- пользователь проходит верификацию: регистрируется или входит в аккаунт;
- выбирает автомобиль;
- оформляет бронирование, выбирая даты аренды;
- оплачивает аренду через «встроенную систему платежей».

Схема взаимодействия компонентов:



Каждый прямоугольник на диаграмме представляет собой компонент системы. Линии или стрелки между компонентами показывают взаимодействие между ними. Круги и полукруги указывают на то, что для взаимодействия компонентов нужен интерфейс. То есть, «авто» предоставляет интерфейс (полукруг) для бронирования, а «бронирование» требует этот интерфейс (круг) для резервирования автомобиля. Также есть круг, полукруг, стрелка между ними и надпись «платёж». Это значит, что для взаимодействия компонентов должен произойти платёж.

ER-диаграмма:

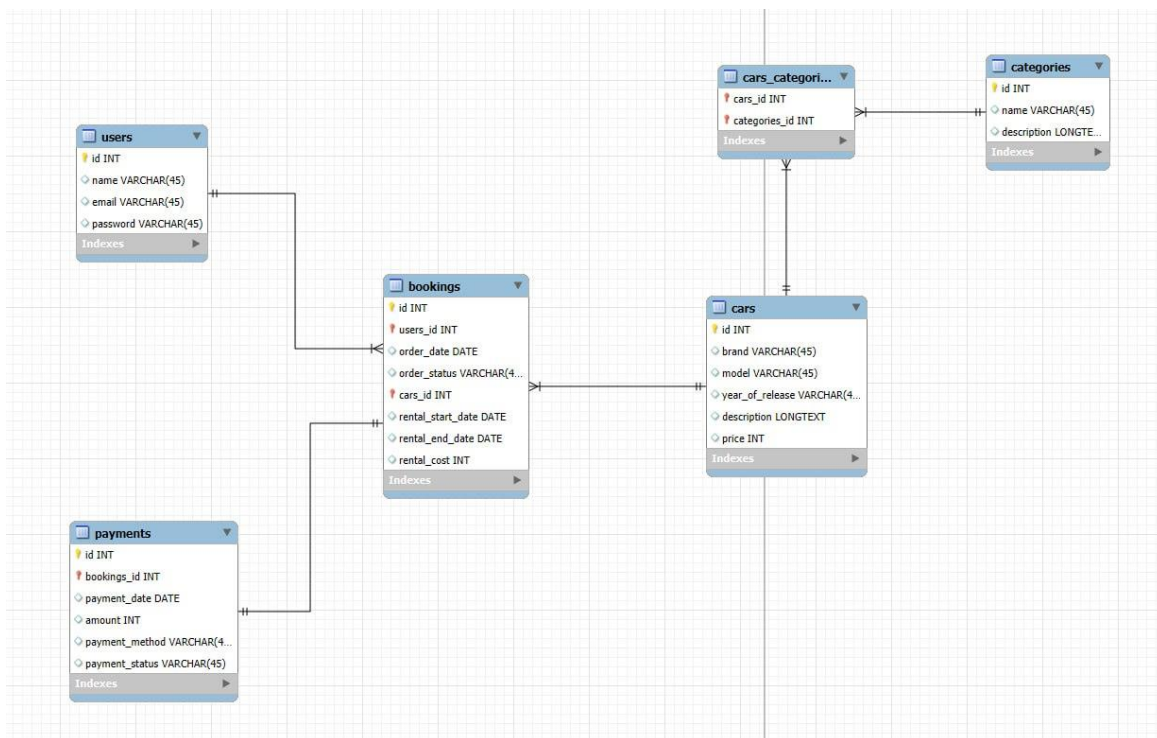


Таблица «users» хранит информацию о пользователях и связана с таблицей «bookings» по «users_id», реализуя связь один-ко-многим, так как один пользователь может иметь несколько бронирований. Таблица «cars» содержит информацию об автомобилях и связана с «bookings» по «cars_id», также через связь один-ко-многим, поскольку один автомобиль может участвовать в нескольких бронированиях. Таблица «bookings» объединяет пользователей и автомобили, храня данные о датах аренды, стоимости и статусе заказа. Она также связана с «payments», где фиксируются платежи по «bookings_id», что формирует связь один-ко-многим или один-к-одному, в зависимости от бизнес-логики (если допускаются несколько платежей за одно бронирование). Категории автомобилей хранятся в таблице «categories», а их связь с автомобилями реализуется через промежуточную таблицу «cars_categories», обеспечивая связь многие-ко-многим, так как один автомобиль может относиться к нескольким категориям, а одна категория может включать разные машины.

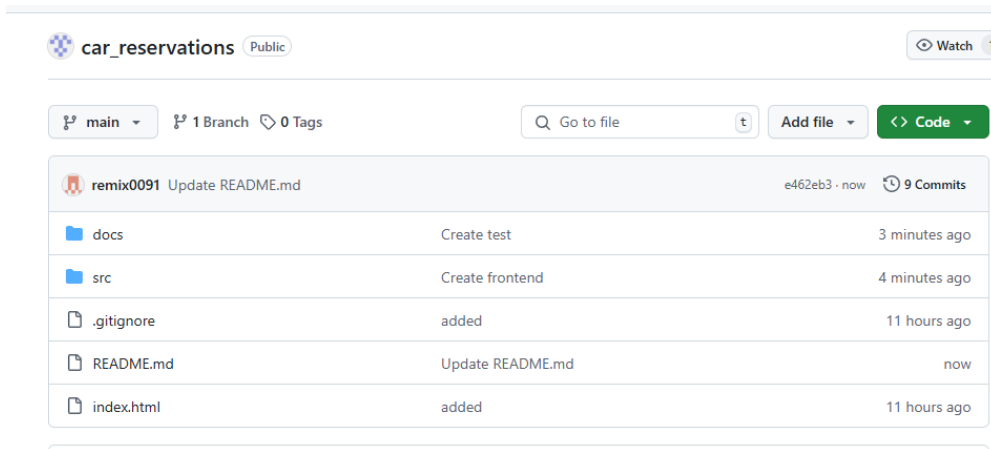
Далее мы описали **структуру API**, используя OpenAPI 3.0. API, построенный на REST-архитектуре. Доступен метод GET /cars, который позволяет получить список автомобилей. Он поддерживает параметры запроса minPrice, maxPrice (числовые значения с плавающей запятой) и bodyType (строка, указывающая тип кузова). Ответы приходят в формате JSON: при успешном выполнении

возвращается массив объектов Car, содержащих car_id (UUID), brand, model и price. В случае ошибки API возвращает объект Error с полями code (числовой код ошибки) и message (текстовое описание). Структура использует HTTP-методы, поддерживает параметры запроса и стандартные коды ответов.

<pre>openapi: 3.0.0 info: title: Аренда автомобилей version: 1.0.0 servers: - url: http://localhost:8080/api/v1/ description: Dev server paths: /cars: get: summary: Метод получения списка автомобилей tags: - Cars operationId: getAllCars parameters: - name: minPrice in: query description: Минимальная цена за сутки required: false schema: type: number format: float - name: maxPrice in: query description: Максимальная цена за сутки required: false schema: type: number format: float - name: bodyType in: query description: Тип кузова автомобиля (например, седан, универсал, хэтчбек) required: false schema: type: string responses: '200': description: Успешный ответ со списком автомобилей content: application/json: schema: \$ref: "#/components/schemas/Cars" 'default': description: Все нестандартное content: application/json: schema: \$ref: "#/components/schemas/Error"</pre>	<pre>components: schemas: Car: type: object required: - brand - model - price properties: car_id: type: string format: uuid example: "550e8400-e29b-41d4-a716-446655440000" brand: type: string example: BMW model: type: string example: M5 F90 Competition price: type: integer Cars: type: array items: \$ref: "#/components/schemas/Car" Error: type: object required: - code - message properties: code: type: integer message: type: string</pre>
--	---

2) Создание Git-репозитория

Мы выполнили настройку системы контроля версий Git и создали удаленный репозиторий на GitHub. Был создан новый репозиторий на платформе GitHub с именем «car_reservations». Выполнена локальная инициализация, после чего репозиторий был связан с удаленным. Далее мы настроили базовую архитектуру проекта: создали основные директории src/ (исходный код приложения), docs/ (документация), а также добавили файл README.md с кратким описанием проекта. На последнем шаге добавили .gitignore, в который добавили правила для игнорирования ненужных файлов.



2. Разработка базы данных

Для начала мы разработали следующие ORM-модели:

```
class CustomUserManager(BaseUserManager):
    4 usages (3 dynamic)  remix0091
    def create_user(self, email, name, password=None, **extra_fields):
        if not email:
            raise ValueError('Поле email обязательно к заполнению')
        email = self.normalize_email(email)
        user = self.model(email=email, name=name, **extra_fields)
        user.set_password(password) # хэширование пароля
        user.save(using=self._db)
        return user

    remix0091
    def create_superuser(self, email, name, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)

        return self.create_user(email, name, password, **extra_fields)
```

«CustomUserManager» - это пользовательский менеджер модели пользователей, реализующий методы создания обычных пользователей и суперпользователей.

Метод «create_user» используется для регистрации обычных пользователей, проверяет наличие почты и задаёт хэшированный пароль.

Метод «create_superuser» создаёт администратора системы, устанавливая флаги «is_staff» и «is_superuser» в «True».

Так это выглядит в БД:

id	password	last_login	is_superuser	email	name	is_active	is_staff
1	pbkdf2_sha256\$600000\$taIG	[NULL]	[]	vera2023@example.org	Колесников Филипп Жорек	[v]	[]
2	pbkdf2_sha256\$600000\$LVY	[NULL]	[]	rodion_42@example.com	Федотова Дарья Федоровн	[v]	[]
3	pbkdf2_sha256\$600000\$dEV	[NULL]	[]	djachkovnikon@example.com	Евфросиния Натановна Ро,	[v]	[]
4	pbkdf2_sha256\$600000\$6zP	[NULL]	[]	sofija_11@example.com	Осипова Тамара Семенов	[v]	[]
5	pbkdf2_sha256\$600000\$0wf	[NULL]	[]	fortunatvdokimov@exampl	Изяслав Елисеевич Аксено	[v]	[]
6	pbkdf2_sha256\$600000\$N5L	[NULL]	[]	orlovaljudmila@example.net	Авксентий Валерьевич Сид	[v]	[]
7		[NULL]	[]	florentin_david@example.ne	Дмитрий Эдгарович Лукин	[v]	[]
8	pbkdf2_sha256\$600000\$PPjr	[NULL]	[]	fff@yandex.ru	Иван Бондарев	[v]	[]
9	pbkdf2_sha256\$600000\$QXv	[NULL]	[]	rand@example.net	Давид Де Лео	[v]	[]
10	pbkdf2_sha256\$600000\$KvZ	[NULL]	[]	moiseevvladislav@example. d d		[v]	[]
11	pbkdf2_sha256\$600000\$Spl	[NULL]	[]	vorobevanina@example.net	f	[v]	[]
12	pbkdf2_sha256\$600000\$MvZ5-04-11 18:06:49.408 +0400		[v]	admin@gmail.com	Ivan	[v]	[v]
13	pbkdf2_sha256\$600000\$7Pq	[NULL]	[]	vera2024@example.org	dava	[v]	[]

```
class User(AbstractBaseUser, PermissionsMixin):
    email = models.EmailField(max_length=255, unique=True)
    name = models.CharField(max_length=45)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)

    objects = CustomUserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['name']

    @remix0091
    def str(self):
        return self.email
```

«User» - это кастомная модель пользователя, основанная на «AbstractBaseUser» и «PermissionsMixin», предназначенная для авторизации и управления доступом в системе аренды автомобилей.

```
class Category(models.Model):
    name = models.CharField(max_length=45)
    description = models.TextField()

    @remix0091
    def str(self):
        return self.name
```

«Category» — это модель, представляющая категорию автомобилей в системе аренды. Она используется для классификации транспортных средств по типам.

```

15 usages  remix0091
class Car(models.Model):
    brand = models.CharField(max_length=45)
    model = models.CharField(max_length=45)
    year_of_release = models.CharField(max_length=45)
    description = models.TextField()
    price = models.IntegerField()
    categories = models.ManyToManyField(Category)

    def __str__(self):
        return f"{self.brand} {self.model}"

```

«Car» - это модель, представляющая транспортное средство, доступное для аренды в системе. Она содержит основные характеристики автомобиля и связь с его категориями.

```

13 usages  remix0091
class Booking(models.Model):
    ORDER_STATUS_CHOICES = [
        ('в обработке', 'В обработке'),
        ('подтверждено', 'Подтверждено'),
        ('отменено', 'Отменено'),
        ('завершено', 'Завершено'),
    ]
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    car = models.ForeignKey(Car, on_delete=models.CASCADE)
    order_date = models.DateField()
    order_status = models.CharField(
        max_length=20,
        choices=ORDER_STATUS_CHOICES,
        default='в обработке' # Статус по умолчанию
    )
    rental_start_date = models.DateField()
    rental_end_date = models.DateField()
    rental_cost = models.IntegerField()

    def __str__(self):
        return f"Booking {self.id} by {self.user}"

```

«Booking» - это модель, представляющая заявку на аренду автомобиля. Она связывает пользователя с выбранным автомобилем и фиксирует информацию о дате аренды, стоимости и статусе заказа.


```

class Payment(models.Model):
    PAYMENT_STATUS_CHOICES = [
        ('не оплачено', 'Не оплачено'),
        ('оплачено', 'Оплачено'),
    ]

    PAYMENT_METHOD_CHOICES = [
        ('наличные', 'Наличные'),
        ('онлайн', 'Онлайн'),
    ]

    booking = models.ForeignKey(
        to='Booking',
        on_delete=models.CASCADE,
        related_name='payments'
    )

    payment_date = models.DateField(auto_now_add=True)
    amount = models.IntegerField(blank=True, null=True)
    payment_method = models.CharField(
        max_length=45,
        choices=PAYMENT_METHOD_CHOICES
    )
    payment_status = models.CharField(
        max_length=45,
        choices=PAYMENT_STATUS_CHOICES,
        default='uncompleted'
    )

```

«Payment» - это модель, представляющая оплату за аренду автомобиля. Она связывается с конкретной заявкой на бронирование и содержит информацию о способе и статусе оплаты.

Далее мы написали скрипт генерации тестовых данных с использованием библиотеки Faker, которая генерирует: имена, email-адреса, адреса, даты, текстовые описания, номера телефонов и т.д.

Далее представлены фрагменты кода:

```

generate_data():
    print("Генерация тестовых данных...")

    CATEGORY_NAMES = ["Эконом", "Премиум", "Седан", "Универсал", "Кроссовер", "Внедорожник", "Минивэн"]
    categories = []
    for cat_name in CATEGORY_NAMES:
        cat = Category.objects.create(
            name=cat_name,
            description=f"{cat_name} класс автомобилей"
        )
        categories.append(cat)

    users = []
    for _ in range(10):
        user = User.objects.create_user(
            email=fake.unique.email(),
            name=fake.name(),
            password="test1234"
        )
        users.append(user)

```

```

BRANDS_MODELS = {
    "Toyota": ["Camry", "Corolla", "RAV4"],
    "BMW": ["X5", "3 Series", "5 Series"],
    "Kia": ["Rio", "Sportage", "Ceed"],
    "Hyundai": ["Elantra", "Solaris", "Tucson"],
    "Mercedes": ["C-Class", "E-Class", "GLA"],
    "Volkswagen": ["Golf", "Passat", "Tiguan"],
}

cars = []
for _ in range(15):
    brand = random.choice(list(BRANDS_MODELS.keys()))
    model = random.choice(BRANDS_MODELS[brand])
    price = random.randint(a=1500, b=6000)

    car = Car.objects.create(
        brand=brand,
        model=model,
        year_of_release=str(random.randint(a=2015, b=2023)),
        description=fake.text(max_nb_chars=100),
        price=price,
    )
    car.categories.set(random.sample(categories, k=random.randint(a=1, b=2)))
    cars.append(car)

for _ in range(20):
    user = random.choice(users)
    car = random.choice(cars)
    start_date = fake.date_between(start_date='-30d', end_date='today')
    rental_days = random.randint(a=1, b=10)
    end_date = start_date + timedelta(days=rental_days)
    cost = car.price * rental_days

    booking = Booking.objects.create(
        user=user,
        car=car,
        order_date=start_date - timedelta(days=1),
        order_status=random.choice(['в обработке', 'подтверждено', 'отменено']),
        rental_start_date=start_date,
        rental_end_date=end_date,
        rental_cost=cost
    )
    bookings.append(booking)

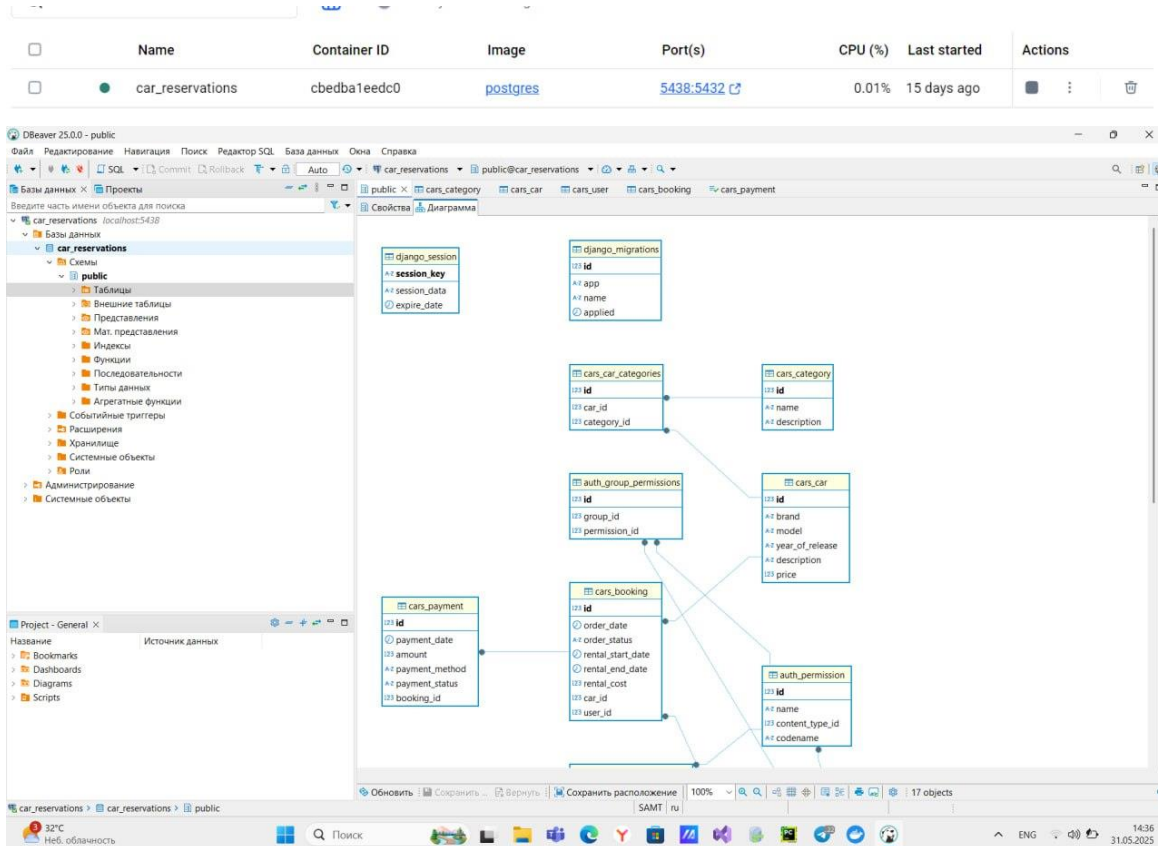
for booking in bookings:
    Payment.objects.create(
        booking=booking,
        payment_date=booking.order_date + timedelta(days=1),
        amount=booking.rental_cost,
        payment_method=random.choice(['карта', 'наличные', 'онлайн']),
        payment_status=random.choice(['оплачено', 'не оплачено'])
    )

```

Так это выглядит в БД для модели «Cars»:

	123 id	A-Z brand	A-Z model	A-Z year_of_release	A-Z description	123 price
1	27	Mercedes	E-Class	2018	Возмутиться чем спорт рабочий запу	4 572
2	28	Toyota	RAV4	2018	Монета новый выбирать ботинок рас	2 323
3	29	Mercedes	GLA	2016	Бак желание миг зато. Ребятишки вск	2 938
4	30	Toyota	Camry	2021	Передо пропаганда доставать рота л	3 963
5	31	Kia	Ceed	2023	Мрачно находить какой следователы	2 982
6	32	Kia	Ceed	2019	Скользить карман ложиться. Сынок з	2 913
7	33	BMW	3 Series	2019	Страсть соответствие идея возмутиты	4 152
8	34	Volkswagen	Golf	2016	Набор аллея банда изба трясти кома	4 805
9	35	Mercedes	C-Class	2020	Миф ставить вряд мальчишка. Пятерс	3 824
10	36	Hyundai	Tucson	2023	Палка коричневый рота хотеть преж	2 060
11	37	BMW	X5	2020	Левый светило костер наступать запе	2 766
12	38	Kia	Rio	2023	Монета тревога очередной солнце. З	4 898
13	39	Toyota	RAV4	2019	Какой поймать очередной художеств	2 065
14	40	Volkswagen	Golf	2022	Пропадать изба возбуждение число.	2 965
15	41	Kia	Rio	2016	Набор аллея банда изба трясти кома	5

БД PostgreSQL развернута в Docker, подключились мы к ней с помощью DBeaver.



3. Разработка API

Далее была разработана система RESTful API для управления данными об автомобилях. С помощью созданных эндпоинтов были реализованы основные операции: просмотр списка автомобилей, просмотр конкретного автомобиля, создание, обновление и удаление записей.

```
4
5     router = DefaultRouter()
6     router.register(r'cars', CarViewSet)
7     router.register(r'bookings', BookingViewSet)
8     router.register(r'payments', PaymentViewSet)
9     router.register(r'users', UserViewSet)
10
```

```

class UserViewSet(viewsets.ModelViewSet): 3 usages 1 idd0d0
    queryset = User.objects.all()
    serializer_class = UserSerializer

    def get_permissions(self): 1 idd0d0
        if self.action == 'list':
            return [IsAdminUser()]
        elif self.action in ['retrieve', 'update', 'partial_update']:
            return [IsAuthenticated(), IsSelfOrAdmin()]
        elif self.action == 'destroy':
            return [IsAuthenticated(), IsSelfOrAdmin()]
        return [IsAuthenticated()]

    @action(detail=False, methods=['get'], permission_classes=[IsAuthenticated]) 1 usage 1 idd0d0
    def me(self, request):
        serializer = self.get_serializer(request.user)
        return Response(serializer.data)

class CarViewSet(viewsets.ModelViewSet): 2 usages 1 idd0d0
    queryset = Car.objects.all()
    serializer_class = CarSerializer
    permission_classes = [IsAdminOrReadOnly]
    parser_classes = [JSONParser, MultiPartParser, FormParser]

class CarImageViewSet(viewsets.ModelViewSet): 2 usages 1 idd0d0
    queryset = CarImage.objects.all()
    serializer_class = CarImageSerializer
    permission_classes = [IsAdminUser]

```

```

class BookingViewSet(viewsets.ModelViewSet): 2 usages 1 idd0d0
    queryset = Booking.objects.all()
    permission_classes = [IsAuthenticated]

    def get_serializer_class(self): 1 idd0d0
        if self.request.method == 'GET':
            return BookingReadSerializer
        return BookingWriteSerializer

    def get_queryset(self): 1 idd0d0
        user = self.request.user
        if user.is_staff:
            return Booking.objects.all()
        return Booking.objects.filter(user=user)

    def perform_create(self, serializer): 1 idd0d0
        booking = serializer.save(user=self.request.user)
        # Payment.objects.create(
        #     booking=booking,
        #     amount=booking.rental_cost,
        #     payment_method='card',
        #     payment_status='uncompleted'
        # )

```

```

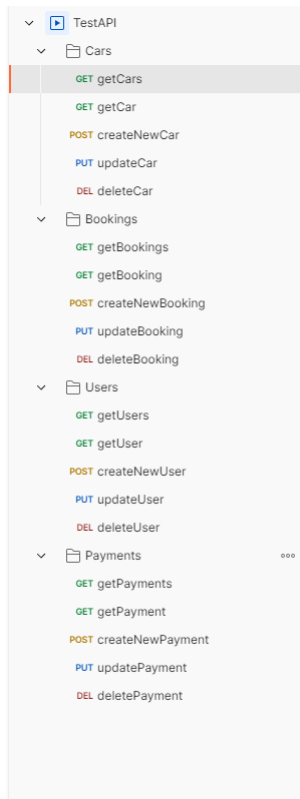
class PaymentViewSet(viewsets.ModelViewSet): 2 usages 1 idd0d0
    queryset = Payment.objects.all()
    serializer_class = PaymentSerializer
    permission_classes = [IsAuthenticated, IsOwnerCanEditOnly]

    def create(self, request, *args, **kwargs): 3 usages (3 dynamic) 1 idd0d0
        booking_id = request.data.get('booking')
        if Payment.objects.filter(booking_id=booking_id).exists():
            return Response({'detail': 'Платёж уже существует для этого бронирования'}, status=400)
        return super().create(request, *args, **kwargs)

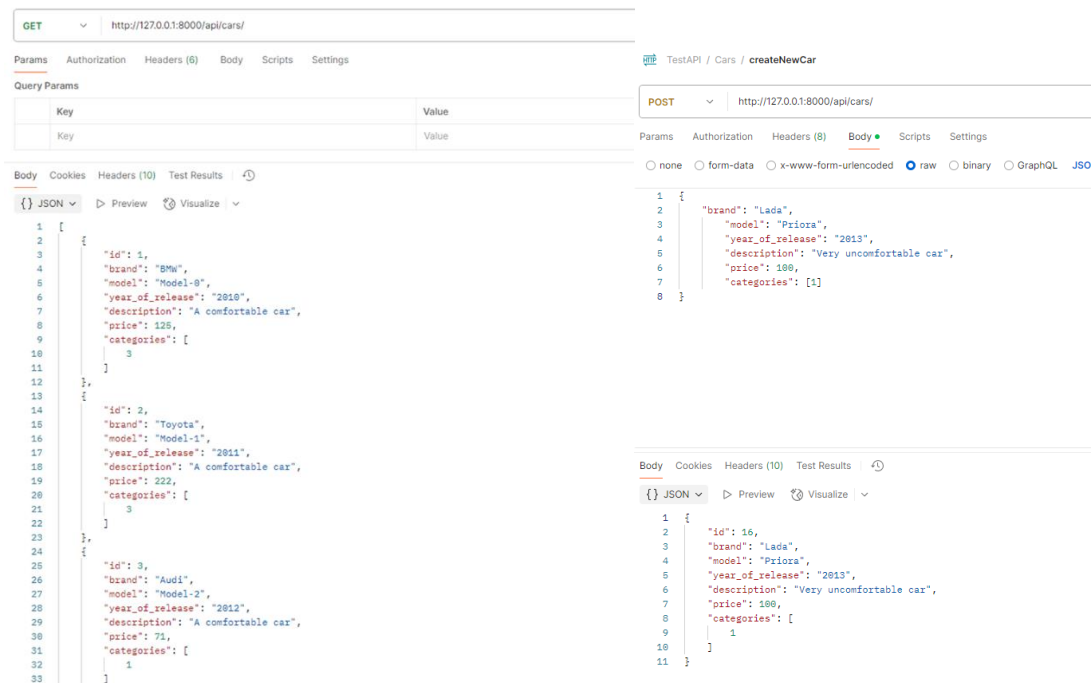
    def get_queryset(self): 1 idd0d0
        user = self.request.user
        if user.is_staff:
            return Payment.objects.all() # админ видит всё
        return Payment.objects.filter(booking__user=user) # только свои платежи

```

Далее для тестирования мы создали коллекцию запросов в Postman:



Проведено пошаговое тестирование всех операций. Далее представлены некоторые из них:



4. Авторизация

Следующим действием была реализована система авторизации пользователей с использованием JWT-токенов. Были разработаны и протестированы основные запросы для регистрации, входа в систему, смены пароля, выхода из аккаунта, а также обновления access-токена.

```
from api/register/

class RegisterView(APIView): 2 usages 1 idddol
    permission_classes = [AllowAny]

    def post(self, request): 1 idddol
        serializer = UserSerializer(data=request.data)
        if serializer.is_valid():
            user = User.objects.create_user(
                email=serializer.validated_data['email'],
                name=serializer.validated_data['name'],
                password=request.data['password']
            )
            return Response({'message': 'User created successfully'}, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

```
class ChangePasswordView(APIView): 2 usages 1 idddol
    permission_classes = [IsAuthenticated]

    def post(self, request): 1 idddol
        user = request.user

        old_password = request.data.get('old_password')
        new_password = request.data.get('new_password')
        new_password_repeat = request.data.get('new_password_repeat')

        # Проверка наличия всех полей
        if not all([old_password, new_password, new_password_repeat]):
            return Response({'detail': 'All fields are required'}, status=400)

        # Проверка текущего пароля
        if not check_password(old_password, user.password):
            return Response({'detail': 'Old password is incorrect'}, status=400)

        # Проверка совпадения новых паролей
        if new_password != new_password_repeat:
            return Response({'detail': 'New passwords do not match'}, status=400)

        # Проверка длины
        if len(new_password) < 8:
            return Response({'detail': 'New password must be at least 8 characters long'}, status=400)

        # Сохраняем
        user.set_password(new_password)
        user.save()

        return Response({'detail': 'Password updated successfully'}, status=200)
```



```

class LogoutView(APIView): 2 usages  ± idddol
    permission_classes = [IsAuthenticated]

    def post(self, request):  ± idddol
        try:
            refresh_token = request.data["refresh"]
            token = RefreshToken(refresh_token)
            token.blacklist()
            return Response({"detail": "Successfully logged out"}, status=205)
        except Exception as e:
            return Response({"detail": "Invalid token"}, status=400)

```

```

1  from rest_framework.permissions import BasePermission, SAFE_METHODS
2
3  class IsAdminOrReadOnly(BasePermission): 2 usages  ± idddol
4      #Только админ может создавать/редактировать/удалять
5      def has_permission(self, request, view):  ± idddol
6          return request.method in SAFE_METHODS or request.user.is_staff
7
8
9  class IsSelfOrAdmin(BasePermission): 3 usages  ± idddol
10     #Пользователь может управлять только собой. Админ - всеми
11     def has_object_permission(self, request, view, obj):  ± idddol
12         return request.user.is_staff or obj == request.user
13
14
15     class IsOwnerOrAdmin(BasePermission):  ± idddol
16         #Только владелец объекта или админ может редактировать/удалять
17         def has_object_permission(self, request, view, obj):  ± idddol
18             return request.user.is_staff or obj.user == request.user
19
20
21     class IsOwnerCanEditOnly(BasePermission): 2 usages  ± idddol
22         #Владелец может создавать/редактировать, но не удалять
23         def has_object_permission(self, request, view, obj):  ± idddol
24             if request.method in SAFE_METHODS or request.method in ('PUT', 'PATCH'):
25                 return obj.booking.user == request.user
26             if request.method == 'DELETE':
27                 return request.user.is_staff
28             return True
29

```

Набор маршрутов (эндпоинтов) для работы с системой авторизации и JWT-токенами:

```
# Регистрация и JWT
path('register/', RegisterView.as_view(), name='register'), # регистрация пользователя
path('token/', TokenObtainPairView.as_view(), name='token_obtain_pair'), # логин + получение JWT
path('token/refresh/', TokenRefreshView.as_view(), name='token_refresh'), # обновление access-токена
path('token/verify/', TokenVerifyView.as_view(), name='token_verify'), # проверка валидности access-токена
path('change-password/', ChangePasswordView.as_view(), name='change-password'), # смена пароля юзера
path('logout/', LogoutView.as_view(), name='logout'), # выход из системы пользователя
```

Модель «User» представляет пользователей системы аренды автомобилей и заменяет стандартную модель Django.

Основные поля: email — основной идентификатор (логин); name — имя пользователя; is_active — флаг активности; is_staff — определяет, является ли пользователь администратором.

Особенности:

- Вход в систему происходит по email;
- Используется собственный менеджер CustomUserManager для создания обычных и суперпользователей;
- Модель поддерживает авторизацию через JWT.

Права доступа реализованы через флаги is_staff и is_superuser, что позволяет разграничить функциональность между администраторами и обычными пользователями.

Также было проведено тестирование API: проверка работы аутентификации с помощью Postman; проверка статусов HTTP-ответов (200, 400, 401, 403 и т. д.).

▼

<http://127.0.0.1:8000/api/register/>

Params Authorization Headers (8) **Body** ● Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   "email": "testovik@example.com",
3   "name": "name",
4   "password": "123"
5 }
```

Body Cookies Headers (10) Test Results

 JSON Preview Visualize

```
1  {
2  |    "message": "User created successfully"
3  }
```

Save Share

1127

<http://127.0.0.1:8000/api/token/>

Send

Params Authorization Headers (8) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   "email": "testovik@example.com",
3   "password": "123"
4 }
```

body Cookies Headers (10) Test Results 200 OK - 610 ms

{ } JSON Preview Visualize

```
1 {
2   "refresh": "eyJ3bGdc1O1IUz11NiIsInR8cCt6IkpXVCJ9.eyJ0b2t1b190eXB1IjoicmVncmVzaCI6InV4cCt6MTc0NTAwNTYBMClawF0iXjNkQNDAwODQwL3Q6dGk1O1I3OW8zJmM3IjE0YjJ1YmUxZDZlNDhkZmFhZABOSiIsInVzZK3IawQ1OjEweQ.ZwMStsT-S3jM_-TMkClVP1gImnuRoDuwG_VY8Wz1w",
3   "access": "eyJ3bGdc1O1IUz11NiIsInR8cCt6IkpXVCJ9.eyJ0b2t1b190eXB1IjoicmVncmVzaCI6InV4cCt6MTc0NTAwNTYBMClawF0iXjNkQNDAwODQwL3Q6dGk1O1I3OW8zJmM3IjE0YjJ1YmUxZDZlNDhkZmFhZABOSiIsInVzZK3IawQ1OjEweQ.ZwMStsT-S3jM_-TMkClVP1gImnuRoDuwG_VY8Wz1w"
4 }
```

Body Cookies Headers (10) Test Results 200 OK • 610 ms • 797 B • Save Response

{ } JSON ▾ ▶ Preview 📊 Visualize ▾

[illegible]

TestAPI / Users / ChangePassword

Save

Share

POST

http://127.0.0.1:8000/api/change-password/

Send

Params

Authorization

Headers (10)

Body

Scripts

Settings

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON

```
1 {
2   "old_password": "123",
3   "new_password": "eeee8888",
4   "new_password_repeat": "eeee8888"
5 }
```

Body

Cookies

Headers (10)

Test Results

⌵

JSON

Preview

Visualize

```
1 {
2   "detail": "Password updated successfully"
3 }
```

200 OK

1.20 s

352 B

⌵

Save Response

⌵

🔍

🔗

TestAPI / Users / Logout

SaveShare

POST

http://127.0.0.1:8000/apl/logout/

Send

ParamsAuthorizationHeaders (10)BodyScriptsSettings

noneform-datax-www-form-urlencodedrawbinaryGraphQLJSON

Beautify

123

```
"refresh": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ0b2t1b190eXBIIjoiYm9mcmVzaCIsImlhV4oCI6MTc0NTAwNTY0MCIwF0IjoxNzQ0NDAwODQwLCJqdGkiOiIi30WE6Zjg5MmJlNjc0YjY3IiwiaXNja2N0b2NzPhYzA0OSIsInVzZXJfaWQ10jEwfi0.ZmHSgtsT-S3yM_-TMkClVP1gImnuRoDuwG_Yy0Nwz1d"
```

BodyCookiesHeaders (10)Test Results

205 Reset Content31 ms357 BSave Response

JSONPreviewVisualize

```
1 {
2   "detail": "Successfully logged out"
3 }
```

5. Разработка пользовательского интерфейса и взаимодействие с API

Затем было разработано клиентское веб-приложение с использованием Django и шаблонов HTML. Реализованы основные пользовательские сценарии: регистрация, вход в систему, просмотр данных, оформление бронирований и оплат.

Приложение взаимодействует с сервером через REST API: данные передаются в шаблоны и отображаются динамически. Для авторизации и защиты запросов используется JWT. Интерфейс протестирован на корректность отображения и связи с сервером.

Базовая страница «base.html»:

```
1 <!DOCTYPE html>
2 <html lang="ru">
3 <head>
4 <meta charset="UTF-8">
5 <title>{% block title %}Аэропорт{% endblock %}</title>
6 <meta name="viewport" content="width=device-width, initial-scale=1">
7 <!-- Bootstrap 5 CSS -->
8 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" rel="stylesheet">
9 <style>
10 body { background: #f5f5f5; }
11 .navbar-custom {
12 background: #fff;
13 border-bottom: 1px solid #000;
14 box-shadow: 0 2px 0 rgba(0,0,0,0.03);
15 }
16 .navbar-custom .navbar-brand {
17 font-weight: 400;
18 color: #007bff;
19 }
20 .navbar-custom .nav-link {
21 font-weight: 500;
22 color: #212121;
23 }
24 .navbar-custom .nav-link.active,
25 .navbar-custom .nav-link:focus,
26 .navbar-custom .nav-link:hover {
27 color: #0056b3;
28 }
29 </style>
30 </head>
31 <body>
32 <nav class="navbar navbar-expand-lg navbar-custom">
33 <div class="container-fluid">
34 <a class="navbar-brand" href="/api/site/get/-/desktop/">
35 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav"
36 aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
37 <span class="navbar-toggler-icon"></span>
38 </button>
39 <div class="collapse navbar-collapse" id="navbarNav">
40 <ul class="navbar-nav me-auto" id="navbarLinks">
41 <!-- Тут будет JS-код -->
42 </ul>
43 </div>
44 </div>
45 </nav>
46 <div class="container py-4">
47 <div class="block content">
48 </div>
49 </div>
50 <!-- Bootstrap JS (bundle) -->
51 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>
52 <script>
53 function logout() {
54 localStorage.clear();
55 window.location.href = "/api/site/login/";
56 }
57 function renderNavbar() {
58 const token = localStorage.getItem('access');
59 const links = document.getElementById('navbarLinks');
60 if (!links) return;
61 if (token) {
62 links.innerHTML = `
63 <li class="nav-item"><a class="nav-link" href="/api/site/cars/">Машины</li>
64 <li class="nav-item"><a class="nav-link" href="/api/site/profile/">Профиль</li>
65 <li class="nav-item"><a class="nav-link" href="/api/site/register/">Регистрация</li>
66 </li>
67 </div>
68 <div class="collapse navbar-collapse">
69 <div class="collapse navbar-collapse" id="navbarNav">
70 <ul class="navbar-nav me-auto" id="navbarLinks">
71 <li class="nav-item"><a class="nav-link" href="/api/site/cars/">Машины</li>
72 <li class="nav-item"><a class="nav-link" href="/api/site/register/">Регистрация</li>
73 <li class="nav-item"><a class="nav-link" href="/api/site/login/">Вход</li>
74 </ul>
75 </div>
76 </div>
77 </script>
78 </body>
79 </html>
```

Далее все страницы наследуются от нее:

1 Регистрация

```
1 {% extends 'blog/base.html' %}
2 {% block title %}Регистрация{% endblock %}
3
4 {% block content %}
5 <h2>Регистрация</h2>
6 <form method="post">
7 <div class="form-group">
8 <label>Email:</label><br>
9 <input type="email" name="email" required><br>
10
11 <label>Имя:</label><br>
12 <input type="text" name="name" required><br>
13
14 <label>Пароль:</label><br>
15 <input type="password" name="password" required><br>
16
17 <button type="submit">Зарегистрироваться</button>
18 </form>
19 {% endblock %}
20
```

```
1 <!-- api/site/register -->
2 def register_page(request):
3     if request.method == 'POST':
4         email = request.POST.get('email')
5         name = request.POST.get('name')
6         password = request.POST.get('password')
7
8         if User.objects.filter(email=email).exists():
9             messages.error(request, 'Пользователь с таким email уже существует')
10             return redirect('register-page')
11
12         user = User.objects.create_user(email=email, name=name, password=password)
13         messages.success(request, 'Регистрация прошла успешно! Теперь войдите')
14         return redirect('login-page')
15
16 return render(request, template_name='blog/register.html')
```

2 Платежи

```

document.getElementById("paymentForm").addEventListener("submit", async (e) => {
  e.preventDefault();
  const method = document.getElementById("method").value;

  const response = await secureFetch("/api/payments/", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({
      booking: bookingId,
      payment_method: method,
      payment_status: "completed"
    })
  });

  const msg = document.getElementById("message");
  const paymentInfo = document.getElementById("paymentInfo");

  if (response.ok) {
    const data = await response.json();
    msg.style.color = "green";
    msg.innerText = "Оплата прошла успешно!";

    // Обновляем информацию о платеже
    paymentInfo.innerHTML = `
      <p><strong>Сумма:</strong> ${data.amount} Р</p>
      <p><strong>Статус:</strong> Оплачен ☒

```

3 Профиль

```

async function loadUser() {
  const res = await secureFetch("/api/users/me/");
  const data = await res.json();
  document.getElementById("userName").innerText = data.name || "-";
  document.getElementById("userEmail").innerText = data.email || "-";
}

async function loadBookings() {
  const bookingRes = await secureFetch("/api/bookings/");
  const paymentRes = await secureFetch("/api/payments/");

  const bookings = await bookingRes.json();
  const payments = await paymentRes.json();

  const list = document.getElementById("bookingList");
  list.innerHTML = "";

  if (bookings.length === 0) {
    list.innerHTML = "<li>Бронирований нет</li>";
  } else {
    bookings.forEach(b => {
      // Maximus ace nitroxx ace pro0 f0ame
      const related = payments.filter(p => p.booking === b.id);
      const latestPayment = related.sort((a, b) => new Date(b.payment_date) - new Date(a.payment_date))[0];

      const isPaid = latestPayment?.payment_status === "completed";
      const payStatus = isPaid ? "Оплачено" : "Не оплачено";
      const payLink = isPaid ? `<a href="/api/site/payment/${b.id}/"> Оплатить</a>` : "";

      const item = document.createElement("li");
      item.innerHTML = `
        <strong>${b.car.brand} ${b.car.model}</strong><br>
        ${b.rental_start_date} - ${b.rental_end_date}<br>
        Статус брони: ${b.order.status}<br>
        Оплата: ${payStatus}<br>
        ${payLink}
        <br><button onClick="deleteBooking(${b.id})">Удалить</button>
      `;
      list.appendChild(item);
    });
  }
}

async function loadPayments() {
  const res = await secureFetch("/api/payments/");
  const data = await res.json();

  const list = document.getElementById("paymentList");
  list.innerHTML = "";

  if (data.length === 0) {
    list.innerHTML = "<li>Платежей нет</li>";
  } else {
    data.forEach(p => {
      const item = document.createElement("li");
      item.innerHTML = `<p>${p.amount} Р - ${p.payment_method} - ${p.payment_status}</p>`;
      list.appendChild(item);
    });
  }
}

async function deleteBooking(bookingId) {
  if (!confirm("Удалить бронирование?")) return;

  const res = await secureFetch("/api/bookings/${bookingId}/", {
    method: "DELETE"
  });

  if (res.ok) {
    alert("Бронирование удалено");
    await loadBookings();
    await loadPayments();
  } else {
    alert("Ошибка при удалении");
  }
}

```

4 Логин

```

document.getElementById("loginForm").addEventListener("submit", async function(e) {
  e.preventDefault();

  const email = this.email.value;
  const password = this.password.value;

  const response = await fetch("/api/token/", {
    method: "POST",
    headers: {
      "Content-Type": "application/json"
    },
    body: JSON.stringify({ email, password })
  });

  const data = await response.json();

  if (response.ok) {
    localStorage.setItem("access", data.access);
    localStorage.setItem("refresh", data.refresh);

    // Получаем инфу о пользователе
    const userRes = await fetch("/api/users/me/", {
      headers: {
        Authorization: "Bearer " + data.access
      }
    });
    const userData = await userRes.json();
    localStorage.setItem("is_staff", userData.is_staff); // сохраняем флаг

    window.location.href = "/api/site/profile/";
  } else {
    document.getElementById("loginError").innerText = data.detail || "Ошибка входа";
  }
});

```

5 Бронирования

```

70 document.getElementById("bookingForm").addEventListener("submit", async (e) => {
71   e.preventDefault();
72
73   const token = localStorage.getItem("access");
74   const carId = document.getElementById("bookingForm").dataset.carId;
75   const startDate = document.getElementById("start_date").value;
76   const endDate = document.getElementById("end_date").value;
77   const errorMessage = document.getElementById("errorMessage");
78
79   if (!token) {
80     window.location.href = "/api/site/login/";
81     return;
82   }
83   const parsedCarId = parseInt(carId);
84   console.log("ID машины, переданный в запрос:", parsedCarId); // отладка
85
86   const response = await secureFetch("/api/bookings/", {
87     method: "POST",
88     headers: {
89       "Content-Type": "application/json",
90     },
91     body: JSON.stringify({
92       car: parseInt(carId),
93       rental_start_date: startDate,
94       rental_end_date: endDate
95     })
96   });
97
98   if (response.ok) {
99     const data = await response.json();
100     window.location.href = "/api/site/payment/${data.id}/";
101   } else {
102     try {
103       const errorData = await response.json();
104       console.error("Ошибка бронирования:", errorData);
105       errorMessage.innerText = "Ошибка: " + JSON.stringify(errorData);
106     } catch (e) {
107       errorMessage.innerText = "Ошибка бронирования.";
108     }
109   }
110 }

```

blog > booking.html

6 Автомобили

```
65 ~ async function loadCars(categoryId = null) {
66   const res = await secureFetch("/api/cars/");
67   const carList = document.getElementById("carList");
68   carList.innerHTML = "";
69
70   if (!res.ok) {
71     carList.innerHTML = "<p>Ошибка загрузки машин</p>";
72     return;
73   }
74
75   const data = await res.json();
76   const isAdmin = localStorage.getItem("is_staff") === "true";
77   const filtered = categoryId ? data.filter(car => car.categories.includes(categoryId)) : data;
78
79   if (filtered.length === 0) {
80     carList.innerHTML = "<p>Нет машин в выбранной категории.</p>";
81     return;
82   }
83
84   filtered.forEach(car => {
85     const col = document.createElement("div");
86     col.className = "col-md-4 mb-4";
87
88     const images = car.images.length > 0
89       ? car.images.map((img, i) => `
90         <div class="carousel-item ${i === 0 ? 'active' : ''}>
91           
92         </div>
93       `).join('')
94       : `<div class="carousel-item active">img src="https://via.placeholder.com/400x200?text=${car.brand}${car.model}" class="d-block w-100" alt="Placeholder"></div>`;
95
96     const carouselId = `carousel-${car.id}`;
97
98     col.innerHTML = `
99     <div class="card shadow-sm h-100">
100       <div id="${carouselId}" class="carousel slide" data-bs-ride="carousel">
101         <div class="carousel-inner">${images}</div>
102         <button class="carousel-control-prev" type="button" data-bs-target="#${carouselId}" data-bs-slide="prev">
103           <span class="carousel-control-prev-icon"></span>
104         </button>
105         <button class="carousel-control-next" type="button" data-bs-target="#${carouselId}" data-bs-slide="next">
106           <span class="carousel-control-next-icon"></span>
107         </button>
108       </div>
109       <div class="card-body d-flex flex-column">
110         <h5 class="card-title">${car.brand} ${car.model}</h5>
111         <p class="card-text">
112           Год: ${car.year_of_release}<br>
113           Цена: <strong>${car.price}</strong>₽/день</strong>
114         </p>
115         <a href="/api/site/booking/${car.id}/" class="btn btn-primary mt-auto">Забронировать</a>
116         ${isAdmin ? `
117         <div class="mt-2">
118           <button class="btn btn-sm btn-outline-secondary mb-1" onclick="toggleImageInput('${car.id}')">Добавить фото</button>
119           <div id="input-container-${car.id}" style="border: 1px solid #ccc; padding: 5px; margin-top: 5px;">
120             <input type="text" id="url-${car.id}" class="form-control form-control-sm mb-1" placeholder="https://example.com/car.jpg">
121             <button class="btn btn-sm btn-success" onclick="submitImageUrl('${car.id}')">Сохранить</button>
122           </div>
123         </div>
124         ` : ''}
125       </div>
126     </div>
127     `;
128     carList.appendChild(col);
129   });
130 }
131
132 function toggleImageInput(carId) {
133   const container = document.getElementById(`input-container-${carId}`);
134   container.style.display = container.style.display === "none" ? "block" : "none";
135 }
136
137 async function submitImageUrl(carId) {
138   const url = document.getElementById(`url-${carId}`).value.trim();
139   if (!url) {
140     alert("Введите ссылку на изображение");
141     return;
142   }
143 }
```

Функция для обновления access-токена:

```
async function refreshAccessToken() {
  const refresh = localStorage.getItem("refresh");
  if (!refresh) return false;

  const response = await fetch("/api/token/refresh/", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ refresh })
  });

  if (response.ok) {
    const data = await response.json();
    localStorage.setItem("access", data.access);
    return true;
  } else {
    localStorage.clear();
    window.location.href = "/api/site/login/";
    return false;
  }
}
```

Проверка access-токена на валидность:

```
async function verifyToken() {
  const access = localStorage.getItem("access");
  if (!access) {
    window.location.href = "/api/site/login/";
    return false;
  }

  const res = await fetch("/api/token/verify/", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ token: access })
  });

  if (res.ok) {
    return true;
  } else {
    // access истёк — попробуем обновить
    const refreshed = await refreshAccessToken();
    return refreshed;
  }
}
```


Безопасный fetch или же кастомный fetch, который перед этим отлавливает 401 ошибку и если нужно обновляет access-токен:

```
async function secureFetch(url, options = {}) {
  const access = localStorage.getItem("access");
  const opts = {
    ...options,
    headers: {
      ...options.headers,
      "Authorization": `Bearer ${access}`
    }
  };
  let res = await fetch(url, opts);

  if (res.status === 401) {
    const refreshed = await refreshAccessToken();
    if (refreshed) {
      const newAccess = localStorage.getItem("access");
      opts.headers["Authorization"] = `Bearer ${newAccess}`;
      res = await fetch(url, opts);
    } else {
      window.location.href = "/api/site/login/";
    }
  }

  return res;
}
```

Маршруты к нашим страницам:

```
# HTML-страницы
<= /api/site/register/
path( route: 'site/register/', register_page, name='register-page'),
<= /api/site/login/
path( route: 'site/login/', login_page, name='login-page'),
<= /api/site/profile/
path( route: 'site/profile/', profile_page, name='profile-page'),
<= /api/site/cars/
path( route: 'site/cars/', cars_page, name='cars-page'),
<= /api/site/booking/{car_id}/
path( route: 'site/booking/<int:car_id>', booking_page, name='booking-page'),
<= /api/site/payment/{booking_id}/
path( route: 'site/payment/<int:booking_id>', payment_page, name='payment-page'),
<= /api/site/booking/{booking_id}/delete/
path( route: 'site/booking/<int:booking_id>/delete/', delete_booking, name='delete-booking'),
```

6. Финализация приложения и упаковка в Docker

Заключительным шагом интеграция клиентской и серверной частей приложения и выполнена контейнеризация проекта с использованием Docker.

Созданы два отдельных контейнера:

- контейнер с веб-приложением (Django), содержащий серверную часть и API;
- контейнер с базой данных PostgreSQL, подключаемый через переменные окружения.

С помощью `docker-compose.yml` организовано взаимодействие сервисов и запуск всего приложения одной командой. Для сборки образов использованы `Dockerfile` и `.dockerignore`. Проведено тестирование доступности API и клиентского интерфейса.

`.dockerignore`:

```
1 # Виртуальные окружения
2 .venv/
3 venv/
4 .venv1/
5
6 # Python cache
7 __pycache__/
8 *.py[cod]
9 *.pyo
10
11 # IDE/редактор
12 .idea/
13 .vscode/
14
15 # Git-репозиторий
16 .git/
17 .gitignore
18
19 # Конфиденциальные данные
20 .env
21
22 # Файлы БД, резервные копии
23 *.postgres
24 *.bak
25 *.mwb
26
27 # Логи
28 *.log
29
30 # Служебные и системные файлы
31 *.DS_Store
32 *.swp
```

`.env` (переменные окружения):

```
1 POSTGRES_DB=mydb
2 POSTGRES_USER=myuser
3 POSTGRES_PASSWORD=123
4 DB_HOST=db
5 DB_PORT=5432
```

docker-compose.yml:

```
1 version: '3.9'
2
3 services:
4   db:
5     image: postgres:15
6     restart: always
7     env_file:
8       - .env
9     volumes:
10      - postgres_data:/var/lib/postgresql/data
11     ports:
12      - "5438:5432"
13
14   web:
15     build: .
16     command: gunicorn Project_TSP.wsgi:application --bind 0.0.0.0:8000
17     volumes:
18       - ./app
19     ports:
20       - "8000:8000"
21     depends_on:
22       - db
23     env_file:
24       - .env
25
26 volumes:
27   postgres_data:
```

Dockerfile:

```
1 # Dockerfile
2 FROM python:3.13
3
4 ENV PYTHONDONTWRITEBYTECODE=1
5 ENV PYTHONUNBUFFERED=1
6
7 WORKDIR /app
8
9 COPY requirements.txt .
10 RUN pip install -r requirements.txt
11
12 COPY . .
13
14 CMD ["gunicorn", "Project_TSP.wsgi:application", "--bind", "0.0.0.0:8000"]
```

Контейнеры в Docker:

Containers [Give feedback](#)

View all your running containers and applications. [Learn more](#)

Container CPU usage ⓘ

No containers are running.

Container memory usage ⓘ

No containers are running.

[Show charts](#)

<input type="checkbox"/>	Name	Container ID	Image	Port(s)	CPU (%)	Last state	Actions
<input type="checkbox"/>	project_tsp	-	-	-	N/A	4 seconds ago	▶ ⋮ 🗑
<input type="checkbox"/>	web-1	7311e861da7f	project_tsp	8000:8000	N/A	4 seconds ago	▶ ⋮ 🗑
<input type="checkbox"/>	db-1	8f75edb322da	postgres:15	5438:5432	N/A	8 seconds ago	▶ ⋮ 🗑

Конечный вид веб-приложения:

Автопрокат

МашиныПрофильВыйти

Категории


Все

Economy


Business

Luxury


SUV




Ford Focus
Год: 2016
Цена: 13561Р/день
Забронировать




Tesla Model S
Год: 2016
Цена: 6491Р/день
Забронировать




BMW A6
Год: 2019
Цена: 4546Р/день
Забронировать



Mercedes Model S



Tesla Corolla



Автопрокат

МашиныПрофильВыйти

Личный кабинет

Имя: admin

Email: admin@example.com

Мои бронирования:

BMW A6

2025-05-16 → 2025-05-18

Статус брони: pending

Оплата: ☒ оплачено

Удалить

Tesla Corolla

2025-05-18 → 2025-05-28

Статус брони: pending

Оплата: ☒ оплачено

Удалить

Ford Focus

2025-05-18 → 2025-05-19

Статус брони: pending

Оплата: ☒ оплачено

Удалить

Ford Focus

2025-05-23 → 2025-05-24

Статус брони: pending

Оплата: ☒ оплачено

Удалить

Мои платежи:

9092Р — crypto — completed

32510Р — crypto — completed

13561Р — cash — completed

13561Р — cash — completed

Регистрация

Email:

admin@example.com

Имя:

Пароль:

.....

Вход

Email:

admin@example.com

Пароль:

.....