МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

(национальный исследовательский университет)

Институт № 7 «Интеллектуальные и робототехнические системы»

Кафедра 703

**Отчёт**

**«Технологии сетевого взаимодействия»**

**Курсовой проект на тему:**

**“Разработка серверного приложения с API“**

Работу выполнил

студент группы М7О-506С-20

Моисеев Илья Николаевич\_\_\_\_\_\_\_\_\_\_\_\_

Работу принял

Старший преподаватель кафедры 703

Барчев Николай Борисович\_\_\_\_\_\_\_\_\_\_\_\_

Дата сдачи отчета

\_\_\_\_\_\_\_\_\_\_\_\_

Москва

2024

**Оглавление**

[1.Задание 3](#_Toc185957342)

[2.Псевдокод алгоритмов решения поставленной задачи 4](#_Toc185957343)

[3.Сведения о программной реализации 9](#_Toc185957344)

[4.Инструкция пользователя 14](#_Toc185957345)

[1. Аутентификация и управление пользователями 15](#_Toc185957346)

[2. Работа с автомобилями 16](#_Toc185957347)

[3. Управление регистрациями 18](#_Toc185957348)

[5. Тестирование программы 20](#_Toc185957349)

[5.1. Тестирование API с помощью запросов через Swagger 20](#_Toc185957350)

[Тестирование API для /carsdb 20](#_Toc185957351)

[Тестирование API для /regdb 24](#_Toc185957352)

[5.2. Тестирование API с помощью запросов через html страницу 28](#_Toc185957353)

[6.Листинг программы 28](#_Toc185957354)

# **1.Задание**

1. Подготовить серверное приложение с API, реализующим следующие методы взаимодействия с клиентами:

* сущность автомобилей: состоит из атрибутов, каждая из которых включает три поля – марки, модели и номерного знака;
* сущность регистрационных сведений: состоит из атрибутов, каждая из которых включает четыре поля – номерного знака, фамилии и адреса владельца, года выпуска.

2. В процессе проектирования предусмотреть необходимые по смыслу задания проверки корректности данных, а также адекватное задаче взаимодействие с пользователем.

**В обязательном порядке использовать языковые средства организации программных единиц.**

3. Программы подготовить с использованием средств одной из реализаций языков программирования **C++ или Python.**

# **2.Псевдокод алгоритмов решения поставленной задачи**

**auth\_crud.py**

register\_user\_crud

Начало  
Проверить, существует ли пользователь с данным email в базе данных  
Если пользователь существует  
Выбросить ошибку: Email уже зарегистрирован  
Захешировать пароль пользователя  
Сформировать данные нового пользователя (имя, фамилия, email, захешированный пароль)  
Сохранить данные пользователя в базе данных  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Вернуть сообщение об успешной регистрации  
Конец

login\_user\_crud

Начало  
Найти пользователя в базе данных по email  
Если пользователь не найден или пароль неверный  
Выбросить ошибку: Неверные учетные данные  
Создать JWT-токен для пользователя  
Вернуть токен и его тип  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

**car\_crud.py**

update\_car\_by\_license\_plate

Начало  
Если ключ license\_plate присутствует в данных для обновления  
Выбросить ошибку: Изменение номерного знака запрещено  
Найти автомобиль в базе данных по номерному знаку  
Если автомобиль не найден  
Выбросить ошибку: Автомобиль не найден  
Обновить данные автомобиля в базе данных  
Вернуть флаг успешного обновления  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

search\_cars

Начало  
Найти автомобили в базе данных, соответствующие запросу  
Преобразовать найденные записи в объекты модели Car  
Вернуть список найденных автомобилей  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

add\_car

Начало  
Проверить, существует ли автомобиль с данным номерным знаком в базе данных  
Если автомобиль найден  
Выбросить ошибку: Автомобиль с таким номерным знаком уже существует  
Сохранить данные нового автомобиля в базу данных  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

delete\_car\_by\_license\_plate

Начало  
Найти автомобиль в базе данных по номерному знаку  
Если автомобиль не найден  
Выбросить ошибку: Автомобиль с указанным номерным знаком не найден  
Удалить запись об автомобиле  
Вернуть флаг успешного удаления  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

get\_all\_cars

Начало  
Извлечь все записи автомобилей из базы данных  
Преобразовать записи в объекты модели Car  
Вернуть список всех автомобилей  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

**registration\_crud.py**

update\_registration\_by\_license\_plate

Начало  
Если ключ license\_plate присутствует в данных для обновления  
Выбросить ошибку: Изменение номерного знака запрещено  
Найти запись в базе данных по номерному знаку  
Если запись не найдена  
Выбросить ошибку: Регистрация не найдена  
Обновить данные регистрации в базе данных  
Вернуть флаг успешного обновления  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

search\_registrations

Начало  
Найти в базе данных регистрации, которые соответствуют запросу  
Преобразовать найденные записи в объекты модели Registration  
Вернуть список найденных регистраций  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

add\_registration

Начало  
Проверить, существует ли регистрация с таким номерным знаком в базе данных  
Если регистрация найдена  
Выбросить ошибку: Регистрация с данным номерным знаком уже существует  
Сохранить данные новой регистрации в базе данных  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

get\_all\_registrations

Начало  
Извлечь все записи регистраций из базы данных  
Преобразовать записи в объекты модели Registration  
Вернуть список всех регистраций  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

delete\_registration\_by\_license\_plate

Начало  
Найти запись регистрации по номерному знаку  
Если запись не найдена  
Выбросить ошибку: Регистрация не найдена  
Удалить запись о регистрации  
Вернуть флаг успешного удаления  
Если возникает ошибка базы данных  
Выбросить соответствующую ошибку  
Если происходит любая другая ошибка  
Выбросить ошибку неожиданного поведения  
Конец

# **3.Сведения о программной реализации**

При написании программы использовался язык программирования *Python 3.10*. Среда разработки – *PyCharm*, версия 1.70

Системные требования для запуска программы – ОС Windows 11, ОЗУ- 300 Мб, 250 Мб места на жёстком диске.

**Краткое описание спроектированных программных единиц логики**:

|  |  |  |
| --- | --- | --- |
| **Название программной единицы** | **Параметры** | **Описание функции** |
| **async def register\_user\_crud(user: UserCreate) -> Dict[str, str]** | user — объект данных пользователя для регистрации | Регистрирует нового пользователя. Проверяет уникальность email, хеширует пароль и сохраняет данные в базу данных. Выбрасывает исключения при ошибках валидации, работе с базой или других проблемах. |
| **async def login\_user\_crud(user: UserLogin) -> Dict[str, str]** | user — объект данных пользователя для авторизации | Аутентифицирует пользователя, проверяет email и пароль, создает и возвращает JWT-токен. Выбрасывает исключения при ошибках данных или взаимодействия с базой. |
| **async def update\_car\_by\_license\_plate(license\_plate: str, update\_data: Dict) -> bool** | license\_plate — номерной знак автомобиля, update\_data — данные для обновления | Обновляет данные автомобиля по номерному знаку. Запрещает изменение поля license\_plate. Возвращает флаг успешности или выбрасывает исключения при ошибках. |
| **async def search\_cars(query: str) -> List[Car]** | query — строка поиска (марка, модель или номерной знак) | Ищет автомобили в базе данных по запросу. Возвращает список подходящих объектов Car. Выбрасывает исключения при ошибках работы с базой. |
| **async def add\_car(car: Car) -> None** | car — объект автомобиля для добавления | Добавляет новый автомобиль в базу данных. Проверяет уникальность номерного знака. Выбрасывает исключения при ошибках данных или подключения к базе. |
| **async def delete\_car\_by\_license\_plate(license\_plate: str) -> bool** | license\_plate — номерной знак автомобиля | Удаляет автомобиль по номерному знаку. Возвращает флаг успешного удаления или выбрасывает исключения, если автомобиль не найден либо произошли ошибки в работе с базой. |
| **async def get\_all\_cars() -> List[Car]** | - | Извлекает список всех автомобилей из базы данных и возвращает их в виде объектов Car. Выбрасывает исключения при ошибках подключения к базе. |
| **async def update\_registration\_by\_license\_plate(license\_plate: str, update\_data: Dict[str, Union[str, int]]) -> bool** | license\_plate — номерной знак регистрации, update\_data — данные для обновления | Обновляет данные регистрации по номерному знаку. Запрещает изменение поля license\_plate. Возвращает флаг успешного обновления или выбрасывает исключения. |
| **async def search\_registrations(query: str) -> List[Registration]** | query — строка поиска (номерной знак, имя владельца или адрес) | Ищет регистрации в базе данных по запросу. Возвращает список объектов Registration. Выбрасывает исключения при ошибках работы с базой. |
| **async def add\_registration(registration: Registration) -> None** | registration — объект регистрации для добавления | Добавляет новую регистрацию в базу данных. Проверяет уникальность номерного знака. Выбрасывает исключения при ошибке данных или работы с базой. |
| **async def get\_all\_registrations() -> List[Registration]** | - | Извлекает список всех регистраций из базы данных и возвращает их в виде объектов Registration. Выбрасывает исключения при ошибках подключения к базе. |
| **async def delete\_registration\_by\_license\_plate(license\_plate: str) -> bool** | license\_plate — номерной знак регистрации | Удаляет регистрацию по номерному знаку. Возвращает флаг успешного удаления или выбрасывает исключения, если регистрация не найдена либо произошли ошибки с базой. |
| **def hash\_password(password: str) -> str** | password — пароль в виде обычной строки | Хэширует введенный пароль с использованием алгоритма bcrypt из библиотеки PassLib. Возвращает строку с хэшированным паролем. |
| **def verify\_password(plain\_password: str, hashed\_password: str) -> bool** | plain\_password — введенный пароль, hashed\_password — хэшированный пароль | Проверяет соответствие введенного пароля и хэшированного. Возвращает True, если пароли совпадают, иначе False. |
| **def create\_access\_token(data: dict) -> str** | data — словарь с данными, которые нужно закодировать в токен | Создает JWT-токен доступа, добавляя к данным время истечения (30 минут). Возвращает строку с созданным токеном. |
| **def decode\_access\_token(token: str) -> Optional[dict]** | token — JWT-токен для расшифровки | Расшифровывает и валидирует переданный JWT-токен. Возвращает извлеченные из токена данные в виде словаря или None, если токен недействителен. |

**Краткое описание спроектированных программных единиц моделей**:

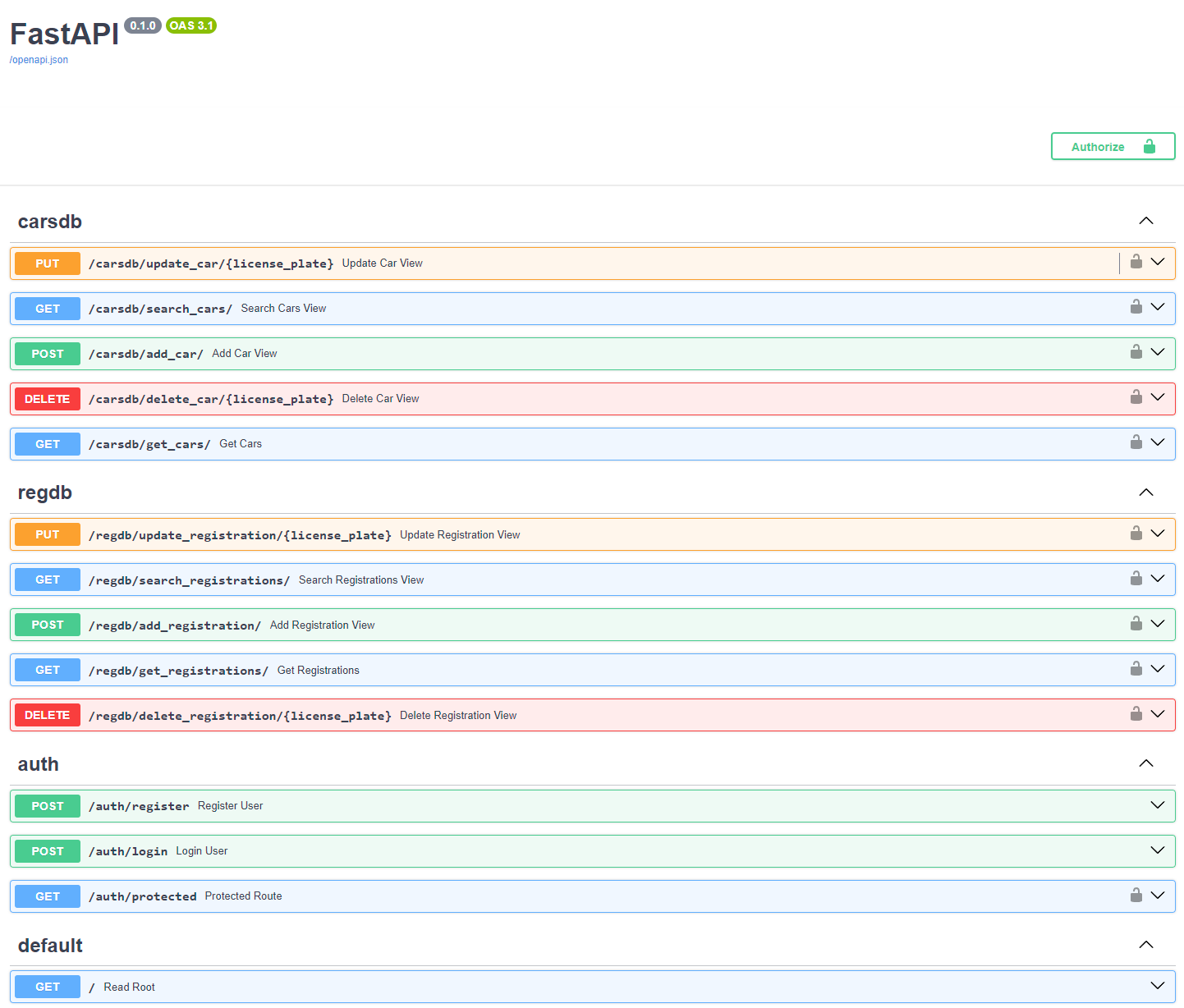
|  |  |  |
| --- | --- | --- |
| **Название программной единицы** | **Параметры** | **Описание функции** |
| **class Car(BaseModel)** | - | Модель для представления автомобиля. Определяет атрибуты make, model, license\_plate с валидацией их значений через методы. |
| **@field\_validator("make") def validate\_make(cls, value: str) -> str** | value — строковое значение марки автомобиля | Проверяет, что марка автомобиля состоит только из латинских букв, пробелов и дефисов, длина от 1 до 50 символов. |
| **@field\_validator("model") def validate\_model(cls, value: str) -> str** | value — строковое значение модели автомобиля | Проверяет, что модель автомобиля содержит только латинские буквы, цифры, пробелы и дефисы, длина от 1 до 50 символов. |
| **@field\_validator("license\_plate") def validate\_license\_plate(cls, value: str) -> str** | value — строковое значение номерного знака | Проверяет, что номерной знак состоит только из заглавных латинских букв и цифр, длиной от 3 до 10 символов. |
| **class Registration(BaseModel)** | - | Модель для представления регистрации транспортного средства. Определяет атрибуты license\_plate, owner\_name, owner\_address, year\_of\_manufacture с валидацией их значений через методы. |
| **@field\_validator("license\_plate") def validate\_license\_plate(cls, value: str) -> str** | value — строковое значение номерного знака | Проверяет корректность номерного знака: заглавные буквы, цифры, дефисы, длина до 10 символов. |
| **@field\_validator("owner\_name") def validate\_owner\_name(cls, value: str) -> str** | value — строковое значение имени владельца | Проверяет, что имя владельца состоит только из букв, пробелов и дефисов, длиной от 1 до 50 символов. |
| **@field\_validator("owner\_address") def validate\_owner\_address(cls, value: str) -> str** | value — строковое значение адреса владельца | Проверяет, что адрес состоит из букв, цифр и разрешенных символов адреса (точки, запятые, дефисы и др.), длиной от 1 до 100 символов. |
| **class UserCreate(BaseModel)** | - | Модель для создания нового пользователя. Определяет атрибуты first\_name, last\_name, email, password, confirm\_password, включает валидацию полей. |
| **class UserLogin(BaseModel)** | - | Модель для входа пользователя. Содержит атрибуты email и password. |

**Краткое описание спроектированных программных единиц маршрутов**:

|  |  |  |
| --- | --- | --- |
| **Название программной единицы** | **Параметры** | **Описание функции** |
| **@router.post("/register") async def register\_user(user: UserCreate) -> dict** | user — данные нового пользователя | Обработчик маршрута регистрации. Проверяет совпадение паролей, вызывает CRUD-функцию register\_user\_crud, возвращает сообщение об успешной регистрации или выбрасывает исключение. |
| **@router.post("/login") async def login\_user(user: UserLogin) -> dict** | user — данные пользователя для входа | Обработчик маршрута входа в систему. Проверяет email/пароль через CRUD-функцию login\_user\_crud, возвращает JWT-токен при успешной аутентификации или выбрасывает исключение. |
| **@router.get("/protected") async def protected\_route(token: str = Depends(oauth2\_scheme)) -> dict** | token — JWT-токен пользователя | Защищенный маршрут, проверяет и декодирует токен через decode\_access\_token. Возвращает сообщение с информацией о пользователе или выбрасывает исключение при недействительном/просроченном токене. |
| **@router.put("/update\_car/{license\_plate}") async def update\_car\_view(license\_plate: str, update\_data: Dict[str, Any], user: str = Depends(get\_current\_user))** | license\_plate — номерной знак, update\_data — данные для обновления, user — текущий пользователь | Обновляет данные автомобиля через CRUD-функцию update\_car\_by\_license\_plate. Проверяет токен пользователя, запрещает изменение номерного знака, выбрасывает ошибки, если данные некорректны или отсутствуют. |
| **@router.get("/search\_cars/") async def search\_cars\_view(query: str, user: str = Depends(get\_current\_user)) -> Dict[str, Any]** | query — строка поиска, user — текущий пользователь | Возвращает список автомобилей, удовлетворяющих поисковому запросу, через CRUD-функцию search\_cars. Проверяет токен пользователя. Выбрасывает исключение при ошибке работы с базой. |
| **@router.post("/add\_car/") async def add\_car\_view(car: Car, user: str = Depends(get\_current\_user)) -> Dict[str, str]** | car — объект автомобиля для добавления, user — текущий пользователь | Добавляет автомобиль через CRUD-функцию add\_car. Проверяет токен пользователя. Выбрасывает исключения при ошибке валидации, нарушении уникальности или других ошибках. |
| **@router.delete("/delete\_car/{license\_plate}") async def delete\_car\_view(license\_plate: str, user: str = Depends(get\_current\_user)) -> Dict[str, str]** | license\_plate — номерной знак автомобиля для удаления, user — текущий пользователь | Удаляет автомобиль через CRUD-функцию delete\_car\_by\_license\_plate. Проверяет токен пользователя и возвращает сообщение об успешном удалении или выбрасывает исключение при ошибках. |
| **@router.get("/get\_cars/") async def get\_cars(user: str = Depends(get\_current\_user)) -> Dict[str, Any]** | user — текущий пользователь | Возвращает список всех автомобилей через CRUD-функцию get\_all\_cars. Проверяет токен пользователя. Выбрасывает исключения при ошибке работы с базой данных. |
| **@router.put("/update\_registration/{license\_plate}") async def update\_registration\_view(license\_plate: str, update\_data: Dict[str, Any], user: str = Depends(get\_current\_user)) -> Dict[str, str]** | license\_plate — номерной знак регистрации, update\_data — данные для обновления, user — текущий пользователь | Обновляет данные регистрации через CRUD-функцию update\_registration\_by\_license\_plate. Проверяет токен пользователя и выбрасывает исключения при некорректных данных или ошибках работы с базой. |
| **@router.get("/search\_registrations/") async def search\_registrations\_view(query: str, user: str = Depends(get\_current\_user)) -> Dict[str, Any]** | query — строка поиска регистраций, user — текущий пользователь | Возвращает найденные регистрации через CRUD-функцию search\_registrations. Проверяет токен пользователя и выбрасывает исключение при ошибке работы с базой данных. |
| **@router.post("/add\_registration/") async def add\_registration\_view(registration: Registration, user: str = Depends(get\_current\_user)) -> Dict[str, str]** | registration — объект регистрации, user — текущий пользователь | Добавляет новую регистрацию через CRUD-функцию add\_registration. Проверяет токен пользователя, выбрасывает исключения при нарушении уникальности или других ошибках. |
| **@router.get("/get\_registrations/") async def get\_registrations(user: str = Depends(get\_current\_user)) -> Dict[str, Any]** | user — текущий пользователь | Возвращает список всех регистраций через CRUD-функцию get\_all\_registrations. Проверяет токен пользователя и выбрасывает исключения при ошибках работы с базой. |
| **@router.delete("/delete\_registration/{license\_plate}") async def delete\_registration\_view(license\_plate: str, user: str = Depends(get\_current\_user)) -> Dict[str, str]** | license\_plate — номерной знак для удаления, user — текущий пользователь | Удаляет регистрацию через CRUD-функцию delete\_registration\_by\_license\_plate. Проверяет токен пользователя и выбрасывает исключения при отсутствии записи или ошибках работы с базой данных. |

# **4.Инструкция пользователя**

Список всех маршрутов API из предоставляемого FastAPI GUI:



**1. Аутентификация и управление пользователями**

API предоставляет возможности регистрации, входа и защиты маршрутов с использованием токенов **JWT**.

**Конечные точки:**

1. **Регистрация пользователя**
   * **Маршрут:** POST /auth/register
   * **Описание:** Регистрирует нового пользователя.
   * **Параметры тела запроса (JSON):**

{  
 "first\_name": "Имя",  
 "last\_name": "Фамилия",  
 "email": "почта@example.com",  
 "password": "пароль",  
 "confirm\_password": "подтверждение пароля"  
}

* **Ответы:**
  + 200: Пользователь успешно зарегистрирован.
  + 400: Ошибка валидации (например, пароли не совпадают или email уже используется).

1. **Вход в систему**
   * **Маршрут:** POST /auth/login
   * **Описание:** Аутентифицирует пользователя и возвращает JWT-токен.
   * **Параметры тела запроса(application/x-www-form-urlencoded)**

grant\_type=password  
username=почта@example.com  
password=пароль

* **Ответы:**
  + 200: Успешная аутентификация, возвращает токен:

{  
 "access\_token": "JWT-токен",  
 "token\_type": "bearer"  
}

* 401: Неправильные учетные данные.

1. **Доступ к защищенному маршруту**
   * **Маршрут:** GET /auth/protected
   * **Описание:** Проверяет JWT-токен и дает доступ к защищенному контенту.
   * **Заголовок авторизации:** Authorization: Bearer <ваш\_токен>
   * **Ответ:**

{  
 "msg": "Hello, <email пользователя>!"  
}

**2. Работа с автомобилями**

API позволяет добавлять, обновлять, удалять, искать и просматривать все автомобили.

**Конечные точки:**

1. **Добавить автомобиль**
   * **Маршрут:** POST /carsdb/add\_car/
   * **Описание:** Добавляет новый автомобиль.
   * **Параметры тела запроса (JSON):**

{  
 "make": "Марка",  
 "model": "Модель",  
 "license\_plate": "Номерной знак"  
}

* **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
* **Ответы:**
  + 200: Автомобиль успешно добавлен.
  + 400: Ошибка валидации (например, автомобиль с таким номерным знаком уже существует).

1. **Обновить данные автомобиля**
   * **Маршрут:** PUT /carsdb/update\_car/{license\_plate}
   * **Описание:** Обновляет данные автомобиля (исключая номерной знак).
   * **Параметры URL:** license\_plate — номерной знак автомобиля.
   * **Параметры тела запроса (JSON):**

{  
 "make": "Новая марка",  
 "model": "Новая модель"  
}

* **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
* **Ответы:**
  + 200: Данные автомобиля обновлены.
  + 400: Ошибка валидации.
  + 404: Автомобиль не найден.

1. **Удалить автомобиль**
   * **Маршрут:** DELETE /carsdb/delete\_car/{license\_plate}
   * **Описание:** Удаляет автомобиль.
   * **Параметры URL:** license\_plate — номерной знак автомобиля.
   * **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
   * **Ответы:**
     + 200: Автомобиль успешно удален.
     + 404: Автомобиль с указанным номерным знаком не найден.
2. **Посмотреть все автомобили**
   * **Маршрут:** GET /carsdb/get\_cars/
   * **Описание:** Возвращает список всех автомобилей.
   * **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
   * **Ответ (пример):**

{  
 "cars": [  
 {  
 "make": "Toyota",  
 "model": "Camry",  
 "license\_plate": "ABC123"  
 }  
 ],  
 "user": "<email пользователя>"  
}

1. **Поиск автомобилей**
   * **Маршрут:** GET /carsdb/search\_cars/?query=<term>
   * **Описание:** Ищет автомобили по запросу (марка, модель или номерной знак).
   * **Пример:** /carsdb/search\_cars/?query=Toyota
   * **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
   * **Ответ (пример):**

{  
 "cars": [  
 {  
 "make": "Toyota",  
 "model": "Camry",  
 "license\_plate": "ABC123"  
 }  
 ],  
 "user": "<email пользователя>"  
}

**3. Управление регистрациями**

API предоставляет возможность создавать, обновлять, удалять и искать регистрации транспортных средств.

**Конечные точки:**

1. **Добавить регистрацию**
   * **Маршрут:** POST /regdb/add\_registration/
   * **Описание:** Добавляет новую регистрацию транспортного средства.
   * **Параметры тела запроса (JSON):**

{  
 "license\_plate": "Номерной знак",  
 "owner\_name": "Имя владельца",  
 "owner\_address": "Адрес владельца",  
 "year\_of\_manufacture": 2010  
}

* **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
* **Ответы:**
  + 200: Регистрация успешно добавлена.
  + 400: Ошибка валидации.

1. **Обновить регистрацию**
   * **Маршрут:** PUT /regdb/update\_registration/{license\_plate}
   * **Описание:** Обновляет данные регистрации (исключая номерной знак).
   * **Параметры URL:** license\_plate — номерной знак автомобиля.
   * **Параметры тела запроса (JSON):**

{  
 "owner\_name": "Новое имя владельца",  
 "owner\_address": "Новый адрес"  
}

* **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
* **Ответы:**
  + 200: Данные регистрации успешно обновлены.
  + 400: Ошибка валидации.
  + 404: Регистрация не найдена.

1. **Удалить регистрацию**
   * **Маршрут:** DELETE /regdb/delete\_registration/{license\_plate}
   * **Описание:** Удаляет регистрацию по номерному знаку.
   * **Параметры URL:** license\_plate — номерной знак автомобиля.
   * **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
   * **Ответы:**
     + 200: Регистрация успешно удалена.
     + 404: Регистрация с указанным номерным знаком не найдена.
2. **Просмотреть все регистрации**
   * **Маршрут:** GET /regdb/get\_registrations/
   * **Описание:** Возвращает список всех регистраций.
   * **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
   * **Ответ (пример):**

{  
 "registrations": [  
 {  
 "license\_plate": "ABC123",  
 "owner\_name": "Владимир",  
 "owner\_address": "Москва, Красная 1",  
 "year\_of\_manufacture": 2010  
 }  
 ],  
 "user": "<email пользователя>"  
}

1. **Поиск регистраций**
   * **Маршрут:** GET /regdb/search\_registrations/?query=<term>
   * **Описание:** Ищет регистрации по запросу (номерной знак, имя владельца, адрес).
   * **Пример:** /regdb/search\_registrations/?query=Владимир
   * **Требуемый заголовок:** Authorization: Bearer <ваш\_токен>
   * **Ответ (пример):**

{  
 "registrations": [  
 {  
 "license\_plate": "ABC123",  
 "owner\_name": "Владимир",  
 "owner\_address": "Москва, Красная 1",  
 "year\_of\_manufacture": 2010  
 }  
 ],  
 "user": "<email пользователя>"  
}

# **5. Тестирование программы**

## **5.1. Тестирование API с помощью запросов через Swagger**

### **Тестирование API для /carsdb**

#### **Тест-кейс №1: Добавление автомобиля**

**Тест:**  
POST запрос на адрес /carsdb/add\_car/ с телом:

{  
 "make": "Toyota",  
 "model": "Camry",  
 "license\_plate": "ABC123"  
}

**Ожидаемый результат:**

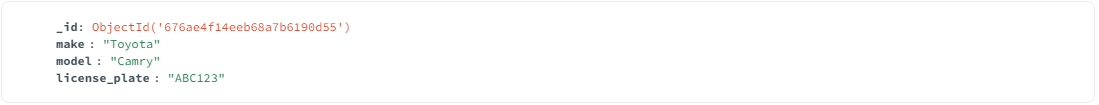
* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "message": "Car added successfully by alice@example.com"  
}

**Полученный результат:**



* **Рис. 1:** Скриншот успешного ответа из Swagger.



* **Рис. 2:** Скриншот из CompassDB, подтверждающий добавление автомобиля с номерным знаком ABC123.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №2: Попытка добавления автомобиля с уже существующим номерным знаком**

**Тест:**  
POST запрос на адрес /carsdb/add\_car/ с телом:

{  
 "make": "Honda",  
 "model": "Civic",  
 "license\_plate": "ABC123"  
}

**Ожидаемый результат:**

* Код возврата: 400 Bad Request
* Ожидаемый ответ:

{  
 "detail": "Car with given license plate already exists"  
}

**Полученный результат:**

****

* **Рис. 3:** Скриншот ответа из Swagger, подтверждающий ошибку при добавлении автомобиля.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №3: Получение списка всех автомобилей**

**Тест:**  
GET запрос на адрес /carsdb/get\_cars/.

**Ожидаемый результат:**

* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "cars": [  
 {  
 "id": "<car\_id>",  
 "make": "Toyota",  
 "model": "Camry",  
 "license\_plate": "ABC123"  
 }  
 ],  
 "user": "ilja.moiseeff2012@gmail.com"  
}

**Полученный результат:**



* **Рис. 4:** Скриншот успешного ответа из Swagger.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №4: Редактирование данных автомобиля**

**Тест:**  
PUT запрос на адрес /carsdb/update\_car/ABC123 с телом:

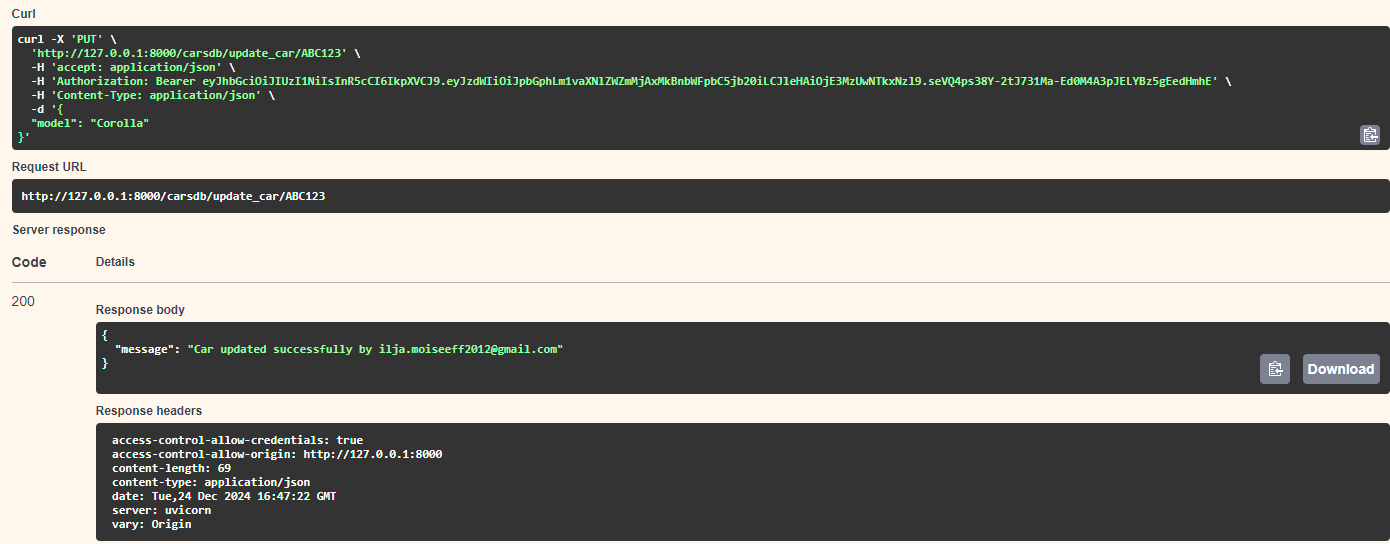
{  
 "model": "Corolla"  
}

**Ожидаемый результат:**

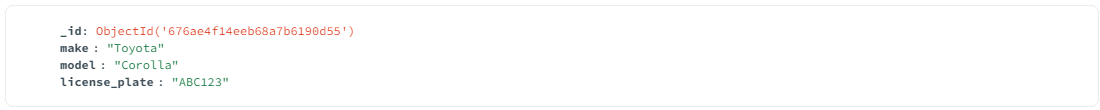
* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "message": "Car updated successfully by alice@example.com"  
}

**Полученный результат:**



* **Рис. 5:** Скриншот успешного ответа из Swagger.



* **Рис. 6:** Скриншот из CompassDB, подтверждающий обновление модели на Corolla.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №5: Попытка редактирования с недействительным номерным знаком**

**Тест:**  
PUT запрос на адрес /carsdb/update\_car/NONEXISTING123 с телом:

{  
 "model": "Accord"  
}

**Ожидаемый результат:**

* Код возврата: 404 Not Found
* Ожидаемый ответ:

{  
 "detail": "Car not found or no updates applied"  
}

**Полученный результат:**



* **Рис. 7:** Скриншот ответа из Swagger, подтверждающий ошибку, связанную с отсутствием автомобиля в системе.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №6: Поиск автомобиля по номеру**

**Тест:**  
GET запрос на адрес /carsdb/search\_cars/ABC.

**Ожидаемый результат:**

* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "cars": [  
 {  
 "make": "Toyota",  
 "model": "Corolla",  
 "license\_plate": "ABC123"  
 }  
 ],  
 "user": "ilja.moiseeff2012@gmail.com"  
}

**Полученный результат:**



* **Рис. 8:** Скриншот успешного ответа из Swagger.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №7: Попытка поиска несуществующего автомобиля**

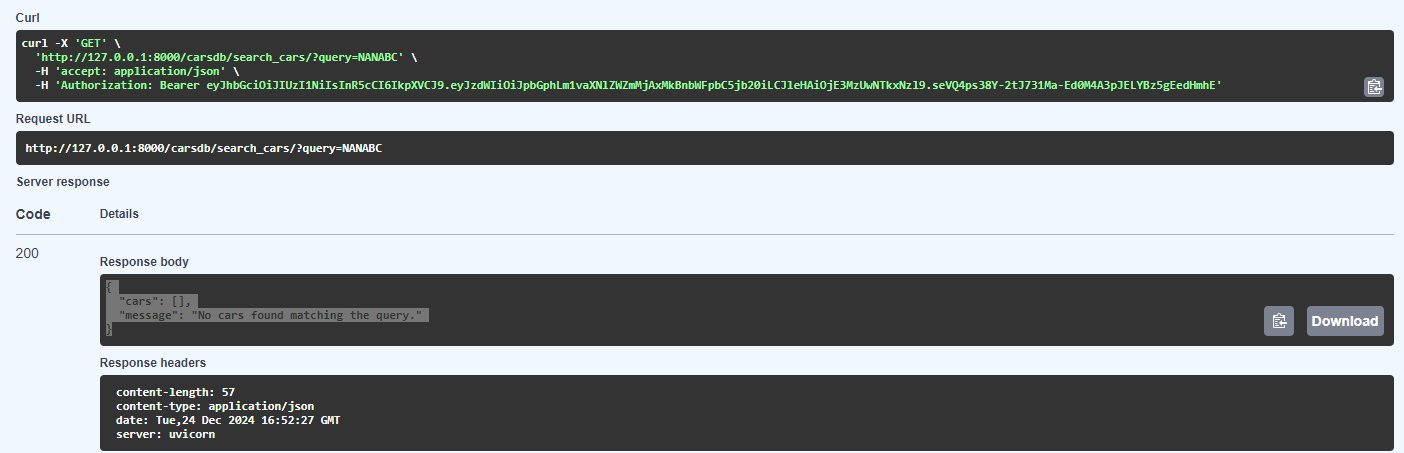
**Тест:**  
GET запрос на адрес /carsdb/search\_cars/NANABC.

**Ожидаемый результат:**

* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "cars": [],  
 "message": "No cars found matching the query."  
}

**Полученный результат:**



* **Рис. 9:** Скриншот ответа из Swagger, подтверждающий отсутствие автомобиля.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №8: Удаление автомобиля**

**Тест:**  
DELETE запрос на адрес /carsdb/delete\_car/ABC123.

**Ожидаемый результат:**

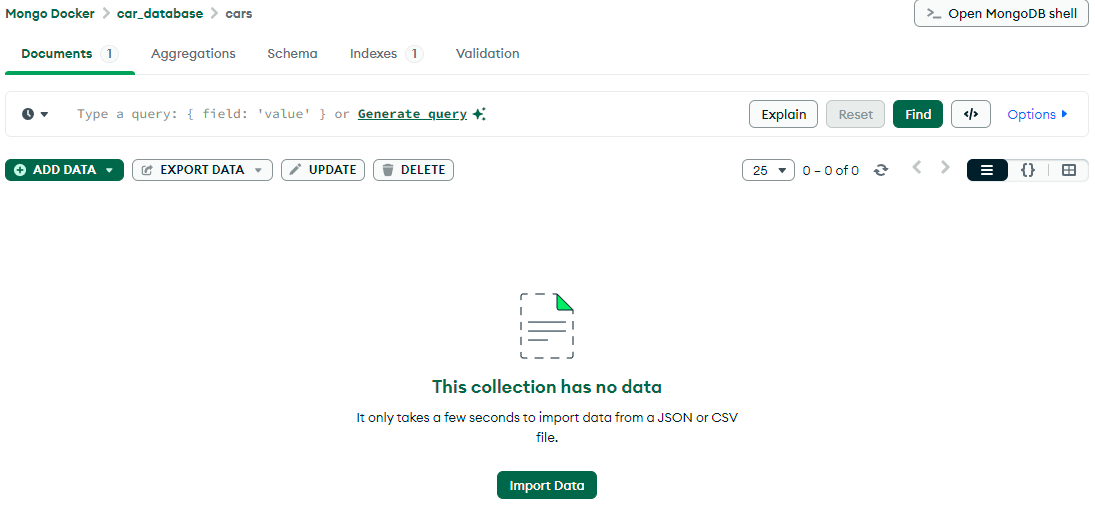
* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "message": "Car deleted successfully by alice@example.com"  
}

**Полученный результат:**



* **Рис. 10:** Скриншот успешного ответа из Swagger.



* **Рис. 11:** Скриншот из CompassDB, подтверждающий удаление автомобиля с номерным знаком ABC123.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №9: Попытка удаления несуществующего автомобиля**

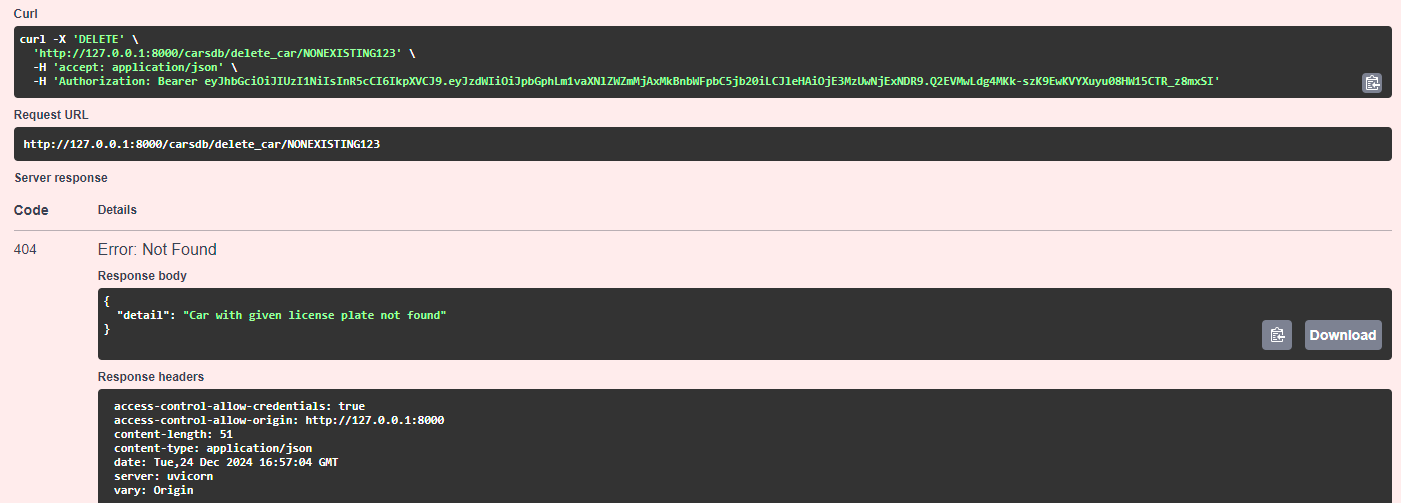
**Тест:**  
DELETE запрос на адрес /carsdb/delete\_car/NONEXISTING123.

**Ожидаемый результат:**

* Код возврата: 404 Not Found
* Ожидаемый ответ:

{  
 "detail": "Car with given license plate not found"  
}

**Полученный результат:**



* **Рис. 12:** Скриншот ответа из Swagger, подтверждающий ошибку об отсутствии автомобиля.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

### **Тестирование API для /regdb**

#### **Тест-кейс №1: Добавление регистрации**

**Тест:**  
POST запрос на адрес /regdb/add\_registration/ с телом:

{  
 "license\_plate": "ABC123",  
 "owner\_name": "Alice Smith",  
 "owner\_address": "123 Main Street",  
 "year\_of\_manufacture": 2015  
}

**Ожидаемый результат:**

* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "message": "Registration added successfully by alice@example.com"  
}

**Полученный результат:**



* **Рис. 13:** Скриншот успешного ответа из Swagger.



* **Рис. 14:** Скриншот из CompassDB, подтверждающий добавление регистрации на автомобиль с номером ABC123.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №2: Попытка добавления регистрации для уже зарегистрированного автомобиля**

**Тест:**  
POST запрос на адрес /regdb/add\_registration/ с телом:

{  
 "license\_plate": "ABC123",  
 "owner\_name": "Bob Brown",  
 "owner\_address": "456 Elm Street",  
 "year\_of\_manufacture": 2020  
}

**Ожидаемый результат:**

* Код возврата: 400 Bad Request
* Ожидаемый ответ:

{  
 "detail": "Registration for the given license plate already exists"  
}

**Полученный результат:**



* **Рис. 15:** Скриншот ответа из Swagger, подтверждающий ошибку.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №3: Получение всех регистраций**

**Тест:**  
GET запрос на адрес /regdb/get\_registrations/.

**Ожидаемый результат:**

* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "registrations": [  
 {  
 "id": "<registration\_id>",  
 "license\_plate": "ABC123",  
 "owner\_name": "Alice Smith",  
 "owner\_address": "123 Main Street",  
 "year\_of\_manufacture": 2015  
 }  
 ],  
 "user": "ilja.moiseeff2012@gmail.com"  
}

**Полученный результат:**



* **Рис. 16:** Скриншот успешного ответа из Swagger.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №4: Редактирование данных регистрации**

**Тест:**  
PUT запрос на адрес /regdb/update\_registration/ABC123 с телом:

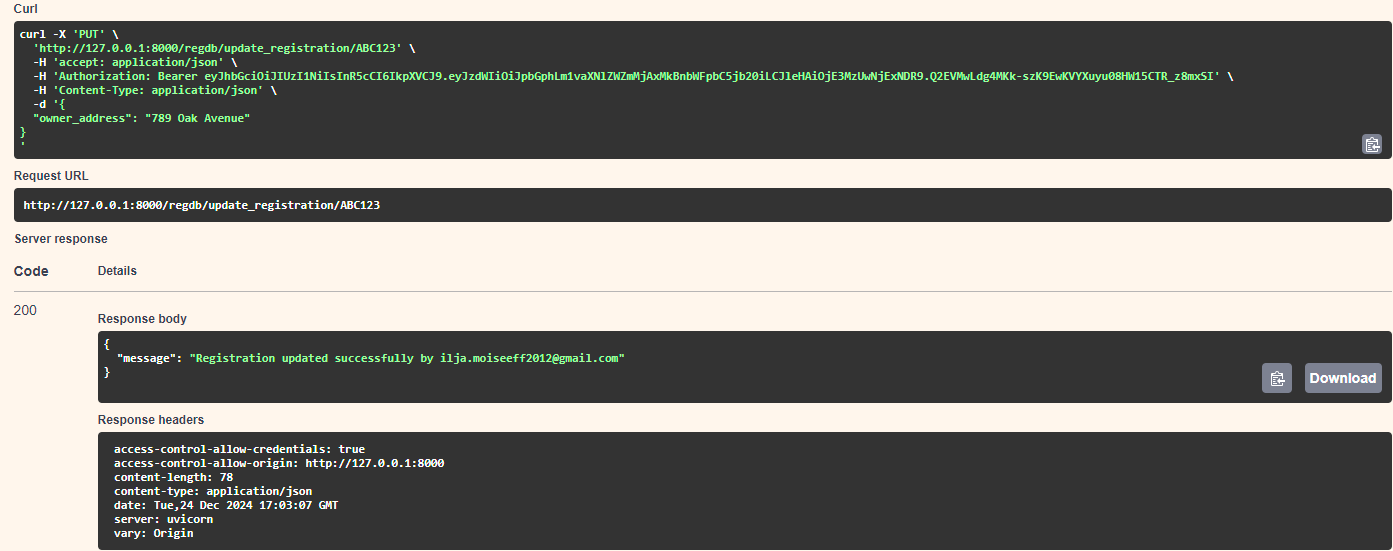
{  
 "owner\_address": "789 Oak Avenue"  
}

**Ожидаемый результат:**

* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "message": "Registration updated successfully by alice@example.com"  
}

**Полученный результат:**



* **Рис. 17:** Скриншот успешного ответа из Swagger.



* **Рис. 18:** Скриншот из CompassDB, подтверждающий обновление адреса владельца до 789 Oak Avenue.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №5: Попытка редактирования данных несуществующей регистрации**

**Тест:**  
PUT запрос на адрес /regdb/update\_registration/NONEXISTING123 с телом:

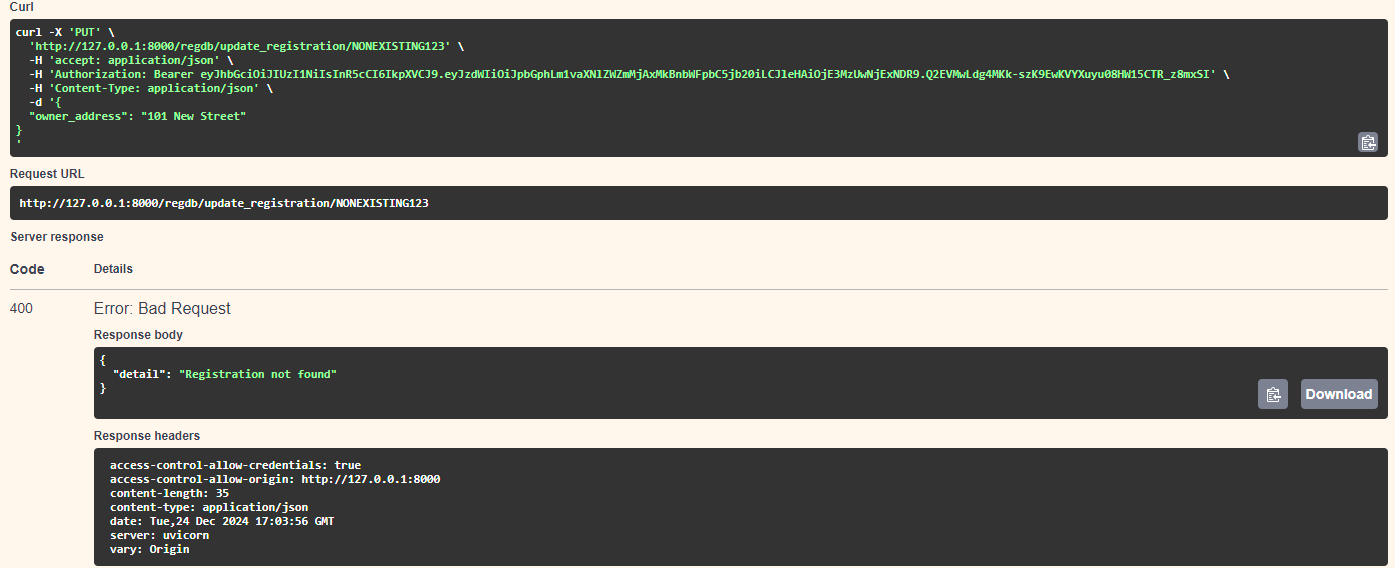
{  
 "owner\_address": "101 New Street"  
}

**Ожидаемый результат:**

* Код возврата: 404 Not Found
* Ожидаемый ответ:

{  
 "detail": "Registration not found"  
}

**Полученный результат:**



* **Рис. 19:** Скриншот ответа из Swagger, подтверждающий, что регистрация не найдена.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №6: Поиск регистрации**

**Тест:**  
GET запрос на адрес /regdb/search\_registrations/ABC123.

**Ожидаемый результат:**

* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "registrations": [  
 {  
 "license\_plate": "ABC123",  
 "owner\_name": "Alice Smith",  
 "owner\_address": "789 Oak Avenue",  
 "year\_of\_manufacture": 2015  
 }  
 ]  
}

**Полученный результат:**



* **Рис. 20:** Скриншот успешного ответа из Swagger.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №7: Попытка поиска несуществующей регистрации**

**Тест:**  
DELETE запрос на адрес /regdb/search\_registrations/NONEXISTING123.

**Ожидаемый результат:**

* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "registrations": [],  
 "message": "No registrations found matching the query."  
}

**Полученный результат:**



* **Рис. 21:** Скриншот ответа из Swagger, подтверждающий отсутствие регистрации.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №6: Удаление регистрации**

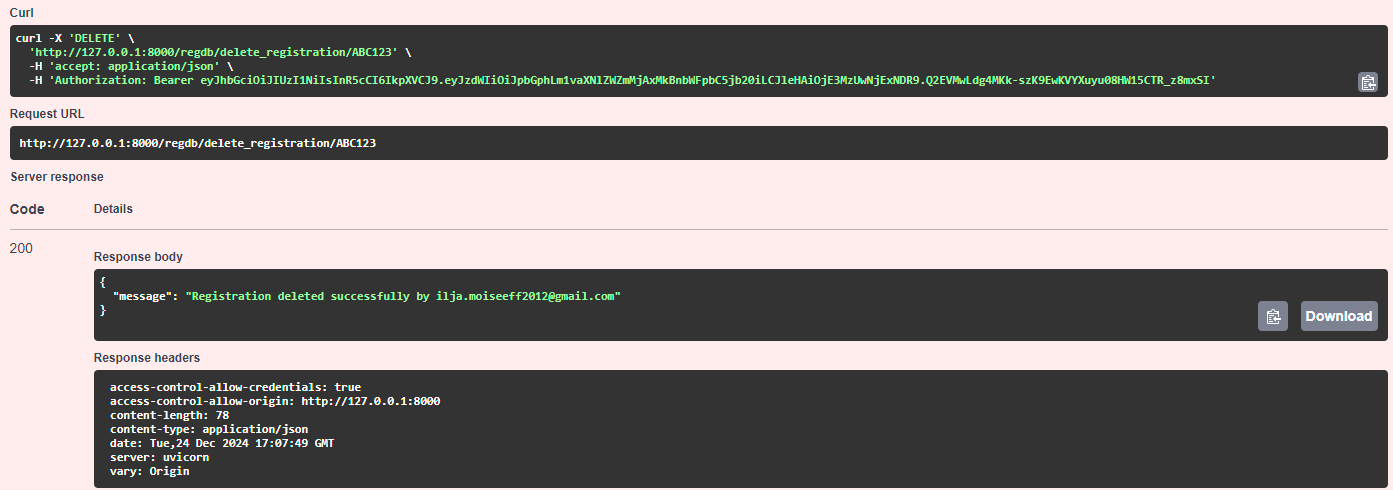
**Тест:**  
DELETE запрос на адрес /regdb/delete\_registration/ABC123.

**Ожидаемый результат:**

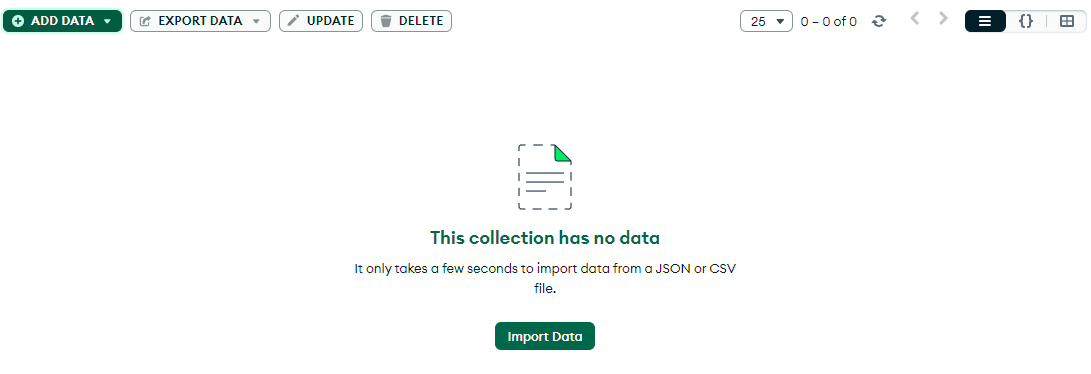
* Код возврата: 200 OK
* Ожидаемый ответ:

{  
 "message": "Registration deleted successfully by ilja.moiseeff2012@gmail.com"  
}

**Полученный результат:**



* **Рис. 22:** Скриншот успешного ответа из Swagger.



* **Рис. 23:** Скриншот из CompassDB, подтверждающий удаление регистрации на автомобиль с номерным знаком ABC123.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

#### **Тест-кейс №7: Попытка удаления несуществующей регистрации**

**Тест:**  
DELETE запрос на адрес /regdb/delete\_registration/NONEXISTING123.

**Ожидаемый результат:**

* Код возврата: 404 Not Found
* Ожидаемый ответ:

{  
 "detail": "Registration with given license plate not found"  
}

**Полученный результат:**



* **Рис. 24:** Скриншот ответа из Swagger, подтверждающий ошибку об отсутствии регистрации.

**Заключение:**  
Полученный результат соответствует ожидаемому результату.

## **5.2. Тестирование API с помощью запросов через html страницу**

### **Тестирование страниц регистрации и авторизации**

#### **Тест-кейс №1: Регистрация нового пользователя**

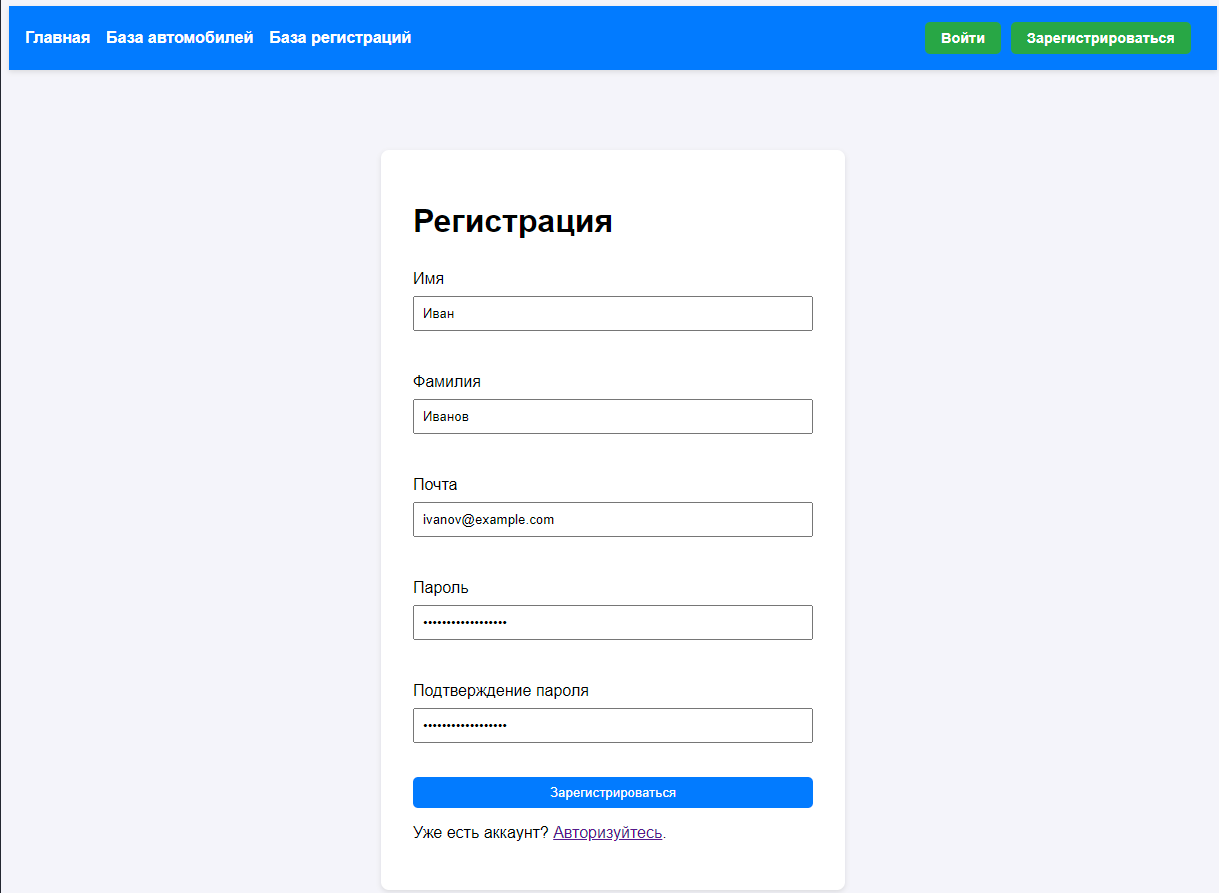
**Тест:**

1. Перейти на страницу регистрации (/static/html/register.html).
2. Заполнить все поля валидными данными:
   * Имя: Иван
   * Фамилия: Иванов
   * Почта: ivanov@example.com
   * Пароль: StrongPassword123!
   * Подтверждение пароля: StrongPassword123!
3. Нажать на кнопку "Зарегистрироваться".

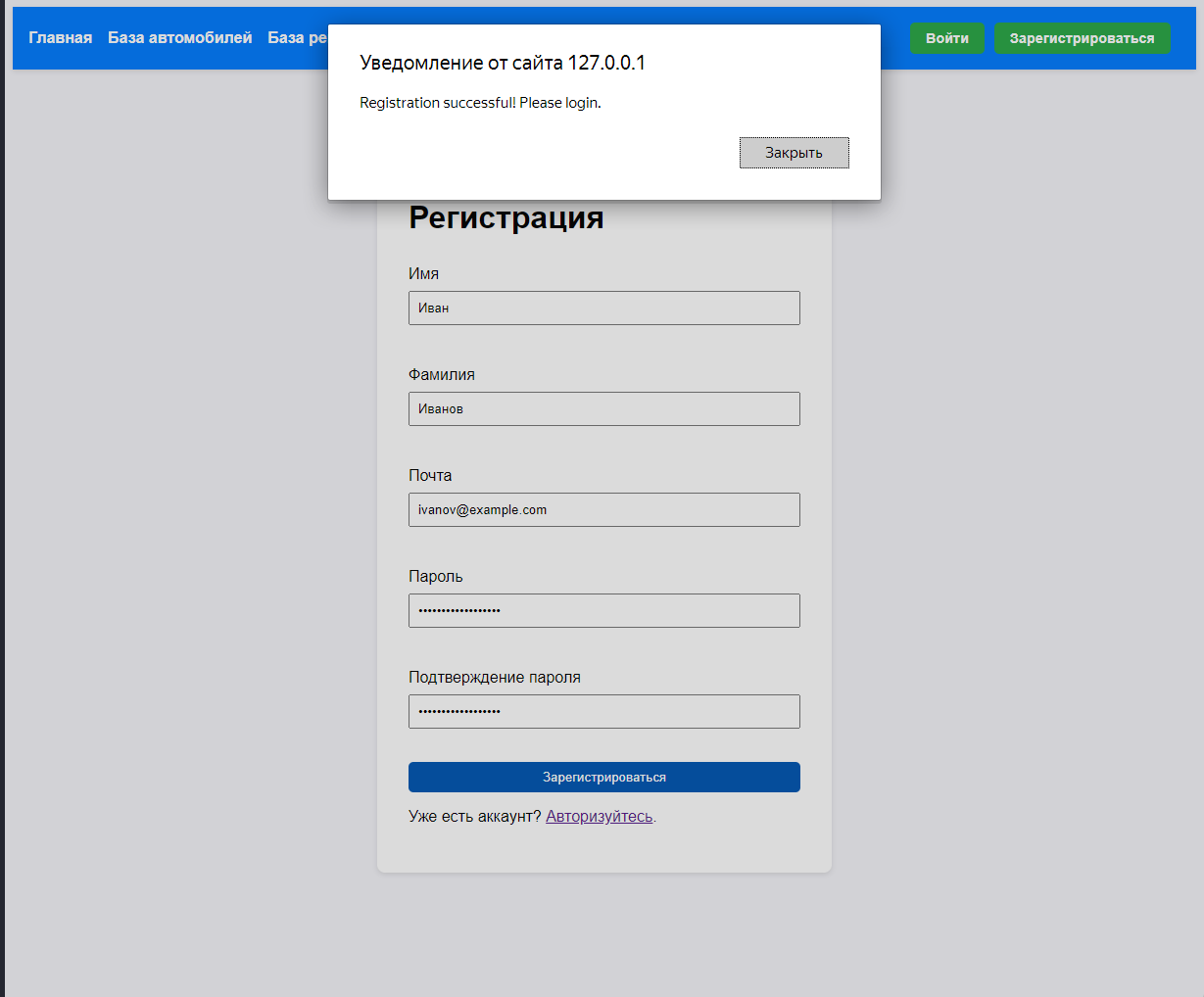
**Ожидаемый результат:**

* Код ответа сервера: 201 Created.
* Пользователь уведомлен: "Регистрация прошла успешно! Авторизуйтесь."
* Перенаправление на страницу авторизации (/static/html/login.html).

**Полученный результат:**



* **Рис. 25:** Скриншот страницы регистрации после заполнения формы.



* **Рис. 26:** Скриншот уведомления с подтверждением успешной регистрации.

**Заключение:**  
Функционал регистрации работает по ожиданиям.

#### **Тест-кейс №2: Регистрация при несовпадении паролей**

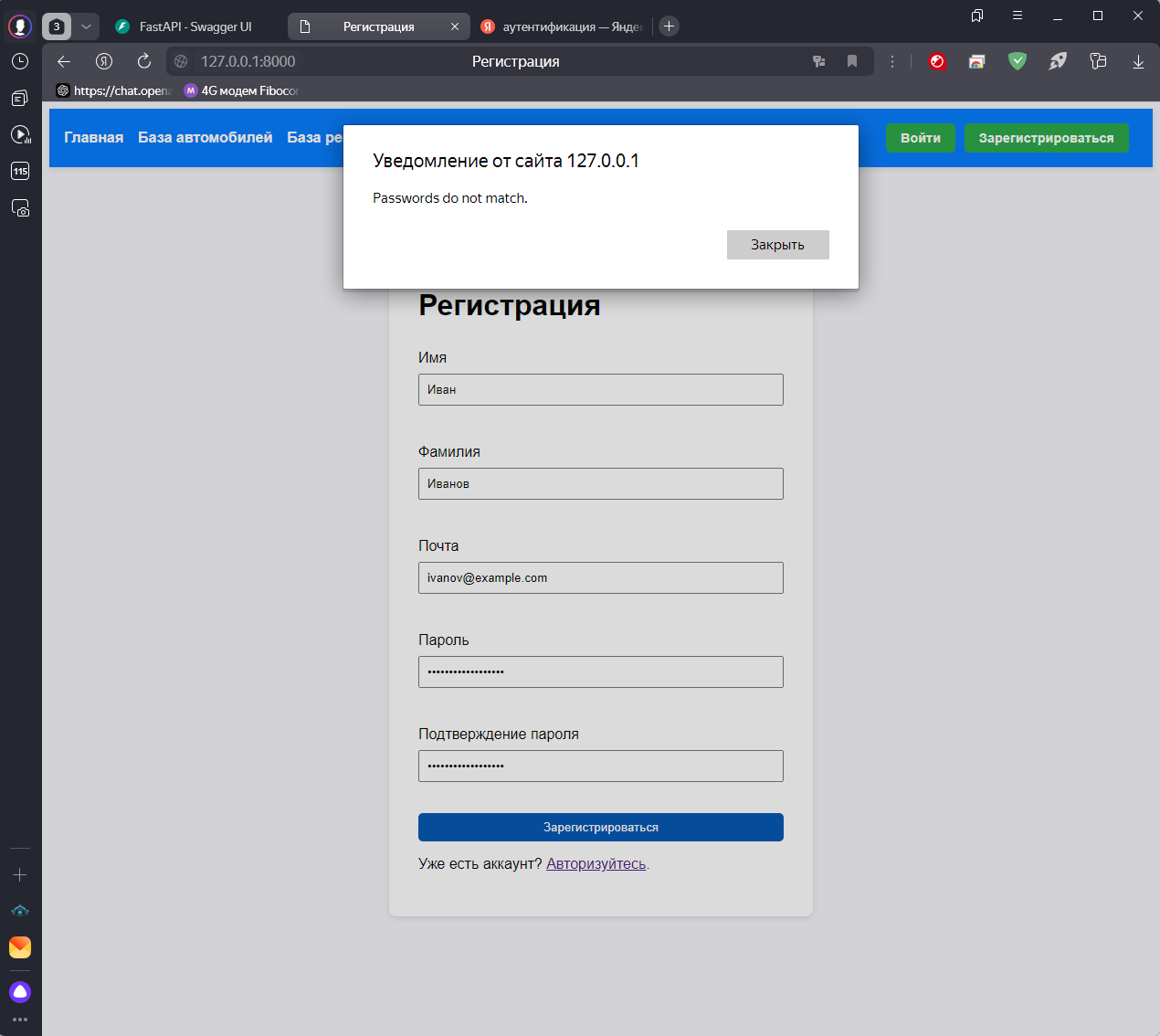
**Тест:**

1. Перейти на страницу регистрации (/static/html/register.html).
2. Заполнить все поля, но в поле подтверждения пароля указать значение, отличное от пароля:
   * Пароль: StrongPassword123!
   * Подтверждение пароля: WrongPassword123!
3. Нажать на кнопку "Зарегистрироваться".

**Ожидаемый результат:**

* Отображено сообщение об ошибке: "Пароли не совпадают."
* Данные не отправлены на сервер.

**Полученный результат:**



* **Рис. 27:** Скриншот страницы с выводом сообщения об ошибке.

**Заключение:**  
Проверка совпадения паролей на клиенте работает корректно.

#### **Тест-кейс №3: Авторизация с валидными данными**

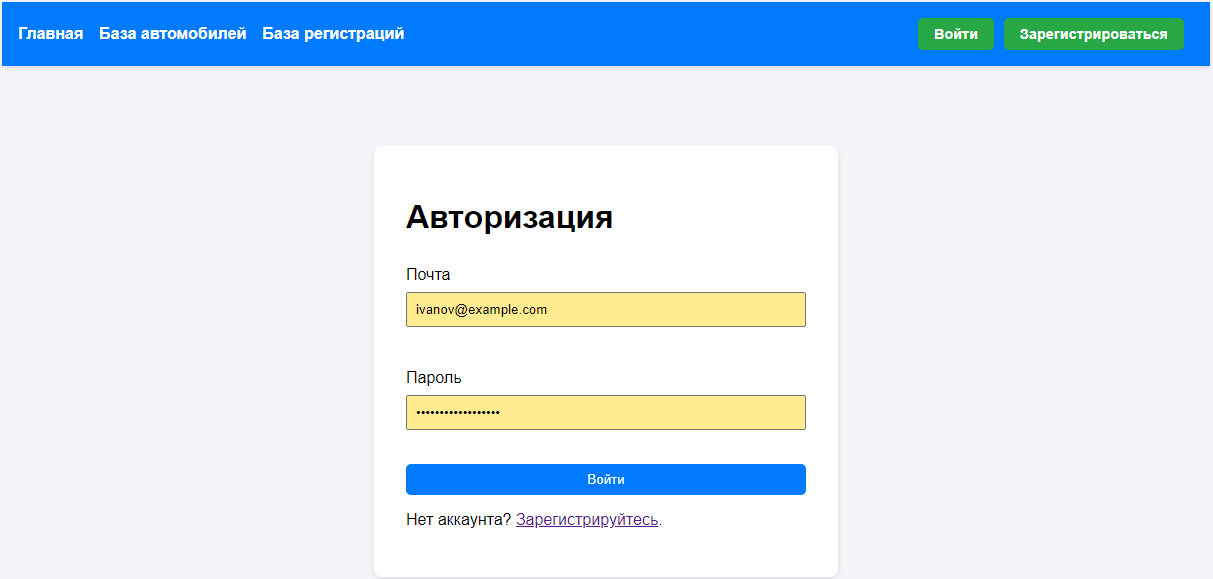
**Тест:**

1. Перейти на страницу авторизации (/static/html/login.html).
2. Заполнить поля формы:
   * Почта: ivanov@example.com
   * Пароль: StrongPassword123!
3. Нажать кнопку "Войти".

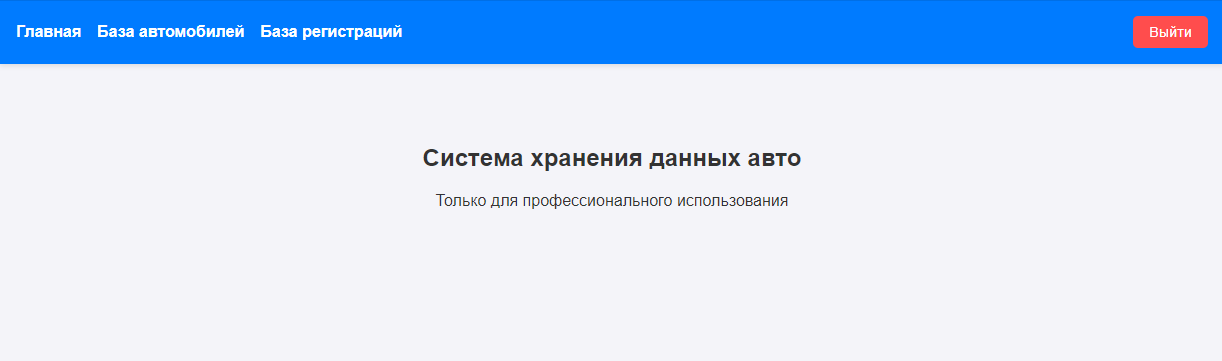
**Ожидаемый результат:**

* Код ответа сервера: 200 OK.
* Пользователь перенаправлен на главную страницу (/static/html/index.html).
* Локальное хранилище содержит токен авторизации: localStorage.getItem("token") возвращает токен.

**Полученный результат:**



* **Рис. 28:** Скриншот страницы входа после заполнения формы.



* **Рис. 29:** Скриншот страницы главной после успешной авторизации.

**Заключение:**  
Авторизация с валидными данными работает корректно. Токен успешно сохранён.

#### **Тест-кейс №4: Авторизация с неверным паролем**

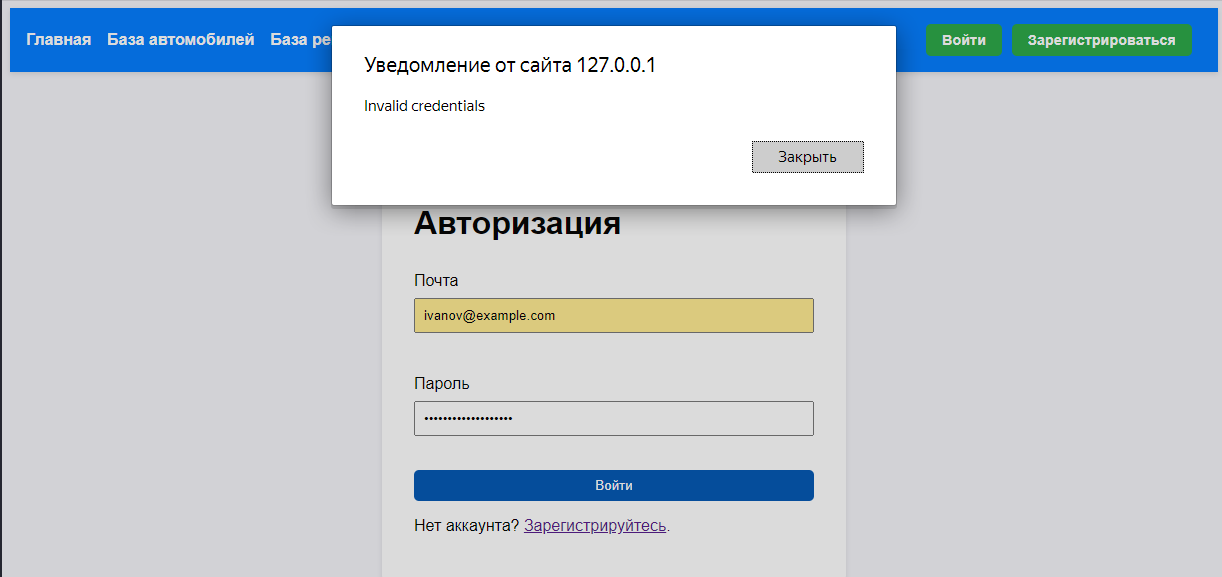
**Тест:**

1. Перейти на страницу авторизации (/static/html/login.html).
2. Заполнить поля формы:
   * Почта: ivanov@example.com
   * Пароль: WrongPassword123!
3. Нажать кнопку "Войти".

**Ожидаемый результат:**

* Код ответа сервера: 401 Unauthorized.
* Отображается сообщение об ошибке: "Неверная почта или пароль."

**Полученный результат:**



* **Рис. 30:** Скриншот страницы входа с выводом сообщения об ошибке.

**Заключение:**  
Сервер корректно обрабатывает попытки входа с неверным паролем. Сообщение об ошибке отображается.

#### **Тест-кейс №5: Проверка перенаправления неавторизованного пользователя**

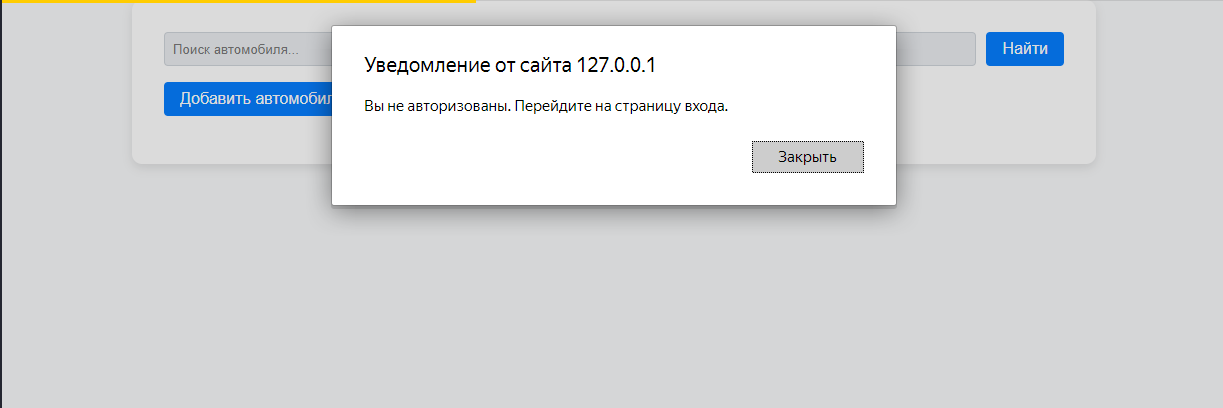
**Тест:**

1. Перейти на любую страницу для авторизованных пользователей (/static/html/cars.html) без авторизации.
2. Попытаться загрузить страницу операций с автомобилями.

**Ожидаемый результат:**

* Пользователь перенаправлен на страницу авторизации (/static/html/login.html).
* Отображено сообщение: "Вы не авторизованы. Перейдите на страницу входа."

**Полученный результат:**



* **Рис. 31:** Скриншот уведомления о необходимости авторизации.

**Заключение:**  
Механизм проверки авторизации работает корректно, пользователь перенаправляется.

### **Тестирование страницы управления автомобилями (/static/html/cars.html)**

#### **Тест-кейс №1: Отображение списка автомобилей**

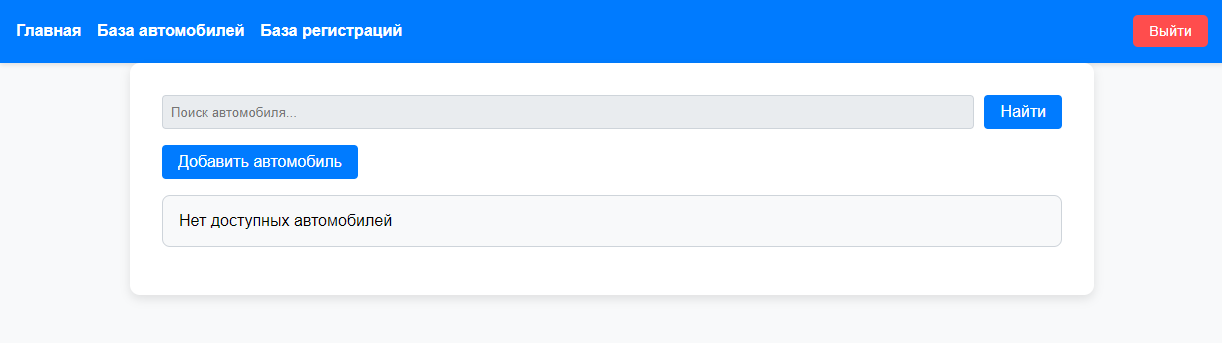
**Тест:**

1. Выполнить успешную авторизацию пользователя.
2. Перейти на страницу /static/html/cars.html.

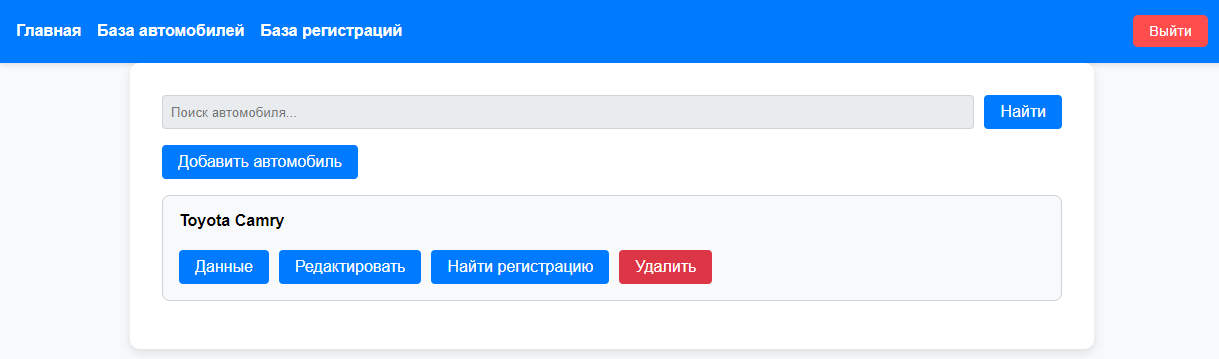
**Ожидаемый результат:**

* Список автомобилей загружается с сервера.
* Если списка автомобилей нет, отображается сообщение: "Нет доступных автомобилей."
* Если автомобили есть, они отображаются в виде списка с соответствующими атрибутами: марка, модель, номер.

**Полученный результат:**



* **Рис. 32:** Скриншот страницы без доступных автомобилей, где отображается сообщение "Нет доступных автомобилей".



* **Рис. 33:** Скриншот страницы со списком автомобилей.

**Заключение:**  
Функционал загрузки и отображения списка автомобилей работает корректно.

#### **Тест-кейс №2: Добавление нового автомобиля с валидными данными**

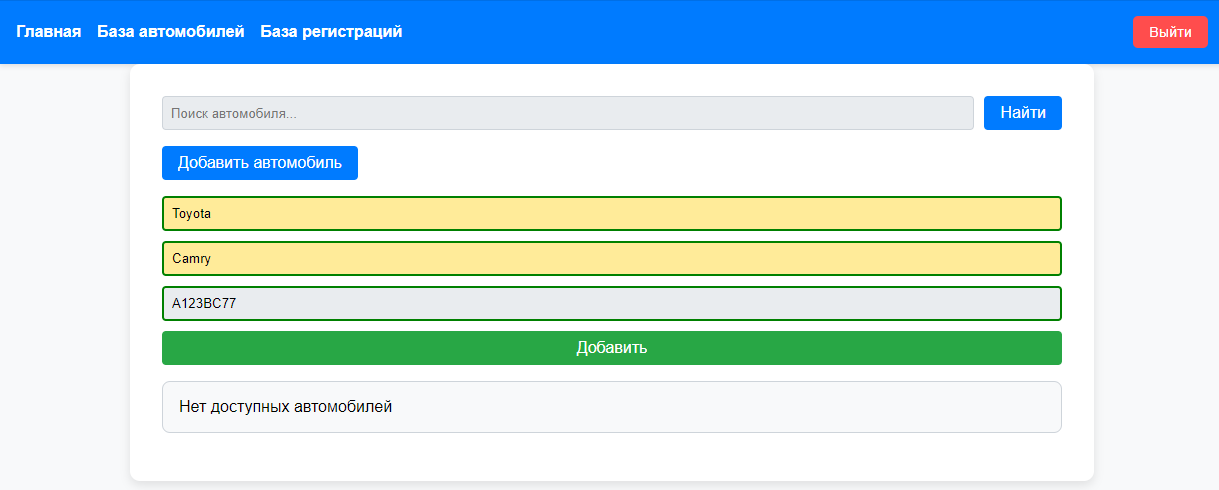
**Тест:**

1. Нажать кнопку "Добавить автомобиль".
2. Заполнить форму:
   * Марка: Toyota
   * Модель: Corolla
   * Номер: А123ВС77.
3. Нажать кнопку "Добавить".

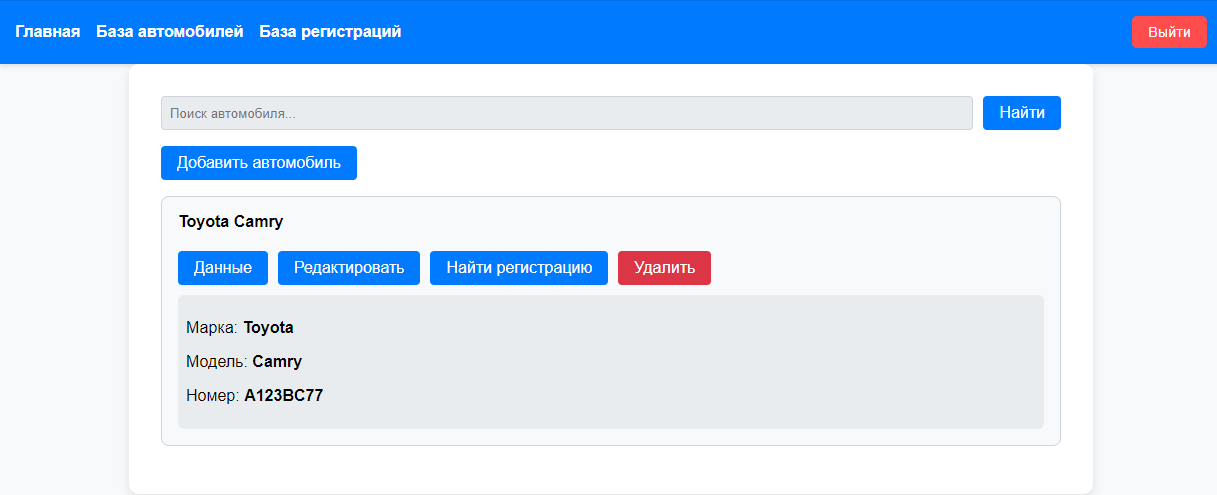
**Ожидаемый результат:**

* Код ответа сервера: 201 Created.
* Новый автомобиль добавлен в список.
* Форма закрывается, список обновляется с новым автомобилем.

**Полученный результат:**



* **Рис. 34:** Скриншот формы добавления автомобиля после ввода данных.



* **Рис. 35:** Скриншот обновлённого списка автомобилей с добавленным элементом.

**Заключение:**  
Функционал добавления автомобиля работает корректно.

#### **Тест-кейс №3: Добавление автомобиля с невалидными данными**

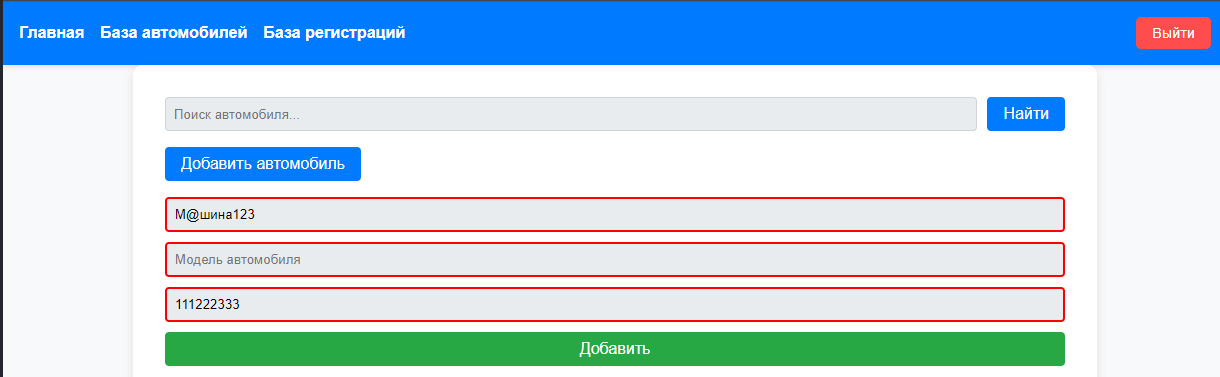
**Тест:**

1. Нажать кнопку "Добавить автомобиль".
2. Заполнить форму:
   * Марка: М@шина123 (некорректные символы).
   * Модель: пусто.
   * Номер: 111222333 (некорректный формат).
3. Нажать кнопку "Добавить".

**Ожидаемый результат:**

* Сообщение об ошибке: "Пожалуйста, исправьте ошибки в форме перед добавлением автомобиля."
* Поля с некорректными данными подсвечиваются (например, красной рамкой).

**Полученный результат:**



* **Рис. 36:** Скриншот формы с указанием ошибок (подсвеченные поля).

**Заключение:**  
Проверка полей и валидация данных работает корректно.

#### **Тест-кейс №4: Поиск автомобиля**

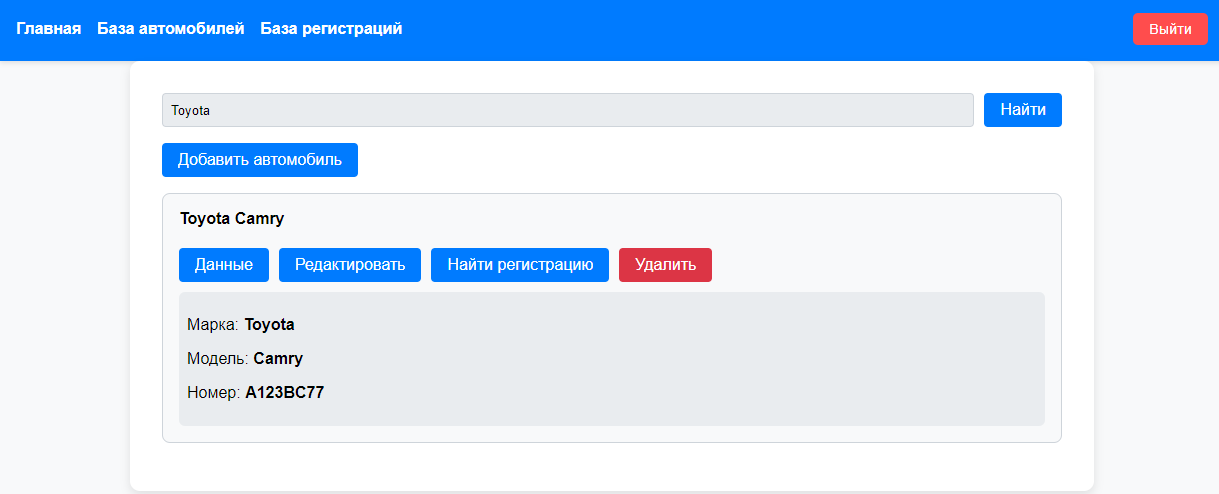
**Тест:**

1. Ввести запрос в поле "Поиск автомобиля..."
   * Запрос: Toyota.
2. Нажать кнопку "Найти".

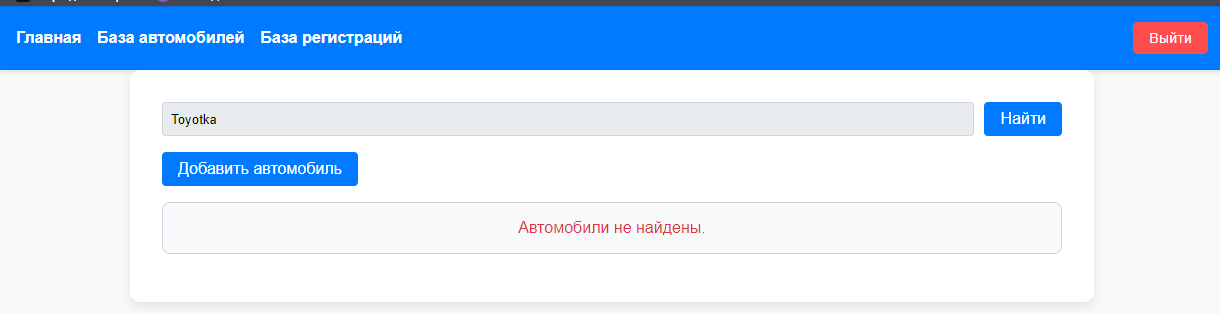
**Ожидаемый результат:**

* Отображаются только те записи, которые соответствуют поисковому запросу.
* Если результата нет, выводится сообщение: "Автомобили не найдены."

**Полученный результат:**



* **Рис. 37:** Скриншот результата успешного поиска.



* **Рис. 38:** Скриншот результата поиска без совпадений.

**Заключение:**  
Функционал поиска работает корректно.

#### **Тест-кейс №5: Редактирование данных автомобиля**

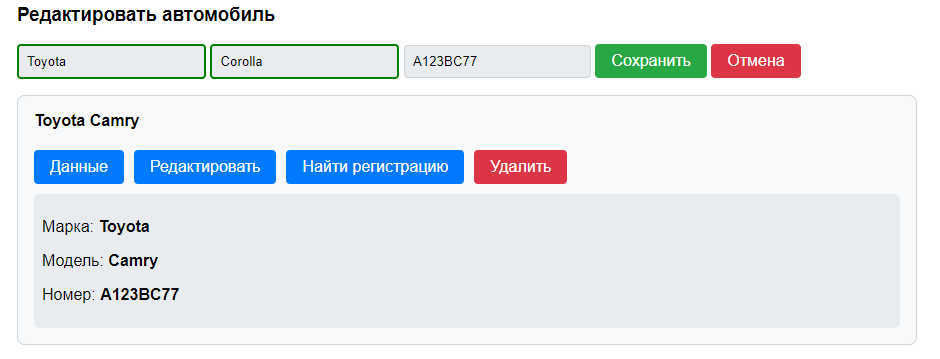
**Тест:**

1. Найти автомобиль в списке.
   * Марка: Toyota.
   * Модель: Corolla.
2. Нажать кнопку "Редактировать".
3. Изменить данные:
   * Марка: Toyota.
   * Модель: Camry.
4. Нажать кнопку "Сохранить".

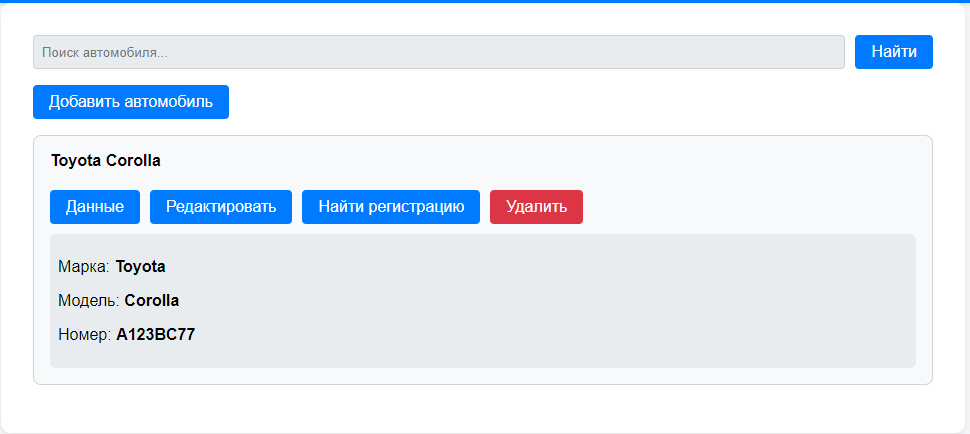
**Ожидаемый результат:**

* Код ответа сервера: 200 OK.
* Изменённые данные сохраняются. Список обновляется с обновлённым автомобилем.

**Полученный результат:**

****

* **Рис. 39:** Скриншот формы редактирования автомобиля.



* **Рис. 40:** Скриншот обновлённого списка автомобилей с изменёнными данными.

**Заключение:**  
Функционал редактирования данных автомобиля работает корректно.

#### **Тест-кейс №6: Удаление автомобиля**

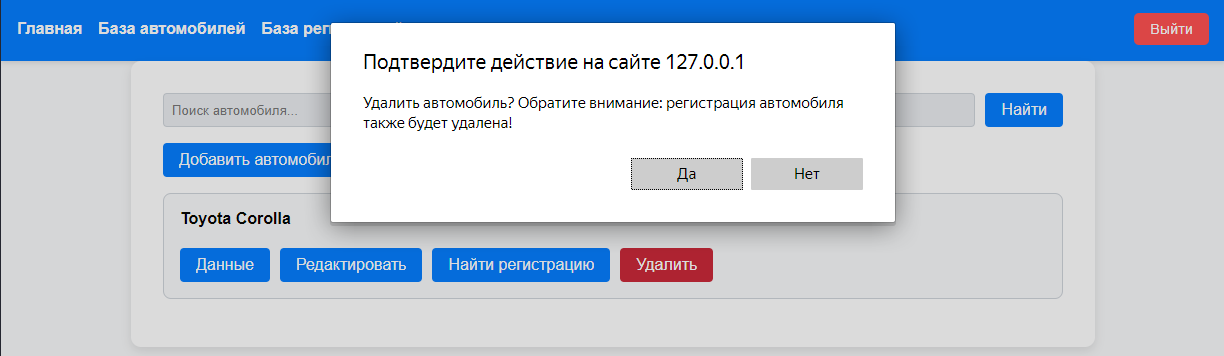
**Тест:**

1. Найти автомобиль в списке.
   * Марка: Toyota.
   * Модель: Camry.
2. Нажать кнопку "Удалить".
3. Подтвердить действие в появившемся диалоговом окне.

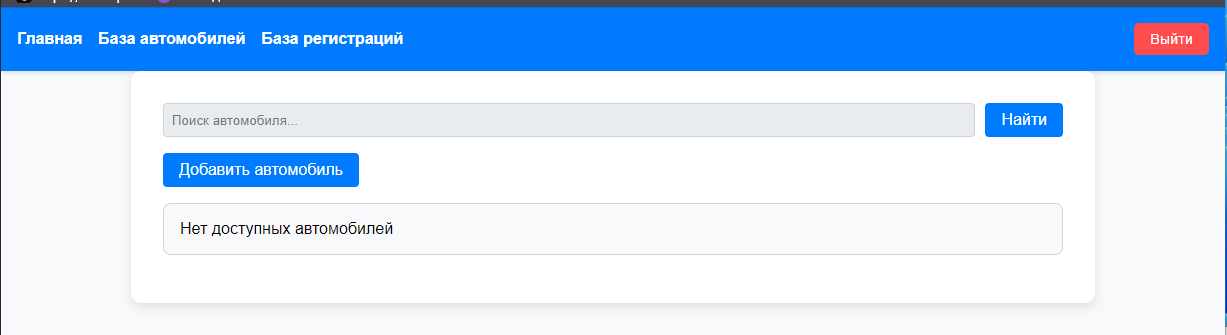
**Ожидаемый результат:**

* Код ответа сервера: 200 OK.
* Автомобиль удалён из списка. Список обновляется.
* Если у автомобиля есть регистрации, отображается сообщение: "Регистрация автомобиля также была удалена."

**Полученный результат:**



* **Рис. 41:** Скриншот диалогового окна с подтверждением удаления.



* **Рис. 42:** Скриншот обновлённого списка автомобилей с удалённой записью.

**Заключение:**  
Функционал удаления автомобиля и связанных регистраций работает корректно.

#### **Тест-кейс №7: Поиск регистрации для автомобиля**

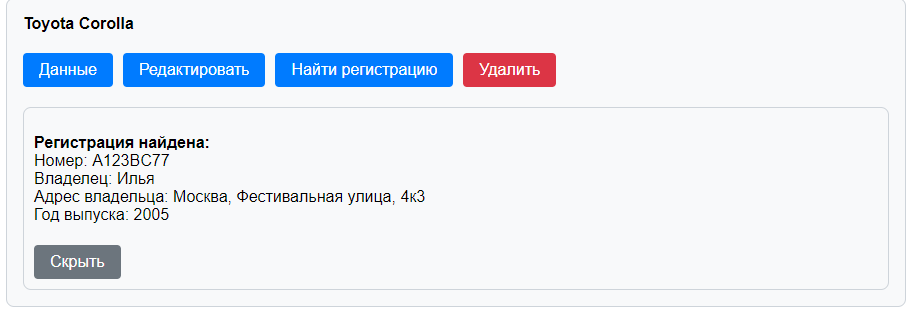
**Тест:**

1. Найти автомобиль в списке:
   * Марка: Toyota,
   * Модель: Corolla,
   * Номер: А123ВС77.
2. Нажать кнопку "Найти регистрацию" для этого автомобиля.

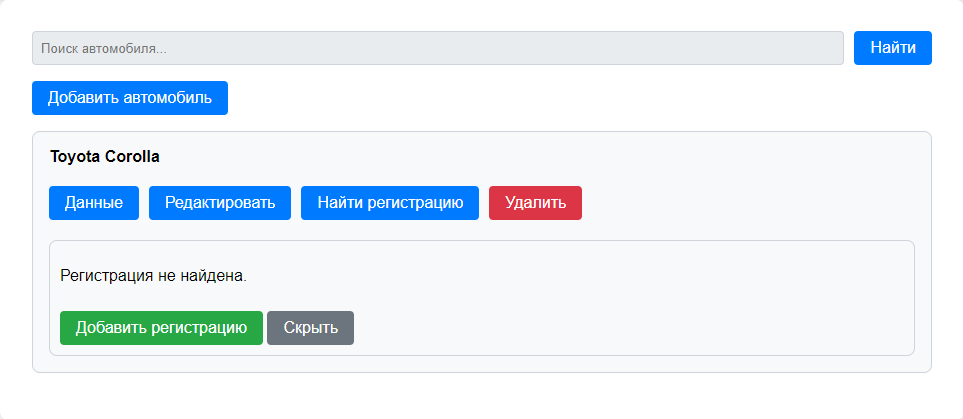
**Ожидаемый результат:**

* Отображается состояние загрузки (текст - "Загрузка...").
* Сервер отправляет запрос GET /regdb/search\_registrations/?query=А123ВС77.
* Если регистрация найдена:
  + Отображается информация о регистрации: номер, имя владельца, адрес владельца, год выпуска (на основании данных из сервера).
  + Кнопка "Добавить регистрацию" остаётся скрытой.
* Если регистрация не найдена:
  + Отображается текст: "Регистрация не найдена."
  + Кнопка "Добавить регистрацию" становится видимой.

**Полученный результат:**



* **Рис. 43**: Регистрация найдена (с данными владельца, адресом и годом выпуска).



* **Рис. 44**: Регистрация не найдена (кнопка "Добавить регистрацию" активна).

**Заключение:**  
Функция поиска регистрации работает корректно, отображает найденную регистрацию или сообщение об отсутствии регистрации.

#### **Тест-кейс №8: Добавление новой регистрации**

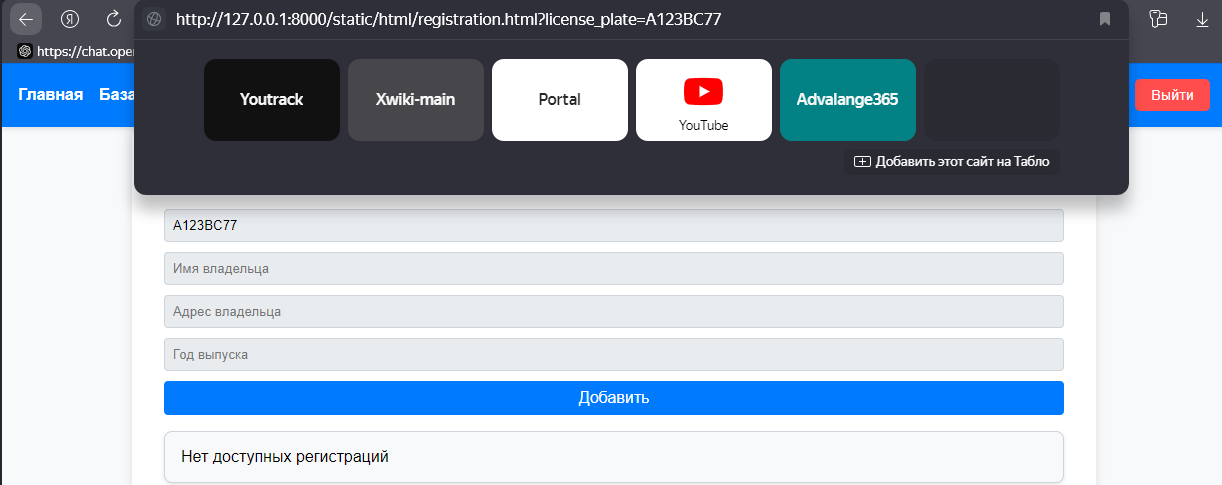
**Тест:**

1. Найти автомобиль в списке:
   * Марка: Toyota,
   * Модель: Corolla,
   * Номер: А123ВС77.
2. Нажать "Найти регистрацию".
3. Если регистрации для данного автомобиля нет, нажать кнопку "Добавить регистрацию".
4. Проверьте, что происходит перенаправление на страницу /static/html/registration.html, а в GET-параметрах передаётся номер автомобиля (license\_plate=А123ВС77).

**Ожидаемый результат:**

* Перенаправление на страницу добавления регистрации.
* В адресе отображается: /static/html/registration.html?license\_plate=А123ВС77.

**Полученный результат:**



* **Рис. 45**: Перенаправление на страницу добавления регистрации.
* URL содержит номер автомобиля.

**Заключение:**  
Функция добавления новой регистрации корректно инициируется после отсутствия текущей регистрации.

#### **Тест-кейс №9: Скрытие данных регистрации**

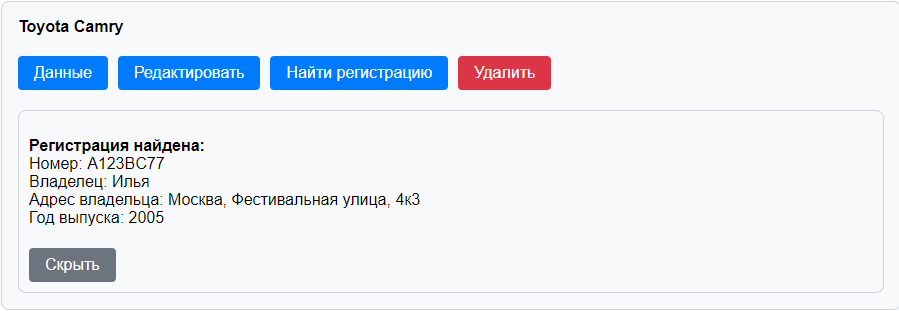
**Тест:**

1. Найти регистрацию для автомобиля.
2. После отображения данных регистрации нажать кнопку "Скрыть".

**Ожидаемый результат:**

* Данные о регистрации скрываются.
* Кнопки и поля регистрации также исчезают.

**Полученный результат:**



* **Рис. 46**: Данные успешно скрыты.

**Заключение:**  
Функция скрытия данных о регистрации функционирует корректно.

### **Тестирование страницы управления регистрациями**

#### **Тест-кейс №1: Загрузка списка регистраций**

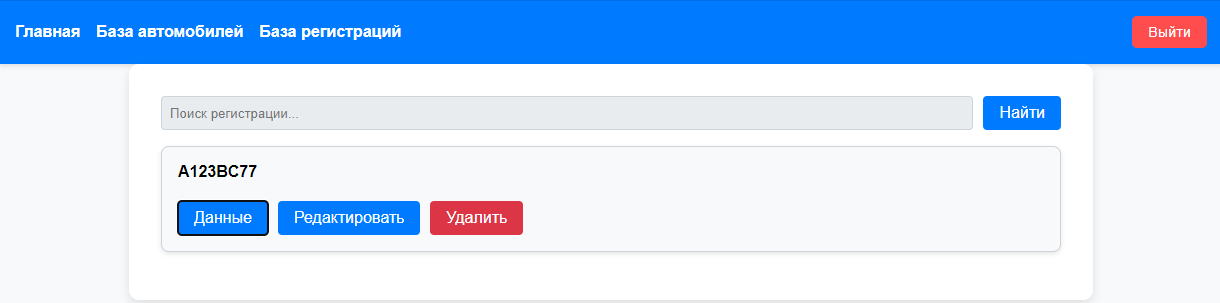
**Тест:**

1. Перейти на страницу регистрации /static/html/registration.html.
2. Оставить поле поиска пустым.
3. Проверить, отображаются ли все доступные регистрации.

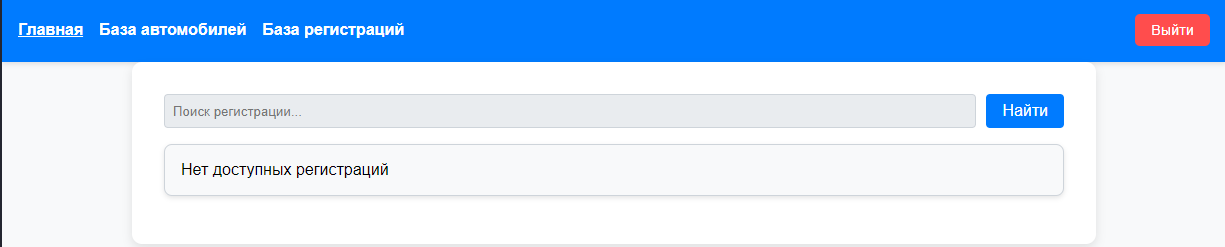
**Ожидаемый результат:**

* Элементы списка регистраций загружены с сервера API через запрос GET /regdb/get\_registrations/.
* Каждая регистрация включает номер, владельца, адрес владельца и год выпуска.
* Если регистраций нет, отображается сообщение "Нет доступных регистраций".

**Полученный результат:**



* **Рис. 47**: Список регистраций успешно загружен из API.



* **Рис. 48**: Сообщение о пустом списке регистраций при отсутствии данных.

**Заключение:**  
Функция загрузки регистраций работает корректно.

#### **Тест-кейс №2: Поиск регистраций**

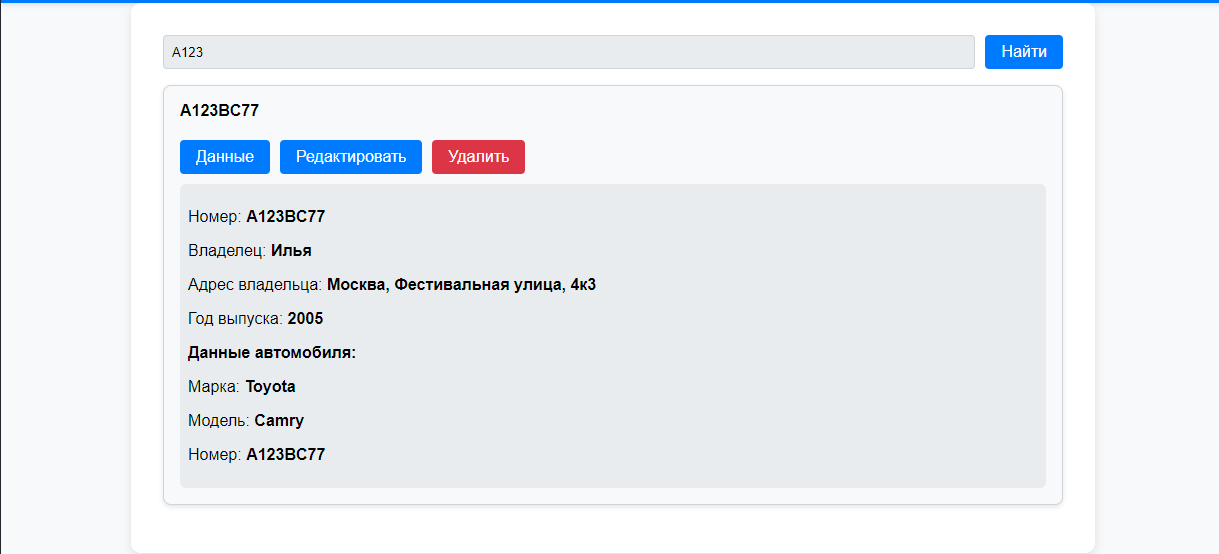
**Тест:**

1. Ввести строку поиска (например, А123В) в поле поиска регистраций.
2. Нажать кнопку "Найти".

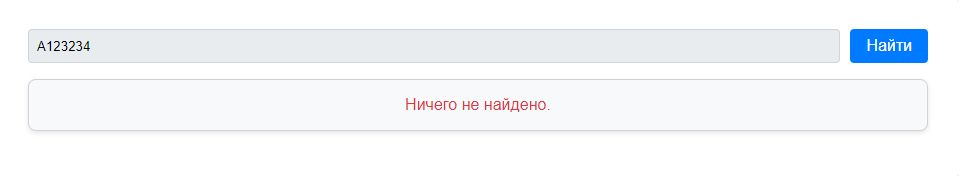
**Ожидаемый результат:**

* Отправляется запрос на API: GET /regdb/search\_registrations/?query=А123.
* Если регистрация найдена, она отображается в списке, другие скрываются.
* Если регистрация не найдена, отображается сообщение "Ничего не найдено."

**Полученный результат:**



* **Рис. 49**: Регистрация найдена и отображена.



* **Рис. 50**: Сообщение "Ничего не найдено" при отсутствии соответствий.

**Заключение:**  
Функция поиска регистраций работает правильно.

#### **Тест-кейс №3: Редактирование существующей регистрации**

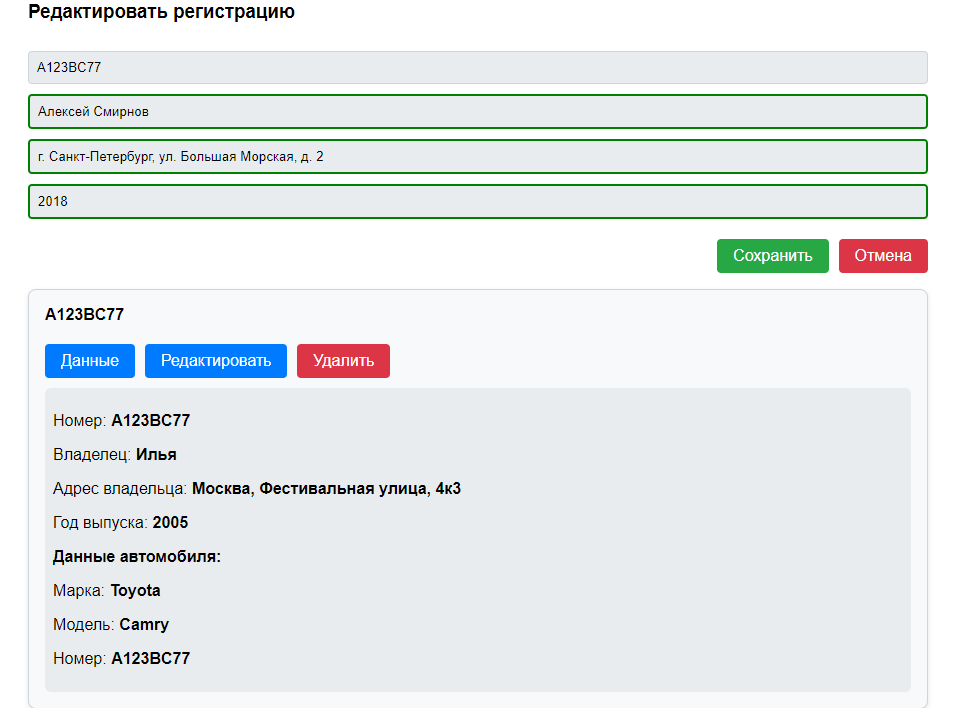
**Тест:**

1. Найти регистрацию (например, A123BC77).
2. Нажать кнопку "Редактировать".
3. Изменить данные:
   * Владелец: Алексей Смирнов,
   * Адрес: г. Санкт-Петербург, ул. Большая Морская, д. 2,
   * Год выпуска: 2018.
4. Нажать кнопку "Сохранить".

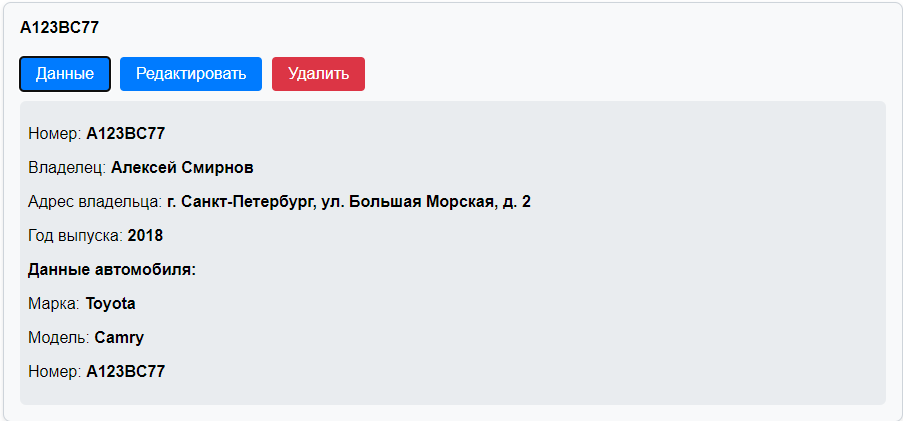
**Ожидаемый результат:**

* Валидация всех полей выполняется корректно.
* Отправляется запрос на API: PUT /regdb/update\_registration/A123BC77.
* Данные обновляются в списке регистраций без перезагрузки страницы.

**Полученный результат:**



* **Рис. 51**: Заполненная форма редактирования регистрации.



* **Рис. 52**: Обновленная запись в списке регистраций.

**Заключение:**  
Редактирование регистраций работает правильно.

#### **Тест-кейс №4: Удаление регистрации**

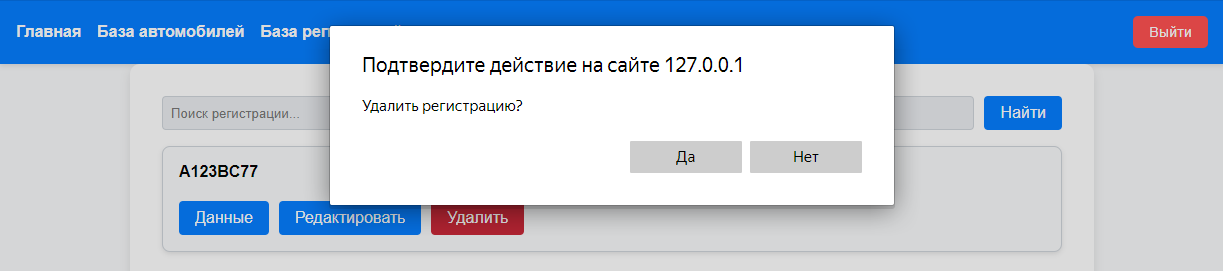
**Тест:**

1. Найти и выбрать регистрацию (например, A123BC77).
2. Нажать кнопку "Удалить".
3. Подтвердить удаление.

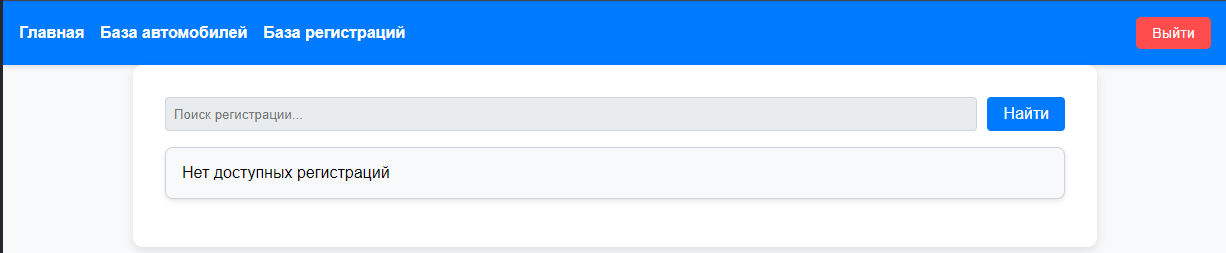
**Ожидаемый результат:**

* Отправляется запрос на API: DELETE /regdb/delete\_registration/A123BC77.
* Регистрация удаляется из списка без перезагрузки страницы.

**Полученный результат:**



* **Рис. 53**: Скриншот диалогового окна с подтверждением удаления.



* **Рис. 54**: Успешное удаление выбранной записи.

**Заключение:**  
Удаление регистраций функционирует корректно.

#### **Тест-кейс №5: Загрузка информации об автомобиле**

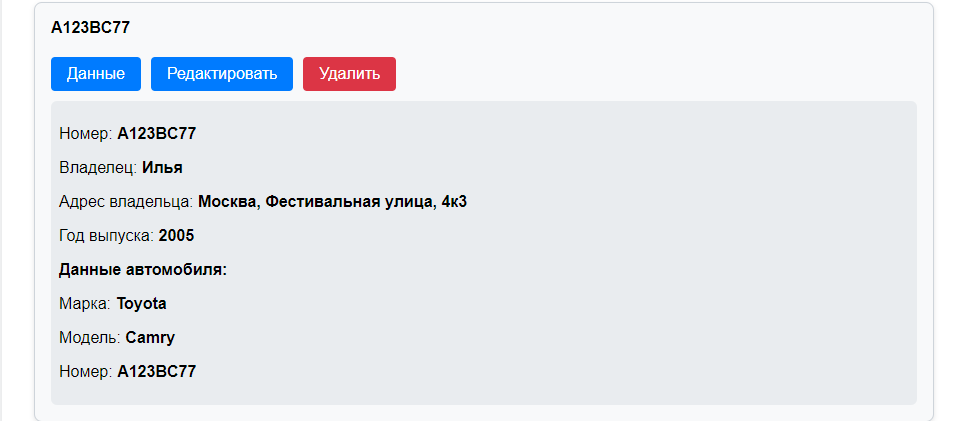
**Тест:**

1. Раскрыть регистрационные данные (нажать "Данные").
2. Проверить, отображается ли информация о связанном автомобиле.

**Ожидаемый результат:**

* Отправляется запрос к API: /carsdb/search\_cars/?query=A123BC77.
* Если автомобиль найден, отображается информация (марка, модель, номер).

**Полученный результат:**



* **Рис. 55**: Информация об автомобиле отобразилась корректно.

**Заключение:**  
Подгрузка информации о машине работает правильно.

# **6.Листинг программы**

## **/backend**

### **/app**

#### **/crud**

\_\_init\_\_.py

from . import auth\_crud  
from . import car\_crud  
from . import registration\_crud  
  
\_\_all\_\_: list[str] = [  
 "auth\_crud",  
 "car\_crud",  
 "registration\_crud",  
]

auth\_crud.py

from pymongo.errors import PyMongoError  
from database import users\_collection  
from models.user import UserCreate, UserLogin  
from security.hashing import hash\_password, verify\_password  
from security.jwt import create\_access\_token  
from typing import Dict  
  
  
async def register\_user\_crud(user: UserCreate) -> Dict[str, str]:  
 *"""  
 Регистрация нового пользователя.  
  
 E-mail должен быть уникальным в базе данных.  
  
 Args:  
 user (UserCreate): Данные нового пользователя.  
  
 Returns:  
 Dict[str, str]: Сообщение о статусе операции.  
  
 Raises:  
 ValueError: Если e-mail уже зарегистрирован.  
 RuntimeError: Если произошла ошибка базы данных или иная ошибка.  
 """* try:  
 *# Проверка существования пользователя с указанным e-mail* existing\_user = users\_collection.find\_one({"email": user.email})  
 if existing\_user:  
 raise ValueError("Email already registered")  
  
 *# Хэширование пароля и сохранение данных пользователя* hashed\_password = hash\_password(user.password)  
 user\_data = {  
 "first\_name": user.first\_name,  
 "last\_name": user.last\_name,  
 "email": user.email,  
 "hashed\_password": hashed\_password,  
 }  
 users\_collection.insert\_one(user\_data)  
 return {"msg": "User registered successfully"}  
 except ValueError as ve:  
 print(f"Validation error during registration: {ve}")  
 raise ve  
 except PyMongoError as pe:  
 print(f"Database error during registration: {pe}")  
 raise RuntimeError(  
 "Database error occurred while registering a user"  
 ) from pe  
 except Exception as e:  
 print(f"Unexpected error during registration: {e}")  
 raise RuntimeError(  
 "Unexpected error occurred during registration"  
 ) from e  
  
  
async def login\_user\_crud(user: UserLogin) -> Dict[str, str]:  
 *"""  
 Аутентификация пользователя.  
  
 Если авторизация успешна, возвращается JWT-токен.  
  
 Args:  
 user (UserLogin): Данные для входа пользователя.  
  
 Returns:  
 Dict[str, str]: JWT-токен и его тип.  
  
 Raises:  
 ValueError: При неверных учетных данных.  
 RuntimeError: При ошибке взаимодействия с базой данных.  
 """* try:  
 *# Поиск пользователя в базе данных по e-mail* db\_user = users\_collection.find\_one({"email": user.email})  
 if not db\_user or not verify\_password(  
 user.password, db\_user["hashed\_password"]  
 ):  
 raise ValueError("Invalid credentials")  
  
 *# Генерация JWT-токена* token = create\_access\_token({"sub": user.email})  
 return {"access\_token": token, "token\_type": "bearer"}  
 except ValueError as ve:  
 print(f"Validation error during login: {ve}")  
 raise ve  
 except PyMongoError as pe:  
 print(f"Database error during login: {pe}")  
 raise RuntimeError(  
 "Database error occurred while logging in"  
 ) from pe  
 except Exception as e:  
 print(f"Unexpected error during login: {e}")  
 raise RuntimeError(  
 "Unexpected error occurred during login"  
 ) from e

car\_crud.py

from typing import List, Dict  
from pymongo.errors import PyMongoError  
from database import car\_collection  
from models.car import Car  
  
  
async def update\_car\_by\_license\_plate(license\_plate: str, update\_data: Dict) -> bool:  
 *"""  
 Обновление автомобиля по номерному знаку (license\_plate).  
  
 Запрещено изменять ключ 'license\_plate' в update\_data.  
  
 Args:  
 license\_plate (str): Номерной знак автомобиля.  
 update\_data (Dict): Данные для обновления автомобиля.  
  
 Returns:  
 bool: True, если обновление завершено успешно, иначе False.  
  
 Raises:  
 ValueError: Если 'license\_plate' в update\_data или автомобиль отсутствует.  
 RuntimeError: При ошибке взаимодействия с базой данных.  
 """* try:  
 if "license\_plate" in update\_data:  
 raise ValueError("Updating license plate is not allowed")  
  
 result = car\_collection.update\_one(  
 {"license\_plate": license\_plate},  
 {"$set": update\_data}  
 )  
  
 if result.matched\_count == 0:  
 raise ValueError("Car not found")  
 return result.modified\_count > 0  
 except ValueError as ve:  
 print(f"Validation error during update: {ve}")  
 raise ve  
 except PyMongoError as pe:  
 print(f"Database error during update: {pe}")  
 raise RuntimeError("Database error occurred while updating the car") from pe  
 except Exception as e:  
 print(f"Unexpected error during update: {e}")  
 raise RuntimeError("Unexpected error occurred while updating the car") from e  
  
  
async def search\_cars(query: str) -> List[Car]:  
 *"""  
 Поиск автомобилей по запросу.  
  
 Args:  
 query (str): Запрос для поиска (марка, модель или номерной знак).  
  
 Returns:  
 List[Car]: Список объектов Car, соответствующих запросу.  
  
 Raises:  
 RuntimeError: При ошибке взаимодействия с базой данных.  
 """* try:  
 cars = car\_collection.find({  
 "$or": [  
 {"make": {"$regex": query, "$options": "i"}},  
 {"model": {"$regex": query, "$options": "i"}},  
 {"license\_plate": {"$regex": query, "$options": "i"}}  
 ]  
 })  
 return [Car(\*\*{\*\*car, "id": str(car["\_id"])}) for car in cars]  
 except PyMongoError as pe:  
 print(f"Database error during search: {pe}")  
 raise RuntimeError("Database error occurred while searching cars") from pe  
 except Exception as e:  
 print(f"Unexpected error during search: {e}")  
 raise RuntimeError("Unexpected error occurred while searching cars") from e  
  
  
async def add\_car(car: Car) -> None:  
 *"""  
 Добавление нового автомобиля.  
  
 Номерной знак (license\_plate) должен быть уникальным.  
  
 Args:  
 car (Car): Объект нового автомобиля.  
  
 Raises:  
 ValueError: Если автомобиль с таким номерным знаком уже существует.  
 RuntimeError: При ошибке взаимодействия с базой данных.  
 """* try:  
 if car\_collection.find\_one({"license\_plate": car.license\_plate}):  
 raise ValueError("Car with given license plate already exists")  
  
 car\_collection.insert\_one(car.dict())  
 except ValueError as ve:  
 print(f"Validation error during addition: {ve}")  
 raise ve  
 except PyMongoError as pe:  
 print(f"Database error during addition: {pe}")  
 raise RuntimeError("Database error occurred while adding a car") from pe  
 except Exception as e:  
 print(f"Unexpected error during addition: {e}")  
 raise RuntimeError("Unexpected error occurred while adding a car") from e  
  
  
async def delete\_car\_by\_license\_plate(license\_plate: str) -> bool:  
 *"""  
 Удаление автомобиля по номерному знаку (license\_plate).  
  
 Args:  
 license\_plate (str): Номерной знак автомобиля для удаления.  
  
 Returns:  
 bool: True, если удаление прошло успешно, иначе False.  
  
 Raises:  
 ValueError: Если автомобиль с указанным номерным знаком отсутствует.  
 RuntimeError: При ошибке взаимодействия с базой данных.  
 """* try:  
 result = car\_collection.delete\_one({"license\_plate": license\_plate})  
 if result.deleted\_count == 0:  
 raise ValueError("Car with given license plate not found")  
 return result.deleted\_count > 0  
 except ValueError as ve:  
 print(f"Validation error during deletion: {ve}")  
 raise ve  
 except PyMongoError as pe:  
 print(f"Database error during deletion: {pe}")  
 raise RuntimeError("Database error occurred while deleting the car") from pe  
 except Exception as e:  
 print(f"Unexpected error during deletion: {e}")  
 raise RuntimeError("Unexpected error occurred while deleting the car") from e  
  
  
async def get\_all\_cars() -> List[Car]:  
 *"""  
 Получение списка всех автомобилей.  
  
 Returns:  
 List[Car]: Список всех объектов Car.  
  
 Raises:  
 RuntimeError: При ошибке взаимодействия с базой данных.  
 """* try:  
 cars = car\_collection.find()  
 return [Car(\*\*{\*\*car, "id": str(car["\_id"])}) for car in cars]  
 except PyMongoError as pe:  
 print(f"Database error during fetching all cars: {pe}")  
 raise RuntimeError("Database error occurred while fetching all cars") from pe  
 except Exception as e:  
 print(f"Unexpected error during fetching all cars: {e}")  
 raise RuntimeError("Unexpected error occurred while fetching all cars") from e

registration\_crud.py

from database import registration\_collection  
from models.registration import Registration  
from typing import List, Dict, Union  
from pymongo.errors import PyMongoError  
  
  
async def update\_registration\_by\_license\_plate(  
 license\_plate: str, update\_data: Dict[str, Union[str, int]]  
) -> bool:  
 *"""  
 Редактировать данные регистрации по её номеру.  
  
 Args:  
 license\_plate (str): Номерной знак, соответствующий записи.  
 update\_data (Dict[str, Union[str, int]]): Данные для обновления  
 (кроме license\_plate).  
  
 Returns:  
 bool: Статус успешности изменения.  
  
 Raises:  
 ValueError: Если попытка изменить license\_plate или запись не найдена.  
 RuntimeError: Если произошла ошибка базы данных или иная ошибка.  
 """* try:  
 if "license\_plate" in update\_data:  
 raise ValueError("Updating license plate is not allowed")  
  
 result = registration\_collection.update\_one(  
 {"license\_plate": license\_plate},  
 {"$set": update\_data}  
 )  
 if result.matched\_count == 0:  
 raise ValueError("Registration not found")  
  
 return result.modified\_count > 0  
 except ValueError as ve:  
 print(f"Validation error during update: {ve}")  
 raise ve  
 except PyMongoError as pe:  
 print(f"Database error during update: {pe}")  
 raise RuntimeError("Database error occurred during update") from pe  
 except Exception as e:  
 print(f"Unexpected error during update: {e}")  
 raise RuntimeError("Unexpected error occurred during update") from e  
  
  
async def search\_registrations(query: str) -> List[Registration]:  
 *"""  
 Поиск регистраций в базе данных по заданному запросу.  
  
 Args:  
 query (str): Поисковый запрос.  
  
 Returns:  
 List[Registration]: Список найденных регистраций.  
  
 Raises:  
 RuntimeError: Если произошла ошибка базы данных или иная ошибка.  
 """* try:  
 registrations = registration\_collection.find({  
 "$or": [  
 {"license\_plate": {"$regex": query, "$options": "i"}},  
 {"owner\_name": {"$regex": query, "$options": "i"}},  
 {"owner\_address": {"$regex": query, "$options": "i"}}  
 ]  
 })  
 return [  
 Registration(\*\*{\*\*registration, "id": str(registration["\_id"])})  
 for registration in registrations  
 ]  
 except PyMongoError as pe:  
 print(f"Database error during search: {pe}")  
 raise RuntimeError(  
 "Database error occurred while searching registrations"  
 ) from pe  
 except Exception as e:  
 print(f"Unexpected error during search: {e}")  
 raise RuntimeError(  
 "Unexpected error occurred while searching registrations"  
 ) from e  
  
  
async def add\_registration(registration: Registration) -> None:  
 *"""  
 Добавление новой регистрации.  
  
 Args:  
 registration (Registration): Объект регистрации.  
  
 Raises:  
 ValueError: Если регистрация с таким license\_plate уже существует.  
 RuntimeError: Если произошла ошибка базы данных или иная ошибка.  
 """* try:  
 if registration\_collection.find\_one(  
 {"license\_plate": registration.license\_plate}  
 ):  
 raise ValueError("Registration with given license plate already exists")  
  
 registration\_collection.insert\_one(registration.dict())  
 except ValueError as ve:  
 print(f"Validation error during addition: {ve}")  
 raise ve  
 except PyMongoError as pe:  
 print(f"Database error during addition: {pe}")  
 raise RuntimeError(  
 "Database error occurred while adding registration"  
 ) from pe  
 except Exception as e:  
 print(f"Unexpected error during addition: {e}")  
 raise RuntimeError(  
 "Unexpected error occurred while adding registration"  
 ) from e  
  
  
async def get\_all\_registrations() -> List[Registration]:  
 *"""  
 Получение всех регистраций.  
  
 Returns:  
 List[Registration]: Список всех регистраций.  
  
 Raises:  
 RuntimeError: Если произошла ошибка базы данных или иная ошибка.  
 """* try:  
 registrations = registration\_collection.find()  
 return [  
 Registration(\*\*{\*\*registration, "id": str(registration["\_id"])})  
 for registration in registrations  
 ]  
 except PyMongoError as pe:  
 print(f"Database error during fetching registrations: {pe}")  
 raise RuntimeError(  
 "Database error occurred while fetching registrations"  
 ) from pe  
 except Exception as e:  
 print(f"Unexpected error during fetching registrations: {e}")  
 raise RuntimeError(  
 "Unexpected error occurred while fetching registrations"  
 ) from e  
  
  
async def delete\_registration\_by\_license\_plate(license\_plate: str) -> bool:  
 *"""  
 Удаление регистрации по номеру.  
  
 Args:  
 license\_plate (str): Номерной знак для удаления.  
  
 Returns:  
 bool: Статус успешности удаления.  
  
 Raises:  
 ValueError: Если запись с указанным номером не найдена.  
 RuntimeError: Если произошла ошибка базы данных или иная ошибка.  
 """* try:  
 result = registration\_collection.delete\_one({"license\_plate": license\_plate})  
 if result.deleted\_count == 0:  
 raise ValueError("Registration not found")  
  
 return result.deleted\_count > 0  
 except ValueError as ve:  
 print(f"Validation error during deletion: {ve}")  
 raise ve  
 except PyMongoError as pe:  
 print(f"Database error during deletion: {pe}")  
 raise RuntimeError(  
 "Database error occurred while deleting registration"  
 ) from pe  
 except Exception as e:  
 print(f"Unexpected error during deletion: {e}")  
 raise RuntimeError(  
 "Unexpected error occurred while deleting registration"  
 ) from e

#### **/models**

\_\_init\_\_.py

from . import user  
from . import car  
from . import registration  
  
\_\_all\_\_: list[str] =[  
 "user",  
 "car",  
 "registration",  
]

car.py

from pydantic import BaseModel, Field, field\_validator  
import re  
  
  
class Car(BaseModel):  
 *"""  
 Модель для представления автомобиля.  
  
 Attributes:  
 make (str): Марка автомобиля.  
 model (str): Модель автомобиля.  
 license\_plate (str): Номерной знак автомобиля.  
 """* make: str = Field(  
 ...,  
 min\_length=1,  
 max\_length=50,  
 description="Марка автомобиля (1–50 символов)"  
 )  
 model: str = Field(  
 ...,  
 min\_length=1,  
 max\_length=50,  
 description="Модель автомобиля (1–50 символов)"  
 )  
 license\_plate: str = Field(  
 ...,  
 description="Номерной знак автомобиля (строковый формат)"  
 )  
  
 @field\_validator("make")  
 def validate\_make(cls, value: str) -> str:  
 *"""  
 Проверяет, что марка автомобиля содержит только допустимые символы.  
  
 Args:  
 value (str): Значение поля make.  
  
 Returns:  
 str: Валидное значение поля make.  
  
 Raises:  
 ValueError: Если значение содержит недопустимые символы.  
 """* if not re.match(r"^[A-Za-z\s-]{1,50}$", value):  
 raise ValueError("Марка автомобиля содержит недопустимые символы.")  
 return value  
  
 @field\_validator("model")  
 def validate\_model(cls, value: str) -> str:  
 *"""  
 Проверяет, что модель автомобиля содержит только допустимые символы.  
  
 Args:  
 value (str): Значение поля model.  
  
 Returns:  
 str: Валидное значение поля model.  
  
 Raises:  
 ValueError: Если значение содержит недопустимые символы.  
 """* if not re.match(r"^[A-Za-z0-9\s-]{1,50}$", value):  
 raise ValueError("Модель автомобиля содержит недопустимые символы.")  
 return value  
  
 @field\_validator("license\_plate")  
 def validate\_license\_plate(cls, value: str) -> str:  
 *"""  
 Проверяет, что номерной знак автомобиля имеет валидный формат.  
  
 Args:  
 value (str): Значение поля license\_plate.  
  
 Returns:  
 str: Валидное значение поля license\_plate.  
  
 Raises:  
 ValueError: Если значение не соответствует формату номерного знака.  
 """* if not re.match(r"^[A-Z0-9]{3,10}$", value):  
 raise ValueError("Номерной знак имеет некорректный формат.")  
 return value

registration.py

from datetime import datetime  
from pydantic import BaseModel, Field, field\_validator  
import re  
  
  
class Registration(BaseModel):  
 *"""  
 Модель регистрации транспортного средства.  
  
 Атрибуты:  
 license\_plate (str): Номерной знак автомобиля.  
 Формат - заглавные буквы, цифры и/или дефисы. Максимум 10 символов.  
 owner\_name (str): Имя владельца автомобиля.  
 От 1 до 50 символов. Допускаются буквы, пробелы и дефисы.  
 owner\_address (str): Адрес владельца автомобиля.  
 От 1 до 100 символов. Допускаются буквы, цифры, пробелы, точки, запятые и другие стандартные символы адреса.  
 year\_of\_manufacture (int): Год выпуска автомобиля.  
 Должен быть не меньше 1900 года и не превышать текущий год.  
 """* license\_plate: str = Field(  
 ...,  
 pattern=r"^[A-Z0-9-]{1,10}$",  
 description="Номерной знак автомобиля (до 10 символов)",  
 )  
 owner\_name: str = Field(  
 ...,  
 min\_length=1,  
 max\_length=50,  
 description="Имя владельца авто (от 1 до 50 символов)",  
 )  
 owner\_address: str = Field(  
 ...,  
 min\_length=1,  
 max\_length=100,  
 description="Адрес владельца (от 1 до 100 символов)",  
 )  
 year\_of\_manufacture: int = Field(  
 ...,  
 ge=1900,  
 le=datetime.now().year,  
 description="Год выпуска автомобиля (от 1900 до текущего года)",  
 )  
  
 @field\_validator("owner\_name")  
 def validate\_owner\_name(cls, value: str) -> str:  
 *"""  
 Валидация имени владельца.  
  
 Args:  
 value (str): Имя для проверки.  
  
 Returns:  
 str: Прошедшее валидацию имя.  
  
 Raises:  
 ValueError: Если имя содержит недопустимые символы.  
 """* if not re.match(r"^[a-zA-Zа-яА-ЯёЁ\s\-]+$", value):  
 raise ValueError(  
 "Имя может содержать только буквы, пробелы и дефисы."  
 )  
 return value  
  
 @field\_validator("owner\_address")  
 def validate\_owner\_address(cls, value: str) -> str:  
 *"""  
 Валидация адреса владельца.  
  
 Args:  
 value (str): Адрес для проверки.  
  
 Returns:  
 str: Прошедший валидацию адрес.  
  
 Raises:  
 ValueError: Если адрес содержит недопустимые символы.  
 """* if not re.match(r"^[А-Яа-яЁёA-Za-z0-9\s.,\-\\/]+$", value):  
 raise ValueError(  
 "Адрес может содержать только буквы, цифры и символы: ., - \/"  
 )  
 return value  
  
 @field\_validator("license\_plate")  
 def validate\_license\_plate(cls, value: str) -> str:  
 *"""  
 Валидация номерного знака автомобиля.  
  
 Args:  
 value (str): Номерной знак для проверки.  
  
 Returns:  
 str: Прошедший валидацию номерной знак.  
  
 Raises:  
 ValueError: Если номерной знак содержит недопустимые символы.  
 """* pattern = r"^[A-Z0-9-]{1,10}$"  
 if not re.match(pattern, value):  
 raise ValueError(  
 "Номерной знак должен содержать только заглавные буквы, цифры и/или дефисы (до 10 символов)."  
 )  
 return value

user.py

from pydantic import BaseModel, EmailStr, Field  
  
  
class UserCreate(BaseModel):  
 *"""  
 Модель для создания нового пользователя.  
  
 Атрибуты:  
 first\_name (str): Имя пользователя. От 2 до 50 символов.  
 last\_name (str): Фамилия пользователя. От 2 до 50 символов.  
 email (EmailStr): E-mail пользователя.  
 password (str): Пароль пользователя. Минимум 6 символов.  
 confirm\_password (str): Подтверждение пароля. Минимум 6 символов.  
 """* first\_name: str = Field(..., min\_length=2, max\_length=50, description="Имя пользователя (2-50 символов)")  
 last\_name: str = Field(..., min\_length=2, max\_length=50, description="Фамилия пользователя (2-50 символов)")  
 email: EmailStr = Field(..., description="E-mail пользователя")  
 password: str = Field(..., min\_length=6, description="Пароль пользователя, минимум 6 символов")  
 confirm\_password: str = Field(..., min\_length=6, description="Подтверждение пароля, минимум 6 символов")  
  
 class Config:  
 orm\_mode = True  
  
  
class UserLogin(BaseModel):  
 *"""  
 Модель для входа пользователя.  
  
 Атрибуты:  
 email (EmailStr): E-mail пользователя.  
 password (str): Пароль пользователя.  
 """* email: EmailStr = Field(..., description="E-mail пользователя")  
 password: str = Field(..., description="Пароль пользователя")

#### **/routes**

\_\_init\_\_.py

from . import auth\_routes  
from . import car\_routes  
from . import registration\_routes  
  
\_\_all\_\_: list[str] = [  
 "auth\_routes",  
 "car\_routes",  
 "registration\_routes",  
]

auth\_routes.py

from fastapi import APIRouter, HTTPException, Depends, Form  
from crud.auth\_crud import register\_user\_crud, login\_user\_crud  
from models.user import UserCreate, UserLogin  
from security.jwt import decode\_access\_token  
from fastapi.security import OAuth2PasswordBearer  
import traceback  
  
router = APIRouter()  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="auth/login")  
  
  
@router.post(  
 "/register",  
 responses={  
 200: {"description": "User was successfully registered."},  
 400: {"description": "Validation error (e.g., passwords mismatch or email already registered)."},  
 500: {"description": "Unexpected error during registration."},  
 },  
)  
async def register\_user(user: UserCreate) -> dict:  
 *"""  
 Регистрация нового пользователя (E-mail уникален).  
  
 Args:  
 user (UserCreate): Данные создаваемого пользователя.  
  
 Returns:  
 dict: Сообщение об успешной регистрации.  
  
 Raises:  
 HTTPException: Если пароли не совпадают или возникли другие ошибки.  
 """* if user.password != user.confirm\_password:  
 raise HTTPException(status\_code=400, detail="Passwords do not match")  
  
 try:  
 return await register\_user\_crud(user)  
 except ValueError as ve:  
 raise HTTPException(status\_code=400, detail=str(ve))  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
 except Exception:  
 print(f"Unexpected error during registration: {traceback.format\_exc()}")  
 raise HTTPException(status\_code=500, detail="Unexpected error during registration")  
  
  
@router.post(  
 "/login",  
 responses={  
 200: {"description": "User authenticated successfully. JWT token returned."},  
 401: {"description": "Invalid credentials provided."},  
 500: {"description": "Unexpected error during login."},  
 },  
)  
async def login\_user(  
 username: str = Form(..., description="User's email"), *# Используем Form для обработки form-data* password: str = Form(..., description="User's password"),  
 grant\_type: str = Form(default="password", description="OAuth2 grant type, default is 'password'"),  
) -> dict:  
 *"""  
 Аутентификация пользователя с использованием OAuth2 формата (username вместо email).  
  
 Args:  
 username (str): E-mail пользователя.  
 password (str): Пароль пользователя.  
 grant\_type (str): Тип гранта для OAuth2 (по умолчанию "password").  
  
 Returns:  
 dict: Токен доступа в случае успешного входа.  
  
 Raises:  
 HTTPException: Если учетные данные некорректны или произошла другая ошибка.  
 """* try:  
 *# Используем username как email* user\_data = UserLogin(email=username, password=password)  
 return await login\_user\_crud(user\_data)  
 except ValueError as ve:  
 raise HTTPException(status\_code=401, detail=str(ve))  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
 except Exception:  
 print(f"Unexpected error during login: {traceback.format\_exc()}")  
 raise HTTPException(status\_code=500, detail="Unexpected error during login")  
  
  
@router.get(  
 "/protected",  
 responses={  
 200: {"description": "Access granted to the protected route."},  
 401: {"description": "Invalid or expired JWT token."},  
 500: {"description": "Unexpected error during token verification."},  
 },  
)  
async def protected\_route(token: str = Depends(oauth2\_scheme)) -> dict:  
 *"""  
 Защищенный маршрут, требует валидного токена.  
  
 Args:  
 token (str): JWT-токен пользователя, переданный через заголовок.  
  
 Returns:  
 dict: Приветственное сообщение для пользователя.  
  
 Raises:  
 HTTPException: Если токен недействителен, истек или произошла другая ошибка.  
 """* try:  
 payload = decode\_access\_token(token)  
 if not payload:  
 raise HTTPException(status\_code=401, detail="Invalid or expired token")  
 return {"msg": f"Hello, {payload.get('sub')}!"}  
 except HTTPException as he:  
 raise he  
 except Exception:  
 print(f"Unexpected error during token decoding: {traceback.format\_exc()}")  
 raise HTTPException(status\_code=500, detail="Unexpected error occurred during token verification")

car\_routes.py

from fastapi import APIRouter, Depends, HTTPException  
from fastapi.security import OAuth2PasswordBearer  
from models.car import Car  
from crud.car\_crud import (  
 add\_car,  
 delete\_car\_by\_license\_plate,  
 get\_all\_cars,  
 search\_cars,  
 update\_car\_by\_license\_plate,  
)  
from security.jwt import decode\_access\_token  
from typing import Dict, Any  
  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="/auth/login")  
  
  
async def get\_current\_user(token: str = Depends(oauth2\_scheme)) -> str:  
 *"""  
 Получение текущего пользователя на основе переданного токена.  
  
 Args:  
 token (str): OAuth2 токен пользователя.  
  
 Returns:  
 str: Идентификатор пользователя.  
  
 Raises:  
 HTTPException: Если токен недействителен или истек.  
 """* payload = decode\_access\_token(token)  
 if not payload:  
 raise HTTPException(status\_code=401, detail="Invalid or expired token")  
 return payload.get("sub")  
  
  
router = APIRouter()  
  
  
@router.put(  
 "/update\_car/{license\_plate}",  
 responses={  
 200: {"description": "Car was successfully updated"},  
 400: {  
 "description": "Validation error (e.g., license plate modification not allowed)"  
 },  
 404: {"description": "Car not found or no updates applied"},  
 500: {"description": "Unexpected error during car update"},  
 },  
)  
async def update\_car\_view(  
 license\_plate: str, update\_data: Dict[str, Any], user: str = Depends(get\_current\_user)  
) -> Dict[str, str]:  
 *"""  
 Редактирование данных автомобиля. Номер автомобиля редактировать запрещено.  
  
 Args:  
 license\_plate (str): Номерной знак автомобиля.  
 update\_data (Dict[str, Any]): Данные для обновления.  
 user (str): ID текущего пользователя (из токена).  
  
 Returns:  
 Dict[str, str]: Сообщение об успешном обновлении автомобиля.  
  
 Raises:  
 HTTPException: При возникновении ошибки.  
 """* try:  
 updated = await update\_car\_by\_license\_plate(license\_plate, update\_data)  
 if not updated:  
 raise HTTPException(  
 status\_code=404, detail="Car not found or no updates applied"  
 )  
 return {"message": f"Car updated successfully by {user}"}  
 except ValueError as ve:  
 raise HTTPException(status\_code=400, detail=str(ve))  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
  
  
@router.get(  
 "/search\_cars/",  
 responses={  
 200: {"description": "Cars searched successfully. Results returned."},  
 500: {"description": "Unexpected error occurred while searching cars"},  
 },  
)  
async def search\_cars\_view(query: str, user: str = Depends(get\_current\_user)) -> Dict[str, Any]:  
 *"""  
 Поиск автомобилей по марке, модели или номерному знаку.  
  
 Args:  
 query (str): Запрос для поиска.  
 user (str): ID текущего пользователя (из токена).  
  
 Returns:  
 Dict[str, Any]: Найденные автомобили и данные текущего пользователя.  
  
 Raises:  
 HTTPException: При возникновении ошибки.  
 """* try:  
 cars = await search\_cars(query)  
 if not cars:  
 return {"cars": [], "message": "No cars found matching the query."}  
 return {"cars": cars, "user": user}  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
  
  
@router.post(  
 "/add\_car/",  
 responses={  
 200: {"description": "Car successfully added"},  
 400: {"description": "Validation error (e.g., duplicate license plate)"},  
 500: {"description": "Unexpected error during car addition"},  
 },  
)  
async def add\_car\_view(car: Car, user: str = Depends(get\_current\_user)) -> Dict[str, str]:  
 *"""  
 Добавление автомобиля.  
  
 Args:  
 car (Car): Данные добавляемого автомобиля.  
 user (str): ID текущего пользователя (из токена).  
  
 Returns:  
 Dict[str, str]: Сообщение об успешном добавлении автомобиля.  
  
 Raises:  
 HTTPException: При возникновении ошибки.  
 """* try:  
 await add\_car(car)  
 return {"message": f"Car added successfully by {user}"}  
 except ValueError as ve:  
 raise HTTPException(status\_code=400, detail=str(ve))  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
  
  
@router.delete(  
 "/delete\_car/{license\_plate}",  
 responses={  
 200: {"description": "Car successfully deleted"},  
 404: {"description": "Car not found"},  
 500: {"description": "Unexpected error during car deletion"},  
 },  
)  
async def delete\_car\_view(license\_plate: str, user: str = Depends(get\_current\_user)) -> Dict[str, str]:  
 *"""  
 Удаление автомобиля.  
  
 Args:  
 license\_plate (str): Номерной знак удаляемого автомобиля.  
 user (str): ID текущего пользователя (из токена).  
  
 Returns:  
 Dict[str, str]: Сообщение об успешном удалении автомобиля.  
  
 Raises:  
 HTTPException: При возникновении ошибки.  
 """* try:  
 await delete\_car\_by\_license\_plate(license\_plate)  
 return {"message": f"Car deleted successfully by {user}"}  
 except ValueError as ve:  
 raise HTTPException(status\_code=404, detail=str(ve))  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
  
  
@router.get(  
 "/get\_cars/",  
 responses={  
 200: {"description": "All cars successfully retrieved"},  
 500: {"description": "Unexpected error during cars retrieval"},  
 },  
)  
async def get\_cars(user: str = Depends(get\_current\_user)) -> Dict[str, Any]:  
 *"""  
 Получение всех автомобилей.  
  
 Args:  
 user (str): ID текущего пользователя (из токена).  
  
 Returns:  
 Dict[str, Any]: Список всех автомобилей и данные пользователя.  
  
 Raises:  
 HTTPException: При возникновении ошибки.  
 """* try:  
 cars = await get\_all\_cars()  
 return {"cars": cars, "user": user}  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))

registration\_routes.py

from fastapi import APIRouter, Depends, HTTPException  
from typing import Dict, Any  
from models.registration import Registration  
from crud.registration\_crud import (  
 add\_registration,  
 get\_all\_registrations,  
 delete\_registration\_by\_license\_plate,  
 search\_registrations,  
 update\_registration\_by\_license\_plate,  
)  
from security.jwt import decode\_access\_token  
from fastapi.security import OAuth2PasswordBearer  
  
oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="/auth/login")  
  
router = APIRouter()  
  
  
async def get\_current\_user(token: str = Depends(oauth2\_scheme)) -> str:  
 *"""  
 Получение текущего пользователя на основе переданного токена.  
  
 Args:  
 token (str): OAuth2 токен пользователя.  
  
 Returns:  
 str: Идентификатор пользователя.  
  
 Raises:  
 HTTPException: Если токен недействителен или истек.  
 """* payload = decode\_access\_token(token)  
 if not payload:  
 raise HTTPException(status\_code=401, detail="Invalid or expired token")  
 return payload.get("sub")  
  
  
@router.put(  
 "/update\_registration/{license\_plate}",  
 responses={  
 200: {"description": "Registration was successfully updated"},  
 400: {"description": "Validation error (e.g., license plate modification not allowed)"},  
 404: {"description": "Registration not found or no updates applied"},  
 500: {"description": "Unexpected error during registration update"},  
 },  
)  
async def update\_registration\_view(  
 license\_plate: str,  
 update\_data: Dict[str, Any],  
 user: str = Depends(get\_current\_user),  
) -> Dict[str, str]:  
 *"""  
 Обновление данных регистрации.  
  
 Args:  
 license\_plate (str): Номерной знак, идентифицирующий регистрацию.  
 update\_data (Dict[str, Any]): Данные для обновления регистрации.  
 user (str): Текущий пользователь, извлеченный из токена (определяется через Depends).  
  
 Returns:  
 Dict[str, str]: Сообщение о статусе обновления.  
  
 Raises:  
 HTTPException: Если регистрация не найдена, есть ошибки валидации или произошла ошибка на сервере.  
 """* try:  
 updated = await update\_registration\_by\_license\_plate(license\_plate, update\_data)  
 if not updated:  
 raise HTTPException(  
 status\_code=404, detail="Registration not found or no updates applied"  
 )  
 return {"message": f"Registration updated successfully by {user}"}  
 except ValueError as ve:  
 raise HTTPException(status\_code=400, detail=str(ve))  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
  
  
@router.get(  
 "/search\_registrations/",  
 responses={  
 200: {"description": "Registrations searched successfully. Results returned."},  
 500: {"description": "Unexpected error during registrations search"},  
 },  
)  
async def search\_registrations\_view(  
 query: str, user: str = Depends(get\_current\_user)  
) -> Dict[str, Any]:  
 *"""  
 Поиск регистраций.  
  
 Args:  
 query (str): Запрос для поиска регистраций.  
 user (str): Текущий пользователь, извлеченный из токена (определяется через Depends).  
  
 Returns:  
 Dict[str, Any]: Найденные регистрации и сообщение о статусе поиска.  
  
 Raises:  
 HTTPException: Если произошла ошибка на сервере.  
 """* try:  
 registrations = await search\_registrations(query)  
 if not registrations:  
 return {"registrations": [], "message": "No registrations found matching the query."}  
 return {"registrations": registrations}  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
  
  
@router.post(  
 "/add\_registration/",  
 responses={  
 200: {"description": "Registration successfully added"},  
 400: {"description": "Validation error (e.g., duplicate license plate)"},  
 500: {"description": "Unexpected error during registration addition"},  
 },  
)  
async def add\_registration\_view(  
 registration: Registration, user: str = Depends(get\_current\_user)  
) -> Dict[str, str]:  
 *"""  
 Добавление новой регистрации.  
  
 Args:  
 registration (Registration): Данные регистрации.  
 user (str): Текущий пользователь, извлеченный из токена (определяется через Depends).  
  
 Returns:  
 Dict[str, str]: Сообщение о статусе добавления.  
  
 Raises:  
 HTTPException: Если есть ошибка валидации или произошла ошибка на сервере.  
 """* try:  
 await add\_registration(registration)  
 return {"message": f"Registration added successfully by {user}"}  
 except ValueError as ve:  
 raise HTTPException(status\_code=400, detail=str(ve))  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
  
  
@router.get(  
 "/get\_registrations/",  
 responses={  
 200: {"description": "All registrations successfully retrieved"},  
 500: {"description": "Unexpected error during registrations retrieval"},  
 },  
)  
async def get\_registrations(user: str = Depends(get\_current\_user)) -> Dict[str, Any]:  
 *"""  
 Получение всех регистраций.  
  
 Args:  
 user (str): Текущий пользователь, извлеченный из токена (определяется через Depends).  
  
 Returns:  
 Dict[str, Any]: Список всех регистраций.  
  
 Raises:  
 HTTPException: Если произошла ошибка на сервере.  
 """* try:  
 registrations = await get\_all\_registrations()  
 return {"registrations": registrations, "user": user}  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))  
  
  
@router.delete(  
 "/delete\_registration/{license\_plate}",  
 responses={  
 200: {"description": "Registration successfully deleted"},  
 404: {"description": "Registration not found"},  
 500: {"description": "Unexpected error during registration deletion"},  
 },  
)  
async def delete\_registration\_view(  
 license\_plate: str, user: str = Depends(get\_current\_user)  
) -> Dict[str, str]:  
 *"""  
 Удаление регистрации.  
  
 Args:  
 license\_plate (str): Номерной знак, идентифицирующий регистрацию.  
 user (str): Текущий пользователь, извлеченный из токена (определяется через Depends).  
  
 Returns:  
 Dict[str, str]: Сообщение о статусе удаления.  
  
 Raises:  
 HTTPException: Если регистрация не найдена или произошла ошибка на сервере.  
 """* try:  
 await delete\_registration\_by\_license\_plate(license\_plate)  
 return {"message": f"Registration deleted successfully by {user}"}  
 except ValueError as ve:  
 raise HTTPException(status\_code=404, detail=str(ve))  
 except RuntimeError as re:  
 raise HTTPException(status\_code=500, detail=str(re))

#### **/security**

\_\_init\_\_.py

from . import hashing  
from . import jwt  
  
\_\_all\_\_: list[str] = [  
 "hashing",  
 "jwt",  
]

hashing.py

from passlib.context import CryptContext  
  
pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")  
  
  
def hash\_password(password: str) -> str:  
 *"""  
 Хэширование пароля.  
  
 Args:  
 password (str): Пароль в виде обычной строки для хэширования.  
  
 Returns:  
 str: Хэшированный пароль.  
 """* return pwd\_context.hash(password)  
  
  
def verify\_password(plain\_password: str, hashed\_password: str) -> bool:  
 *"""  
 Сравнение введенного пароля с хэшированным.  
  
 Args:  
 plain\_password (str): Введенный пароль в виде обычной строки.  
 hashed\_password (str): Хэшированный пароль.  
  
 Returns:  
 bool: True, если пароли совпадают, иначе False.  
 """* return pwd\_context.verify(plain\_password, hashed\_password)

jwt.py

from typing import Optional  
from jose import JWTError, jwt  
from datetime import datetime, timedelta  
  
SECRET\_KEY: str = "keykeykey" *# Пусть будет так))*ALGORITHM: str = "HS256"  
ACCESS\_TOKEN\_EXPIRE\_MINUTES: int = 30  
  
  
def create\_access\_token(data: dict) -> str:  
 *"""  
 Создание JWT-токена доступа.  
  
 Args:  
 data (dict): Данные, которые нужно закодировать в токен.  
  
 Returns:  
 str: Сформированный JWT-токен.  
 """* to\_encode = data.copy()  
 expire = datetime.utcnow() + timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)  
 to\_encode.update({"exp": expire})  
 token = jwt.encode(to\_encode, SECRET\_KEY, algorithm=ALGORITHM)  
 return token  
  
  
def decode\_access\_token(token: str) -> Optional[dict]:  
 *"""  
 Расшифровка и валидация JWT-токена.  
  
 Args:  
 token (str): Токен доступа, который нужно расшифровать.  
  
 Returns:  
 Optional[dict]: Расшифрованные данные из токена, если токен валиден.  
 Возвращает None, если в процессе валидации токен оказался невалидным.  
 """* try:  
 payload = jwt.decode(token, SECRET\_KEY, algorithms=[ALGORITHM])  
 return payload  
 except JWTError:  
 return None

#### **database.py**

from pymongo import MongoClient  
  
*# Создание клиента MongoDB*client: MongoClient = MongoClient("mongodb://mongo:27017")  
  
*# Инициализация базы данных и коллекций*db = client.car\_database  
car\_collection = db.cars  
registration\_collection = db.registrations  
users\_collection = db.users

#### **main.py**

import os  
from fastapi import FastAPI  
from fastapi.staticfiles import StaticFiles  
from fastapi.responses import HTMLResponse  
from fastapi.middleware.cors import CORSMiddleware  
from routes import car\_routes, registration\_routes, auth\_routes  
  
app: FastAPI = FastAPI()  
  
*# Разрешенные источники для CORS*origins: list[str] = [  
 "http://127.0.0.1:8000",  
 "http://localhost:8000"  
]  
  
*# Добавление промежуточного слоя для обработки CORS*app.add\_middleware(  
 CORSMiddleware,  
 allow\_origins=origins,  
 allow\_credentials=True,  
 allow\_methods=["\*"],  
 allow\_headers=["\*"],  
)  
  
*# Подключение роутеров*app.include\_router(  
 car\_routes.router,  
 prefix="/carsdb",  
 tags=["carsdb"]  
)  
app.include\_router(  
 registration\_routes.router,  
 prefix="/regdb",  
 tags=["regdb"]  
)  
app.include\_router(  
 auth\_routes.router,  
 prefix="/auth",  
 tags=["auth"]  
)  
  
*# Подключение статических файлов*app.mount(  
 "/static",  
 StaticFiles(directory="frontend"),  
 name="static"  
)  
  
  
@app.get("/", response\_class=HTMLResponse)  
async def read\_root() -> HTMLResponse:  
 *"""  
 Обработчик для корневого маршрута.  
  
 Returns:  
 HTMLResponse: Содержимое HTML-файла главной страницы.  
 """* index\_path: str = os.path.join("frontend", "html", "index.html")  
 try:  
 with open(index\_path, "r", encoding="utf-8") as html\_file:  
 content: str = html\_file.read()  
 return HTMLResponse(content)  
 except FileNotFoundError:  
 return HTMLResponse(  
 content="Главная HTML-страница не найдена.",  
 status\_code=404  
 )

### **Dockerfile**

FROM python:3.10-slim  
  
WORKDIR /app  
  
COPY requirements.txt .  
  
RUN pip install --no-cache-dir -r requirements.txt  
  
COPY ./app /app  
  
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

## **/frontend**

### **/css**

#### **cars.css**

body {  
 font-family: Arial, sans-serif;  
 margin: 0;  
 padding: 0;  
 background-color: #f8f9fa;  
}  
  
main {  
 padding: 2rem;  
 max-width: 900px;  
 margin: 0 auto;  
 background: #ffffff;  
 border-radius: 10px;  
 box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);  
}  
  
  
.search-container {  
 display: flex;  
 gap: 10px;  
 margin-bottom: 1rem;  
}  
  
input[type="text"] {  
 padding: 0.5rem;  
 flex: 1;  
 border: 1px solid #ced4da;  
 border-radius: 4px;  
 background-color: #e9ecef;  
}  
  
  
button {  
 padding: 0.5rem 1rem;  
 background-color: #007BFF;  
 color: white;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 1rem;  
 transition: background-color 0.3s ease;  
}  
  
button:hover {  
 background-color: #0056b3;  
}  
  
button.add-reg,  
button.hide-reg,  
button.find-reg {  
 background-color: #28a745;  
}  
  
button.add-reg:hover,  
button.hide-reg:hover,  
button.find-reg:hover {  
 background-color: #218838;  
}  
  
button.delete {  
 background-color: #dc3545;  
}  
  
button.delete:hover {  
 background-color: #c82333;  
}  
  
button.expand,  
button.find-reg {  
 background-color: #007BFF;  
 color: white;  
}  
  
button.expand:hover,  
button.find-reg:hover {  
 background-color: #0056b3;  
}  
  
.hidden {  
 display: none !important;  
}  
  
button.edit-car {  
 background-color: #007BFF;  
 color: white;  
}  
  
button.edit-car:hover {  
 background-color: #0056b3;  
}  
  
#addCarForm {  
 margin: 1rem 0;  
 display: flex;  
 flex-direction: column;  
 gap: 10px;  
}  
  
#addCarForm.hidden {  
 display: none;  
}  
  
ul {  
 list-style: none;  
 padding: 0;  
}  
  
li {  
 background-color: #f8f9fa;  
 margin: 0.5rem 0;  
 padding: 1rem;  
 border: 1px solid #ced4da;  
 border-radius: 8px;  
 position: relative;  
 display: flex;  
 flex-direction: column;  
 gap: 10px;  
}  
  
.car-details {  
 padding: 0.5rem;  
 background-color: #e9ecef;  
 border-radius: 6px;  
}  
  
.car-details.hidden {  
 display: none;  
}  
  
  
button#submitAddCar,  
button#submitEditCar {  
 background-color: #28a745; */\* Зелёный цвет \*/* color: white;  
}  
  
button#submitAddCar:hover,  
button#submitEditCar:hover {  
 background-color: #218838; */\* Тёмно-зелёный при наведении \*/*}  
  
button#cancelEditCar {  
 background-color: #dc3545; */\* Красный цвет \*/* color: white;  
}  
  
button#cancelEditCar:hover {  
 background-color: #c82333; */\* Тёмно-красный при наведении \*/*}  
  
.reg-result {  
 margin-top: 10px;  
 background-color: #f8f9fa;  
 padding: 10px;  
 border: 1px solid #ced4da;  
 border-radius: 8px;  
}  
  
.reg-result.hidden {  
 display: none;  
}  
  
button.add-reg {  
 margin-top: 10px;  
}  
  
button.hide-reg {  
 margin-top: 5px;  
 background-color: #6c757d;  
 color: white;  
}  
  
button.hide-reg:hover {  
 background-color: #5a6268;  
}  
  
  
.no-results {  
 font-size: 1rem;  
 color: #dc3545;  
 margin-top: 1rem;  
 text-align: center;  
}  
  
.button-row {  
 display: flex;  
 gap: 10px; */\* Расстояние между кнопками \*/* justify-content: start; */\* Выравнивание кнопок в начале строки \*/* margin-top: 10px; */\* Расстояние от предыдущего контента \*/*}  
  
  
input {  
 border: 1px solid #ced4da;  
 border-radius: 4px;  
 padding: 0.5rem;  
 transition: border-color 0.3s ease;  
}  
  
input:focus {  
 outline: none;  
 border-color: #007BFF;  
}  
  
input[style\*="border: 2px solid red"] {  
 border-color: red !important;  
}  
  
input[style\*="border: 2px solid green"] {  
 border-color: green !important;  
}

#### **header.css**

header {  
 background-color: #007BFF;  
 padding: 1rem;  
 color: white;  
 display: flex;  
 justify-content: space-between;  
 align-items: center;  
 box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);  
}  
  
nav {  
 display: flex;  
 align-items: center;  
 width: 100%;  
}  
  
nav a {  
 color: white;  
 text-decoration: none;  
 margin-right: 1rem;  
 font-size: 1rem;  
 font-weight: bold;  
}  
  
nav a:hover {  
 text-decoration: underline;  
}  
  
nav .auth-buttons {  
 margin-left: auto;  
 display: flex;  
 align-items: center;  
}  
  
#login-button, #register-button {  
 padding: 0.5rem 1rem;  
 background-color: #28a745;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 font-size: 0.9rem;  
 margin-right: 10px;  
}  
  
#login-button:hover, #register-button:hover {  
 background-color: #218838;  
}  
  
#logout-button {  
 padding: 0.5rem 1rem;  
 background-color: #ff4d4d;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 font-size: 0.9rem;  
}  
  
#logout-button:hover {  
 background-color: #cc0000;  
}

#### **index.css**

body {  
 font-family: Arial, sans-serif;  
 margin: 0;  
 padding: 0;  
 background-color: #f4f4f9;  
 color: #333;  
}  
  
.welcome {  
 text-align: center;  
 margin-top: 5rem;  
}  
  
.actions a.btn {  
 margin: 1rem;  
 padding: 0.5rem 1rem;  
 background-color: #007BFF;  
 color: white;  
 text-decoration: none;  
 border-radius: 5px;  
}  
  
.actions a.btn:hover {  
 background-color: #0056b3;  
}

#### **login.css**

body {  
 font-family: Arial, sans-serif;  
 background-color: #f4f4f9;  
}  
  
main {  
 max-width: 400px;  
 margin: 5rem auto;  
 padding: 2rem;  
 background: white;  
 border-radius: 8px;  
 box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);  
}  
  
form {  
 display: flex;  
 flex-direction: column;  
}  
  
form label {  
 margin: 0.5rem 0;  
}  
  
form input {  
 padding: 0.5rem;  
 margin-bottom: 1rem;  
}  
  
form button {  
 padding: 0.5rem;  
 background-color: #007BFF;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
}  
  
form button:hover {  
 background-color: #0056b3;  
}

#### **register.css**

body {  
 font-family: Arial, sans-serif;  
 background-color: #f4f4f9;  
}  
  
main {  
 max-width: 400px;  
 margin: 5rem auto;  
 padding: 2rem;  
 background: white;  
 border-radius: 8px;  
 box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);  
}  
  
form {  
 display: flex;  
 flex-direction: column;  
}  
  
form label {  
 margin: 0.5rem 0;  
}  
  
form input {  
 padding: 0.5rem;  
 margin-bottom: 1rem;  
}  
  
form button {  
 padding: 0.5rem;  
 background-color: #007BFF;  
 color: white;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
}  
  
form button:hover {  
 background-color: #0056b3;  
}

#### **registration.css**

body {  
 font-family: Arial, sans-serif;  
 margin: 0;  
 padding: 0;  
 background-color: #f8f9fa;  
}  
main {  
 padding: 2rem;  
 max-width: 900px;  
 margin: 0 auto;  
 background: #ffffff;  
 border-radius: 10px;  
 box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);  
}  
.search-container {  
 display: flex;  
 gap: 10px;  
 margin-bottom: 1rem;  
}  
input[type="text"], input[type="number"] {  
 padding: 0.5rem;  
 flex: 1;  
 border: 1px solid #ced4da;  
 border-radius: 4px;  
 background-color: #e9ecef;  
}  
button {  
 padding: 0.5rem 1rem;  
 background-color: #007BFF;  
 color: white;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 1rem;  
 transition: background-color 0.3s ease;  
}  
  
button.find-car {  
 background-color: #007BFF; */\* Синий \*/* color: white;  
 padding: 0.5rem 1rem;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 1rem;  
 transition: background-color 0.3s ease;  
}  
  
button.find-car:hover {  
 background-color: #0056b3; */\* Темно-синий при наведении \*/*}  
  
button:hover {  
 background-color: #0056b3;  
}  
button.add-reg,  
button.submit-reg,  
button.hide-reg {  
 background-color: #28a745;  
}  
button.add-reg:hover,  
button.submit-reg:hover,  
button.hide-reg:hover {  
 background-color: #218838;  
}  
button.delete {  
 background-color: #dc3545;  
}  
button.delete:hover {  
 background-color: #c82333;  
}  
button.expand {  
 background-color: #007BFF;  
 color: white;  
}  
button.expand:hover {  
 background-color: #0056b3;  
}  
.hidden {  
 display: none !important;  
}  
ul {  
 list-style: none;  
 padding: 0;  
}  
li {  
 background-color: #f8f9fa;  
 margin: 0.5rem 0;  
 padding: 1rem;  
 border: 1px solid #ced4da;  
 border-radius: 8px;  
 box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);  
 position: relative;  
 display: flex;  
 flex-direction: column;  
 gap: 10px;  
}  
.reg-details {  
 padding: 0.5rem;  
 background-color: #e9ecef;  
 border-radius: 6px;  
}  
.reg-details.hidden {  
 display: none;  
}  
.button-row {  
 display: flex;  
 gap: 10px;  
 justify-content: start;  
 margin-top: 10px;  
}  
#addRegForm,  
#editRegForm {  
 margin: 1rem 0;  
 display: flex;  
 flex-direction: column;  
 gap: 10px;  
}  
#addRegForm.hidden,  
#editRegForm.hidden {  
 display: none;  
}  
#addRegForm input,  
#editRegForm input {  
 padding: 0.5rem;  
 border: 1px solid #ced4da;  
 border-radius: 4px;  
 background-color: #e9ecef;  
}  
#addRegForm button,  
#editRegForm button {  
 padding: 0.5rem 1rem;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 1rem;  
 transition: background-color 0.3s ease;  
}  
#editRegForm button#submitEditReg {  
 background-color: #28a745;  
 color: white;  
}  
#editRegForm button#submitEditReg:hover {  
 background-color: #218838;  
}  
#editRegForm button#cancelEditReg {  
 background-color: #dc3545;  
 color: white;  
}  
#editRegForm button#cancelEditReg:hover {  
 background-color: #c82333;  
}  
.car-result {  
 margin-top: 10px;  
 padding: 10px;  
 border: 1px solid #ced4da;  
 border-radius: 8px;  
 background-color: #f8f9fa;  
 box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);  
 position: relative;  
}  
.car-result.hidden {  
 display: none;  
}  
.car-result .car-info {  
 font-size: 1rem;  
 color: #343a40;  
 line-height: 1.5;  
}  
.car-result .hide-car-result {  
 display: block;  
 margin-top: 10px;  
 padding: 0.5rem 1rem;  
 background-color: #6c757d;  
 color: white;  
 border: none;  
 border-radius: 4px;  
 font-size: 1rem;  
 cursor: pointer;  
 transition: background-color 0.3s ease;  
 text-align: center;  
}  
.car-result .hide-car-result:hover {  
 background-color: #5a6268;  
}  
.no-results {  
 font-size: 1rem;  
 color: #dc3545;  
 margin-top: 1rem;  
 text-align: center;  
}  
  
.edit-buttons {  
 display: flex;  
 justify-content: flex-end;  
 gap: 10px;  
 margin-top: 10px;  
}  
  
#submitEditReg {  
 background-color: #28a745;  
 color: white;  
 padding: 0.5rem 1rem;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 1rem;  
 transition: background-color 0.3s ease;  
}  
  
#submitEditReg:hover {  
 background-color: #218838;  
}  
  
#cancelEditReg {  
 background-color: #dc3545;  
 color: white;  
 padding: 0.5rem 1rem;  
 border: none;  
 border-radius: 4px;  
 cursor: pointer;  
 font-size: 1rem;  
 transition: background-color 0.3s ease;  
}  
  
#cancelEditReg:hover {  
 background-color: #c82333;  
}  
  
input {  
 border: 1px solid #ced4da;  
 border-radius: 4px;  
 padding: 0.5rem;  
 transition: border-color 0.3s ease;  
}  
  
input[style\*="border: 2px solid red"] {  
 border-color: red !important;  
}  
  
input[style\*="border: 2px solid green"] {  
 border-color: green !important;  
}

### **/html**

#### **cars.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>База автомобилей</title>  
 <link rel="stylesheet" href="/static/css/cars.css">  
</head>  
<body>  
 <main>  
 <div class="search-container">  
 <input type="text" id="searchCarInput" placeholder="Поиск автомобиля...">  
 <button id="searchCarButton">Найти</button>  
 </div>  
  
 <button id="addCarButton">Добавить автомобиль</button>  
 <div id="addCarForm" class="hidden">  
 <input  
 type="text"  
 id="newCarMake"  
 placeholder="Марка автомобиля"  
 title="Только буквы, пробелы и дефисы. Например: Toyota"  
 pattern="[A-Za-zА-Яа-яЁё\s-]{1,50}"  
 required>  
 <input  
 type="text"  
 id="newCarModel"  
 placeholder="Модель автомобиля"  
 title="Только буквы, цифры, пробелы и дефисы. Например: Corolla 2023"  
 pattern="[A-Za-zА-Яа-яЁё0-9\s-]{1,50}"  
 required>  
 <input  
 type="text"  
 id="newCarLicensePlate"  
 placeholder="Номер автомобиля"  
 title="Формат: А777АА777"  
 pattern="[А-ЯЁ]{1}\d{3}[А-ЯЁ]{2}\d{2,3}"  
 required>  
 <button id="submitAddCar">Добавить</button>  
 </div>  
  
 <div id="editCarForm" class="hidden">  
 <h3>Редактировать автомобиль</h3>  
 <input  
 type="text"  
 id="editCarMake"  
 placeholder="Марка автомобиля"  
 title="Только буквы, пробелы и дефисы. Например: Toyota"  
 pattern="[A-Za-zА-Яа-яЁё\s-]{1,50}"  
 required>  
 <input  
 type="text"  
 id="editCarModel"  
 placeholder="Модель автомобиля"  
 title="Только буквы, цифры, пробелы и дефисы. Например: Corolla 2023"  
 pattern="[A-Za-zА-Яа-яЁё0-9\s-]{1,50}"  
 required>  
 <input  
 type="text"  
 id="editCarLicensePlate"  
 placeholder="Номер автомобиля"  
 title="Формат: А777АА777"  
 pattern="[А-ЯЁ]{1}\d{3}[А-ЯЁ]{2}\d{2,3}"  
 readonly>  
 <button id="submitEditCar">Сохранить</button>  
 <button id="cancelEditCar">Отмена</button>  
 </div>  
  
 <ul id="carsList"></ul>  
 </main>  
  
 <script src="/static/js/cars.js"></script>  
 <script src="/static/js/header.js"></script>  
</body>  
</html>

#### **header.html**

<header>  
 <link rel="stylesheet" href="/static/css/header.css">  
 <nav>  
 <a href="/static/html/index.html">Главная</a>  
 <a href="/static/html/cars.html">База автомобилей</a>  
 <a href="/static/html/registration.html">База регистраций</a>  
 <div class="auth-buttons">  
 <a href="/static/html/login.html" id="login-button" style="display: none;">Войти</a>  
 <a href="/static/html/register.html" id="register-button" style="display: none;">Зарегистрироваться</a>  
 <button id="logout-button" style="display: none;">Выйти</button>  
 </div>  
 </nav>  
</header>

#### **index.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Система хранения данных</title>  
 <link rel="stylesheet" href="/static/css/index.css">  
</head>  
<body>  
 <main>  
 <section class="welcome">  
 <h1>Система хранения данных авто</h1>  
 <p>Только для профессионального использования</p>  
 </section>  
 </main>  
 <script src="/static/js/index.js"></script>  
 <script src="/static/js/header.js"></script>  
</body>  
</html>

#### **login.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Авторизация</title>  
 <link rel="stylesheet" href="/static/css/login.css">  
</head>  
<body>  
 <main>  
 <h1>Авторизация</h1>  
 <form id="login-form">  
 <label for="email">Почта</label>  
 <input type="email" id="email" name="email" required><br>  
  
 <label for="password">Пароль</label>  
 <input type="password" id="password" name="password" required><br>  
  
 <button type="submit">Войти</button>  
 </form>  
 <p>Нет аккаунта? <a href="/static/html/register.html">Зарегистрируйтесь</a>.</p>  
 </main>  
 <script src="/static/js/login.js"></script>  
 <script src="/static/js/header.js"></script>  
</body>  
</html>

#### **register.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>Регистрация</title>  
 <link rel="stylesheet" href="/static/css/register.css">  
</head>  
<body>  
 <main>  
 <h1>Регистрация</h1>  
 <form id="registration-form">  
 <label for="first\_name">Имя</label>  
 <input type="text" id="first\_name" name="first\_name" required><br>  
  
 <label for="last\_name">Фамилия</label>  
 <input type="text" id="last\_name" name="last\_name" required><br>  
  
 <label for="email">Почта</label>  
 <input type="email" id="email" name="email" required><br>  
  
 <label for="password">Пароль</label>  
 <input type="password" id="password" name="password" required><br>  
  
 <label for="confirm\_password">Подтверждение пароля</label>  
 <input type="password" id="confirm\_password" name="confirm\_password" required><br>  
  
 <button type="submit">Зарегистрироваться</button>  
 </form>  
 <p>Уже есть аккаунт? <a href="/static/html/login.html">Авторизуйтесь</a>.</p>  
 </main>  
 <script src="/static/js/register.js"></script>  
 <script src="/static/js/header.js"></script>  
</body>  
</html>

#### **registration.html**

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <title>База регистраций</title>  
 <link rel="stylesheet" href="/static/css/registration.css">  
</head>  
<body>  
 <main>  
 <div class="search-container">  
 <input type="text" id="searchRegInput" placeholder="Поиск регистрации...">  
 <button id="searchRegButton">Найти</button>  
 </div>  
  
 <div id="addRegForm" class="hidden">  
 <input  
 type="text"  
 id="newRegLicensePlate"  
 placeholder="Номер автомобиля"  
 title="Формат: А777АА777"  
 pattern="[А-ЯЁA-Z]{1}[0-9]{3}[А-ЯЁA-Z]{2}[0-9]{2,3}"  
 readonly>  
 <input  
 type="text"  
 id="newOwnerName"  
 placeholder="Имя владельца"  
 title="Только буквы, пробелы и дефисы. Например: Иван Иванов"  
 pattern="[A-Za-zА-ЯЁа-яё\s-]{1,50}"  
 required>  
 <input  
 type="text"  
 id="newOwnerAddress"  
 placeholder="Адрес владельца"  
 title="Адрес может содержать буквы, цифры, пробелы, запятые и точки"  
 pattern="[A-Za-zА-ЯЁа-яё0-9\s.,-]+"  
 required>  
 <input  
 type="number"  
 id="newYearOfManufacture"  
 placeholder="Год выпуска"  
 title="Введите год в диапазоне от 1900 до текущего"  
 min="1900"  
 max="2024"  
 required>  
 <button id="submitAddReg">Добавить</button>  
 </div>  
 <div id="editRegForm" class="hidden">  
 <h3>Редактировать регистрацию</h3>  
 <input  
 type="text"  
 id="editRegLicensePlate"  
 placeholder="Номер автомобиля"  
 readonly>  
 <input  
 type="text"  
 id="editOwnerName"  
 placeholder="Имя владельца"  
 title="Только буквы, пробелы и дефисы. Например: Иван Иванов"  
 pattern="[A-Za-zА-ЯЁа-яё\s-]{1,50}"  
 required>  
 <input  
 type="text"  
 id="editOwnerAddress"  
 placeholder="Адрес владельца"  
 title="Адрес может содержать буквы, цифры, пробелы, запятые и точки"  
 pattern="[A-Za-zА-ЯЁа-яё0-9\s.,-]+"  
 required>  
 <input  
 type="number"  
 id="editYearOfManufacture"  
 placeholder="Год выпуска"  
 title="Введите год в диапазоне от 1900 до текущего"  
 min="1900"  
 max="2024"  
 required>  
 <div class="edit-buttons">  
 <button id="submitEditReg">Сохранить</button>  
 <button id="cancelEditReg">Отмена</button>  
 </div>  
 </div>  
 <ul id="regList"></ul>  
 </main>  
  
 <script src="/static/js/header.js"></script>  
 <script src="/static/js/registration.js" defer></script>  
</body>  
</html>

### **/js**

#### **cars.js**

const carsList = document.getElementById("carsList");  
const addCarButton = document.getElementById("addCarButton");  
const addCarForm = document.getElementById("addCarForm");  
const submitAddCar = document.getElementById("submitAddCar");  
const searchCarInput = document.getElementById("searchCarInput");  
const searchCarButton = document.getElementById("searchCarButton");  
  
const API\_BASE = "/carsdb";  
const API\_REGS = "/regdb";  
const makeRegex = /^[A-Za-zА-Яа-яЁё\s-]{1,50}$/;  
const modelRegex = /^[A-Za-zА-Яа-яЁё0-9\s-]{1,50}$/;  
const licensePlateRegex = /^[A-Za-zА-ЯЁ]{1}\d{3}[A-Za-zА-ЯЁ]{2}\d{2,3}$/;  
  
  
const AUTH\_TOKEN = localStorage.getItem("token");  
  
if (!AUTH\_TOKEN) {  
 alert("Вы не авторизованы. Перейдите на страницу входа.");  
 window.location.href = "/static/html/login.html";  
}  
  
async function authorizedFetch(url, options = {}) {  
 options.headers = {  
 ...options.headers,  
 "Authorization": `Bearer ${AUTH\_TOKEN}`,  
 "Content-Type": "application/json"  
 };  
  
 const response = await fetch(url, options);  
  
 if (response.status === 401) {  
 alert("Сессия истекла. Пожалуйста, войдите снова.");  
 window.location.href = "/static/html/login.html";  
 return null;  
 }  
  
 if (!response.ok) {  
 const errorData = await response.json();  
 console.error("Ошибка запроса:", errorData.detail);  
 alert(`Ошибка: ${errorData.detail}`);  
 return null;  
 }  
  
 return response;  
}  
  
async function loadCars() {  
 const response = await authorizedFetch(`${API\_BASE}/get\_cars/`);  
  
 if (!response || !response.ok) {  
 console.error("Ошибка загрузки автомобилей:", response?.statusText);  
 return;  
 }  
  
 const data = await response.json();  
  
 if (data && Array.isArray(data.cars)) {  
 const cars = data.cars;  
  
 if (cars.length === 0) {  
 carsList.innerHTML = "<li>Нет доступных автомобилей</li>";  
 } else {  
 carsList.innerHTML = "";  
 cars.forEach(car => {  
 const li = document.createElement("li");  
 li.setAttribute("data-make", car.make);  
 li.setAttribute("data-model", car.model);  
 li.setAttribute("data-license", car.license\_plate);  
 li.innerHTML = `  
 <div class="car-info">  
 <strong>${car.make} ${car.model}</strong>   
 </div>  
 <div class="button-row">  
 <button class="expand">Данные</button>  
 <button class="edit-car">Редактировать</button> <!-- Новая кнопка -->  
 <button class="find-reg">Найти регистрацию</button>  
 <button class="delete">Удалить</button>  
 </div>  
 <div class="car-details hidden">  
 <p>Марка: <strong>${car.make}</strong></p>  
 <p>Модель: <strong>${car.model}</strong></p>  
 <p>Номер: <strong>${car.license\_plate}</strong></p>  
 </div>  
 <div class="reg-result hidden">  
 <p class="reg-info"></p>  
 <button class="add-reg hidden">Добавить регистрацию</button>  
 <button class="hide-reg hidden">Скрыть</button>  
 </div>  
 `;  
 carsList.appendChild(li);  
  
  
 li.querySelector(".expand").addEventListener("click", () => {  
 const details = li.querySelector(".car-details");  
 details.classList.toggle("hidden");  
 });  
  
 li.querySelector(".edit-car").addEventListener("click", () => showEditCarForm(car));  
  
 li.querySelector(".find-reg").addEventListener("click", async () => {  
 const licensePlate = car.license\_plate;  
 const regResultDiv = li.querySelector(".reg-result");  
 const regInfoParagraph = regResultDiv.querySelector(".reg-info");  
 const addRegButton = regResultDiv.querySelector(".add-reg");  
 const hideRegButton = regResultDiv.querySelector(".hide-reg");  
  
  
 regInfoParagraph.textContent = "Загрузка...";  
 addRegButton.classList.add("hidden");  
 hideRegButton.classList.remove("hidden");  
  
 const regResponse = await authorizedFetch(`${API\_REGS}/search\_registrations/?query=${licensePlate}`);  
  
 regInfoParagraph.textContent = "";  
  
 if (regResponse) {  
 const data = await regResponse.json();  
 if (data && Array.isArray(data.registrations) && data.registrations.length > 0) {  
 const registration = data.registrations[0];  
 regInfoParagraph.innerHTML = `  
 <strong>Регистрация найдена:</strong><br>  
 Номер: ${registration.license\_plate}<br>  
 Владелец: ${registration.owner\_name}<br>  
 Адрес владельца: ${registration.owner\_address}<br>  
 Год выпуска: ${registration.year\_of\_manufacture}  
 `;  
 } else {  
 regInfoParagraph.textContent = "Регистрация не найдена.";  
 addRegButton.classList.remove("hidden");  
 }  
 } else {  
 regInfoParagraph.textContent = "Ошибка при поиске регистрации.";  
 }  
  
 regResultDiv.classList.remove("hidden");  
  
  
 addRegButton.addEventListener("click", () => {  
 const licensePlate = car.license\_plate;  
 window.location.href = `/static/html/registration.html?license\_plate=${licensePlate}`;  
 });  
  
 hideRegButton.addEventListener("click", () => {  
 regResultDiv.classList.add("hidden");  
 });  
 });  
  
 li.querySelector(".delete").addEventListener("click", async () => {  
 const licensePlate = car.license\_plate;  
  
 if (confirm("Удалить автомобиль? Обратите внимание: регистрация автомобиля также будет удалена!")) {  
 *// Проверка наличия регистрации* const regCheckResponse = await authorizedFetch(`${API\_REGS}/search\_registrations/?query=${licensePlate}`);  
 if (regCheckResponse && regCheckResponse.ok) {  
 const regData = await regCheckResponse.json();  
 const registrations = regData.registrations;  
  
 if (Array.isArray(registrations) && registrations.length > 0) {  
 *// Если регистрация существует, удаляем ее* const deleteRegResponse = await authorizedFetch(`${API\_REGS}/delete\_registration/${licensePlate}`, {  
 method: "DELETE",  
 });  
  
 if (!deleteRegResponse || !deleteRegResponse.ok) {  
 alert("Ошибка при удалении регистрации.");  
 return;  
 }  
 }  
 }  
  
 *// Удаление автомобиля* const deleteCarResponse = await authorizedFetch(`${API\_BASE}/delete\_car/${licensePlate}`, {  
 method: "DELETE",  
 });  
  
 if (deleteCarResponse && deleteCarResponse.ok) {  
 alert("Автомобиль успешно удален.");  
 loadCars(); *// Обновляем список автомобилей* } else {  
 alert("Ошибка при удалении автомобиля.");  
 }  
 }  
 });  
 });  
 }  
 } else {  
 console.error("Ответ сервера не содержит корректный список машин:", data);  
 alert("Ошибка загрузки автомобилей.");  
 }  
}  
  
function setFieldValidationStyle(input, isValid) {  
 if (isValid) {  
 input.style.border = "2px solid green";  
 } else {  
 input.style.border = "2px solid red";  
 }  
}  
  
document.getElementById("newCarMake").addEventListener("input", (event) => {  
 const isValid = makeRegex.test(event.target.value.trim());  
 setFieldValidationStyle(event.target, isValid);  
});  
  
document.getElementById("newCarModel").addEventListener("input", (event) => {  
 const isValid = modelRegex.test(event.target.value.trim());  
 setFieldValidationStyle(event.target, isValid);  
});  
  
document.getElementById("newCarLicensePlate").addEventListener("input", (event) => {  
 const isValid = licensePlateRegex.test(event.target.value.trim());  
 setFieldValidationStyle(event.target, isValid);  
});  
  
addCarButton.addEventListener("click", () => {  
 addCarForm.classList.toggle("hidden");  
});  
  
  
submitAddCar.addEventListener("click", async () => {  
 const makeInput = document.getElementById("newCarMake");  
 const modelInput = document.getElementById("newCarModel");  
 const licensePlateInput = document.getElementById("newCarLicensePlate");  
  
 const make = makeInput.value.trim();  
 const model = modelInput.value.trim();  
 const licensePlate = licensePlateInput.value.trim();  
  
 const isMakeValid = makeRegex.test(make);  
 const isModelValid = modelRegex.test(model);  
 const isLicensePlateValid = licensePlateRegex.test(licensePlate);  
  
 setFieldValidationStyle(makeInput, isMakeValid);  
 setFieldValidationStyle(modelInput, isModelValid);  
 setFieldValidationStyle(licensePlateInput, isLicensePlateValid);  
  
 if (!isMakeValid || !isModelValid || !isLicensePlateValid) {  
 alert("Пожалуйста, исправьте ошибки в форме перед добавлением автомобиля.");  
 return;  
 }  
  
 const response = await authorizedFetch(`${API\_BASE}/add\_car/`, {  
 method: "POST",  
 body: JSON.stringify({ make, model, license\_plate: licensePlate }),  
 });  
  
 if (response) {  
 addCarForm.classList.add("hidden");  
 loadCars();  
 } else {  
 alert("Ошибка добавления автомобиля.");  
 }  
});  
  
  
searchCarButton.addEventListener("click", () => {  
 const query = searchCarInput.value.toLowerCase();  
 const cars = document.querySelectorAll("li");  
 let found = false;  
  
 const noResultsMessage = document.querySelector(".no-results");  
 if (noResultsMessage) {  
 noResultsMessage.remove();  
 }  
 cars.forEach(car => {  
 const make = car.getAttribute("data-make") || "";  
 const model = car.getAttribute("data-model") || "";  
 const licensePlate = car.getAttribute("data-license") || "";  
  
 if (make.toLowerCase().includes(query) || model.toLowerCase().includes(query) || licensePlate.toLowerCase().includes(query)) {  
 car.style.display = "";  
 found = true;  
 } else {  
 car.style.display = "none";  
 }  
 });  
  
 if (!found && query != "") {  
 if (!document.querySelector(".no-results")) {  
 const noResultsMessage = document.createElement("li");  
 noResultsMessage.classList.add("no-results");  
 noResultsMessage.textContent = "Автомобили не найдены.";  
 carsList.appendChild(noResultsMessage);  
 }  
 }  
});  
  
function showEditCarForm(car) {  
 const editCarForm = document.getElementById("editCarForm");  
 const editCarMake = document.getElementById("editCarMake");  
 const editCarModel = document.getElementById("editCarModel");  
 const editCarLicensePlate = document.getElementById("editCarLicensePlate");  
 const submitEditCar = document.getElementById("submitEditCar");  
  
 editCarMake.value = car.make;  
 editCarModel.value = car.model;  
 editCarLicensePlate.value = car.license\_plate;  
  
 editCarForm.classList.remove("hidden");  
  
 editCarMake.addEventListener("input", (event) => {  
 const isValid = makeRegex.test(event.target.value.trim());  
 setFieldValidationStyle(event.target, isValid);  
 });  
  
 editCarModel.addEventListener("input", (event) => {  
 const isValid = modelRegex.test(event.target.value.trim());  
 setFieldValidationStyle(event.target, isValid);  
 });  
  
 editCarLicensePlate.addEventListener("input", (event) => {  
 const isValid = licensePlateRegex.test(event.target.value.trim());  
 setFieldValidationStyle(event.target, isValid);  
 });  
  
 *// Отмена* document.getElementById("cancelEditCar").addEventListener("click", () => {  
 editCarForm.classList.add("hidden");  
 });  
}  
  
document.getElementById("submitEditCar").addEventListener("click", async () => {  
 const editCarForm = document.getElementById("editCarForm");  
 const editCarMake = document.getElementById("editCarMake");  
 const editCarModel = document.getElementById("editCarModel");  
 const editCarLicensePlate = document.getElementById("editCarLicensePlate");  
  
 const make = editCarMake.value.trim();  
 const model = editCarModel.value.trim();  
 const licensePlate = editCarLicensePlate.value.trim();  
  
 const isMakeValid = makeRegex.test(make);  
 const isModelValid = modelRegex.test(model);  
 const isLicensePlateValid = licensePlateRegex.test(licensePlate);  
  
 setFieldValidationStyle(editCarMake, isMakeValid);  
 setFieldValidationStyle(editCarModel, isModelValid);  
 setFieldValidationStyle(editCarLicensePlate, isLicensePlateValid);  
  
 if (!isMakeValid || !isModelValid || !isLicensePlateValid) {  
 alert("Пожалуйста, исправьте ошибки в форме перед сохранением изменений автомобиля.");  
 return;  
 }  
  
 const updatedCar = {  
 make: make,  
 model: model,  
 };  
  
 const response = await authorizedFetch(`${API\_BASE}/update\_car/${licensePlate}`, {  
 method: "PUT",  
 body: JSON.stringify(updatedCar),  
 });  
  
 if (response) {  
 alert("Данные автомобиля успешно обновлены.");  
 editCarForm.classList.add("hidden");  
 loadCars();  
 } else {  
 alert("Ошибка при обновлении данных автомобиля.");  
 }  
});  
  
loadCars();

#### **header.js**

document.addEventListener("DOMContentLoaded", async () => {  
 const headerContainer = document.createElement("div");  
 headerContainer.id = "header-container";  
 document.body.prepend(headerContainer);  
  
 try {  
 const response = await *fetch*("/static/html/header.html");  
 if (response.ok) {  
 const headerHTML = await response.text();  
 headerContainer.innerHTML = headerHTML;  
  
 const loginButton = document.getElementById("login-button");  
 const registerButton = document.getElementById("register-button");  
 const logoutButton = document.getElementById("logout-button");  
  
 const isLoggedIn = localStorage.getItem("token");  
  
 if (isLoggedIn) {  
 loginButton.style.display = "none";  
 registerButton.style.display = "none";  
 logoutButton.style.display = "block";  
  
 logoutButton.addEventListener("click", () => {  
 localStorage.removeItem("token");  
 window.location.href = "/static/html/index.html";  
 });  
 } else {  
 loginButton.style.display = "inline";  
 registerButton.style.display = "inline";  
 logoutButton.style.display = "none";  
 }  
 } else {  
 console.error("Failed to load header.");  
 }  
 } catch (error) {  
 console.error("Error loading header:", error);  
 }  
});

#### **login.js**

document.getElementById("login-form").addEventListener("submit", async (e) => {  
 e.preventDefault();  
 const email = document.getElementById("email").value;  
 const password = document.getElementById("password").value;  
  
 const formData = new *URLSearchParams*();  
 formData.append("username", email);  
 formData.append("password", password);  
 formData.append("grant\_type", "password");  
  
 try {  
 const response = await *fetch*("http://127.0.0.1:8000/auth/login", {  
 method: "POST",  
 headers: { "Content-Type": "application/x-www-form-urlencoded" },  
 body: formData,  
 });  
  
 if (response.ok) {  
 const data = await response.json();  
 localStorage.setItem("token", data.access\_token);  
 alert("Login successful!");  
 window.location.href = "/static/html/index.html";  
 } else {  
 const data = await response.json();  
 alert(data.detail || "Login failed.");  
 }  
 } catch (error) {  
 alert("An error occurred. Please try again.");  
 }  
});

#### **register.js**

document.getElementById("registration-form").addEventListener("submit", async (e) => {  
 e.preventDefault();  
  
 const firstName = document.getElementById("first\_name").value;  
 const lastName = document.getElementById("last\_name").value;  
 const email = document.getElementById("email").value;  
 const password = document.getElementById("password").value;  
 const confirmPassword = document.getElementById("confirm\_password").value;  
  
 if (password !== confirmPassword) {  
 *alert*("Passwords do not match.");  
 return;  
 }  
  
 try {  
 const response = await *fetch*("http://127.0.0.1:8000/auth/register", {  
 method: "POST",  
 headers: { "Content-Type": "application/json" },  
 body: JSON.stringify({  
 first\_name: firstName,  
 last\_name: lastName,  
 email: email,  
 password: password,  
 confirm\_password: confirmPassword,  
 }),  
 });  
  
 if (response.ok) {  
 *alert*("Registration successful! Please login.");  
 window.location.href = "/static/html/login.html";  
 } else {  
 const data = await response.json();  
 *alert*(data.detail || "Registration failed.");  
 }  
 } catch (error) {  
 *alert*("An error occurred. Please try again.");  
 }  
});

#### **registration.js**

const regList = document.getElementById("regList");  
const addRegButton = document.getElementById("addRegButton");  
const addRegForm = document.getElementById("addRegForm");  
const submitAddReg = document.getElementById("submitAddReg");  
const searchRegInput = document.getElementById("searchRegInput");  
const searchRegButton = document.getElementById("searchRegButton");  
  
const API\_BASE = "/regdb";  
  
  
const AUTH\_TOKEN = localStorage.getItem("token");  
  
  
if (!AUTH\_TOKEN) {  
 alert("Вы не авторизованы. Перейдите на страницу входа.");  
 window.location.href = "/static/html/login.html";  
}  
const licensePlateRegex = /^[A-Z]{1}\d{3}[A-Z]{2}\d{2,3}$/;  
const ownerNameRegex = /^[a-zA-Zа-яА-ЯёЁ\s\-]{1,50}$/;  
const ownerAddressRegex = /^[А-Яа-яЁёA-Za-z0-9\s.,-\\/]+$/;  
const yearRegex = /^(19|20)\d{2}$/;  
  
const urlParams = new URLSearchParams(window.location.search);  
const licensePlateFromURL = urlParams.get("license\_plate");  
  
if (licensePlateFromURL) {  
 const licensePlateInput = document.getElementById("newRegLicensePlate");  
  
 if (licensePlateInput) {  
 licensePlateInput.value = licensePlateFromURL;  
 licensePlateInput.setAttribute("readonly", "readonly");  
  
 const addRegForm = document.getElementById("addRegForm");  
 if (addRegForm) {  
 addRegForm.classList.remove("hidden");  
 }  
 } else {  
 console.error("Field with ID 'newRegLicensePlate' not found in the HTML!");  
 }  
}  
  
async function authorizedFetch(url, options = {}) {  
 options.headers = {  
 ...options.headers,  
 "Authorization": `Bearer ${AUTH\_TOKEN}`,  
 "Content-Type": "application/json"  
 };  
  
 const response = await fetch(url, options);  
  
  
 if (response.status === 401) {  
 alert("Сессия истекла. Пожалуйста, войдите снова.");  
 window.location.href = "/static/html/login.html";  
 return null;  
 }  
  
  
 if (!response.ok) {  
 const errorData = await response.json();  
 console.error("Ошибка запроса:", errorData.detail);  
 alert(`Ошибка: ${errorData.detail}`);  
 return null;  
 }  
  
 return response;  
}  
  
  
async function loadRegs() {  
 const response = await authorizedFetch(`${API\_BASE}/get\_registrations/`);  
  
 if (!response || !response.ok) {  
 console.error("Ошибка загрузки регистраций:", response?.statusText);  
 return;  
 }  
  
 const data = await response.json();  
  
 if (data && Array.isArray(data.registrations)) {  
 const regs = data.registrations;  
  
 if (regs.length === 0) {  
 regList.innerHTML = "<li>Нет доступных регистраций</li>";  
 } else {  
 regList.innerHTML = "";  
 regs.forEach(reg => {  
 const li = document.createElement("li");  
 li.setAttribute("data-license\_plate", reg.license\_plate);  
 li.setAttribute("data-owner\_name", reg.owner\_name);  
 li.setAttribute("data-owner\_address", reg.owner\_address);  
 li.setAttribute("data-year", reg.year\_of\_manufacture);  
 li.innerHTML = `  
 <strong>${reg.license\_plate}</strong>  
 <div class="button-row">  
 <button class="expand">Данные</button>  
 <button class="edit-reg">Редактировать</button>  
 <button class="delete">Удалить</button>  
 </div>  
 <div class="reg-details hidden">  
 <p>Номер: <strong>${reg.license\_plate}</strong></p>  
 <p>Владелец: <strong>${reg.owner\_name}</strong></p>  
 <p>Адрес владельца: <strong>${reg.owner\_address}</strong></p>  
 <p>Год выпуска: <strong>${reg.year\_of\_manufacture}</strong></p>  
 </div>  
 <div class="car-result hidden">  
 <p class="car-info"></p>  
 <button class="hide-car-result hidden">Скрыть</button>  
 </div>  
 `;  
 regList.appendChild(li);  
  
 li.querySelector(".edit-reg").addEventListener("click", () => showEditRegForm(reg));  
  
  
 li.querySelector(".expand").addEventListener("click", async () => {  
 const details = li.querySelector(".reg-details");  
 details.classList.toggle("hidden");  
  
 if (!details.classList.contains("hidden")) {  
 const licensePlate = reg.license\_plate;  
  
 const carResponse = await authorizedFetch(`/carsdb/search\_cars/?query=${licensePlate}`);  
  
 if (carResponse) {  
 const data = await carResponse.json();  
 const car = data.cars.find(c => c.license\_plate === licensePlate);  
  
 if (car) {  
 details.innerHTML += `  
 <p><strong>Данные автомобиля:</strong></p>  
 <p>Марка: <strong>${car.make}</strong></p>  
 <p>Модель: <strong>${car.model}</strong></p>  
 <p>Номер: <strong>${car.license\_plate}</strong></p>  
 `;  
 } else {  
 details.innerHTML += "<p>Информация об автомобиле не найдена.</p>";  
 }  
 } else {  
 details.innerHTML += "<p>Ошибка при попытке загрузить данные об автомобиле.</p>";  
 }  
 }  
 });  
  
  
 li.querySelector(".delete").addEventListener("click", async () => {  
 if (confirm("Удалить регистрацию?")) {  
 const deleteResponse = await authorizedFetch(`${API\_BASE}/delete\_registration/${reg.license\_plate}`, {  
 method: "DELETE"  
 });  
 if (deleteResponse && deleteResponse.ok) {  
 loadRegs();  
 }  
 }  
 });  
 });  
 }  
 } else {  
 console.error("Ответ сервера не содержит корректный список регистраций:", data);  
 *alert*("Ошибка загрузки регистраций.");  
 }  
}  
  
submitAddReg.addEventListener("click", async () => {  
 const licensePlateInput = document.getElementById("newRegLicensePlate");  
 const ownerNameInput = document.getElementById("newOwnerName");  
 const ownerAddressInput = document.getElementById("newOwnerAddress");  
 const yearInput = document.getElementById("newYearOfManufacture");  
  
 const licensePlate = licensePlateInput.value.trim();  
 const ownerName = ownerNameInput.value.trim();  
 const ownerAddress = ownerAddressInput.value.trim();  
 const year = yearInput.value.trim();  
  
 const isLicensePlateValid = licensePlateRegex.test(licensePlate);  
 const isOwnerNameValid = ownerNameRegex.test(ownerName);  
 const isOwnerAddressValid = ownerAddressRegex.test(ownerAddress);  
 const isYearValid = yearRegex.test(year) && year >= 1900 && year <= new Date().getFullYear();  
  
 setFieldValidationStyle(licensePlateInput, isLicensePlateValid);  
 setFieldValidationStyle(ownerNameInput, isOwnerNameValid);  
 setFieldValidationStyle(ownerAddressInput, isOwnerAddressValid);  
 setFieldValidationStyle(yearInput, isYearValid);  
  
 if (!isLicensePlateValid || !isOwnerNameValid || !isOwnerAddressValid || !isYearValid) {  
 alert("Пожалуйста, исправьте ошибки в форме перед добавлением регистрации.");  
 return;  
 }  
  
 const response = await authorizedFetch(`${API\_BASE}/add\_registration/`, {  
 method: "POST",  
 body: JSON.stringify({  
 license\_plate: licensePlate,  
 owner\_name: ownerName,  
 owner\_address: ownerAddress,  
 year\_of\_manufacture: parseInt(year, 10),  
 }),  
 });  
  
 if (response) {  
 addRegForm.classList.add("hidden");  
 loadRegs();  
 } else {  
 alert("Ошибка добавления регистрации.");  
 }  
 window.location.href = "/static/html/registration.html";  
});  
  
searchRegButton.addEventListener("click", async () => {  
 const query = searchRegInput.value.toLowerCase();  
 const regs = document.querySelectorAll("li");  
 let found = false;  
  
 const noResultsMessage = document.querySelector(".no-results");  
 if (noResultsMessage) {  
 noResultsMessage.remove();  
 }  
  
 regs.forEach(reg => {  
 reg.style.display = "none";  
 });  
  
 regs.forEach(reg => {  
 const licensePlate = reg.getAttribute("data-license\_plate") || "";  
 const ownerName = reg.getAttribute("data-owner\_name") || "";  
 const ownerAddress = reg.getAttribute("data-owner\_address") || "";  
 const year = reg.getAttribute("data-year") || "";  
  
 if (  
 licensePlate.toLowerCase().includes(query) ||  
 ownerName.toLowerCase().includes(query) ||  
 ownerAddress.toLowerCase().includes(query) ||  
 year.toLowerCase().includes(query)  
 ) {  
 reg.style.display = "";  
 found = true;  
 }  
 });  
  
 if (!found && query !== "") {  
 const carSearchResponse = await authorizedFetch(`/carsdb/search\_cars/?query=${query}`);  
  
 if (carSearchResponse) {  
 const data = await carSearchResponse.json();  
 const cars = data.cars;  
  
 cars.forEach(car => {  
 regs.forEach(reg => {  
 const licensePlate = reg.getAttribute("data-license\_plate") || "";  
  
 if (licensePlate.toLowerCase() === car.license\_plate.toLowerCase()) {  
 reg.style.display = "";  
 found = true;  
 }  
 });  
 });  
 }  
 }  
  
 if (!found) {  
 if (!document.querySelector(".no-results")) {  
 const noResultsMessage = document.createElement("li");  
 noResultsMessage.classList.add("no-results");  
 noResultsMessage.textContent = "Ничего не найдено.";  
 regList.appendChild(noResultsMessage);  
 }  
 }  
});  
  
function setFieldValidationStyle(input, isValid) {  
 if (isValid) {  
 input.style.border = "2px solid green";  
 } else {  
 input.style.border = "2px solid red";  
 }  
}  
  
function showEditRegForm(reg) {  
 const editRegForm = document.getElementById("editRegForm");  
 const editRegLicensePlate = document.getElementById("editRegLicensePlate");  
 const editOwnerName = document.getElementById("editOwnerName");  
 const editOwnerAddress = document.getElementById("editOwnerAddress");  
 const editYearOfManufacture = document.getElementById("editYearOfManufacture");  
 const submitEditReg = document.getElementById("submitEditReg");  
  
 editRegLicensePlate.value = reg.license\_plate;  
 editOwnerName.value = reg.owner\_name;  
 editOwnerAddress.value = reg.owner\_address;  
 editYearOfManufacture.value = reg.year\_of\_manufacture;  
  
 editRegForm.classList.remove("hidden");  
  
 editOwnerName.addEventListener("input", (e) => {  
 const isValid = ownerNameRegex.test(e.target.value.trim());  
 setFieldValidationStyle(e.target, isValid);  
 });  
  
 editOwnerAddress.addEventListener("input", (e) => {  
 const isValid = ownerAddressRegex.test(e.target.value.trim());  
 setFieldValidationStyle(e.target, isValid);  
 });  
  
 editYearOfManufacture.addEventListener("input", (e) => {  
 const isValid = yearRegex.test(e.target.value.trim()) && e.target.value >= 1900 && e.target.value <= new Date().getFullYear();  
 setFieldValidationStyle(e.target, isValid);  
 });  
  
 document.getElementById("cancelEditReg").addEventListener("click", () => {  
 editRegForm.classList.add("hidden");  
 });  
}  
  
document.getElementById("newRegLicensePlate").addEventListener("input", (e) => {  
 const isValid = licensePlateRegex.test(e.target.value.trim());  
 setFieldValidationStyle(e.target, isValid);  
});  
  
document.getElementById("newOwnerName").addEventListener("input", (e) => {  
 const isValid = ownerNameRegex.test(e.target.value.trim());  
 setFieldValidationStyle(e.target, isValid);  
});  
  
document.getElementById("newOwnerAddress").addEventListener("input", (e) => {  
 const isValid = ownerAddressRegex.test(e.target.value.trim());  
 setFieldValidationStyle(e.target, isValid);  
});  
  
document.getElementById("newYearOfManufacture").addEventListener("input", (e) => {  
 const isValid = yearRegex.test(e.target.value.trim()) && e.target.value >= 1900 && e.target.value <= new Date().getFullYear();  
 setFieldValidationStyle(e.target, isValid);  
});  
  
document.getElementById("submitEditReg").addEventListener("click", async () => {  
 const editRegLicensePlate = document.getElementById("editRegLicensePlate");  
 const editOwnerName = document.getElementById("editOwnerName");  
 const editOwnerAddress = document.getElementById("editOwnerAddress");  
 const editYearOfManufacture = document.getElementById("editYearOfManufacture");  
  
 const licensePlate = editRegLicensePlate.value.trim();  
 const ownerName = editOwnerName.value.trim();  
 const ownerAddress = editOwnerAddress.value.trim();  
 const year = editYearOfManufacture.value.trim();  
  
 const isLicensePlateValid = licensePlateRegex.test(licensePlate);  
 const isOwnerNameValid = ownerNameRegex.test(ownerName);  
 const isOwnerAddressValid = ownerAddressRegex.test(ownerAddress);  
 const isYearValid = yearRegex.test(year) && year >= 1900 && year <= new Date().getFullYear();  
  
 setFieldValidationStyle(editRegLicensePlate, isLicensePlateValid);  
 setFieldValidationStyle(editOwnerName, isOwnerNameValid);  
 setFieldValidationStyle(editOwnerAddress, isOwnerAddressValid);  
 setFieldValidationStyle(editYearOfManufacture, isYearValid);  
  
 if (!isLicensePlateValid || !isOwnerNameValid || !isOwnerAddressValid || !isYearValid) {  
 alert("Пожалуйста, исправьте ошибки в форме перед редактированием регистрации.");  
 return;  
 }  
  
 const updatedReg = {  
 owner\_name: ownerName,  
 owner\_address: ownerAddress,  
 year\_of\_manufacture: parseInt(year, 10),  
 };  
  
 const response = await authorizedFetch(`${API\_BASE}/update\_registration/${licensePlate}`, {  
 method: "PUT",  
 body: JSON.stringify(updatedReg),  
 });  
  
 if (response) {  
 document.getElementById("editRegForm").classList.add("hidden");  
 loadRegs();  
 } else {  
 alert("Ошибка редактирования регистрации.");  
 }  
});  
  
loadRegs();

### **docker-compose.yml**

services:  
 backend:  
 build:  
 context: ./backend  
 dockerfile: Dockerfile  
 volumes:  
 - ./frontend:/app/frontend  
 ports:  
 - "8000:8000"  
 depends\_on:  
 - mongo  
  
 mongo:  
 image: mongo:latest  
 ports:  
 - "27018:27017"  
 volumes:  
 - mongo-data:/data/db  
  
volumes:  
 mongo-data: