**📌 Подробный план реализации новых задач**

📌 **Разделение работы на этапы** поможет **структурировать разработку** и **контролировать прогресс**.  
📌 **После каждого шага делаем коммит в GitHub**.  
📌 **Для работы с новыми методами API создаём отдельную ветку feature/api\_updates**, а после завершения работы **сливаем в master**.

**🚀 Этап 1: Добавление новых методов API**

🔹 **Цель** – реализовать **GET /submitData/<id>/**, **PATCH /submitData/<id>/**, **GET /submitData/?user\_\_email=<email>/**.  
🔹 **Ветка для работы** – feature/api\_updates.

**📌 Шаг 1.1 – Добавление GET /submitData/<id>/**

✅ **Цель:** Получить данные о перевале по id.  
✅ **Действия:**  
🔲 Добавить новый метод в **views.py** (RetrieveAPIView для одной записи).  
🔲 Добавить новый маршрут в **urls.py**.  
🔲 Протестировать в **Postman**.  
🔲 **Закоммитить изменения в GitHub (feature/api\_updates).**

**📌 Шаг 1.2 – Добавление PATCH /submitData/<id>/**

✅ **Цель:** Разрешить редактирование перевалов со статусом new.  
✅ **Действия:**  
🔲 Добавить новый метод в **views.py** (UpdateAPIView).  
🔲 Ограничить редактирование – запретить изменять ФИО, email, телефон.  
🔲 Проверять статус объекта перед редактированием (new – разрешено, остальные – запрещено).  
🔲 Добавить новый маршрут в **urls.py**.  
🔲 Протестировать в **Postman**.  
🔲 **Закоммитить изменения в GitHub (feature/api\_updates).**

**📌 Шаг 1.3 – Добавление GET /submitData/?user\_\_email=<email>/**

✅ **Цель:** Получить список всех перевалов, отправленных пользователем.  
✅ **Действия:**  
🔲 Добавить новый метод в **views.py** (ListAPIView).  
🔲 Фильтровать записи по email.  
🔲 Добавить новый маршрут в **urls.py**.  
🔲 Протестировать в **Postman**.  
🔲 **Закоммитить изменения в GitHub (feature/api\_updates).**

**📌 Шаг 1.4 – Покрытие тестами**

✅ **Цель:** Проверить новые методы API.  
✅ **Действия:**  
🔲 Добавить тесты в **tests.py** для новых методов API.  
🔲 Запустить тесты (pytest).  
🔲 **Закоммитить изменения в GitHub (feature/api\_updates).**

**📌 Шаг 1.5 – Слияние feature/api\_updates в master**

✅ **Цель:** Завершить этап работы с API.  
✅ **Действия:**  
🔲 Создать Pull Request и проверить код.  
🔲 **Слить ветку feature/api\_updates в master**.  
🔲 **Закоммитить финальные изменения в master**.

**🚀 Этап 2: Разработка мобильного приложения (PWA)**

🔹 **Цель** – разработать **PWA**, которое будет работать и в браузере, и как мобильное приложение.  
🔹 **Ветка для работы** – feature/pwa\_app.

**📌 Шаг 2.1 – Создание базовой структуры PWA**

✅ **Цель:** Развернуть базовое PWA-приложение на React + TypeScript.  
✅ **Действия:**  
🔲 Создать новый проект (npx create-react-app + --template typescript).  
🔲 Добавить PWA-манифест (manifest.json).  
🔲 Добавить Service Worker (serviceWorker.js).  
🔲 **Закоммитить в GitHub (feature/pwa\_app).**

**📌 Шаг 2.2 – Добавление страницы отправки данных**

✅ **Цель:** Реализовать форму отправки данных о перевале.  
✅ **Действия:**  
🔲 Добавить страницу /submit.  
🔲 Подключить API (POST /submitData/).  
🔲 **Закоммитить в GitHub (feature/pwa\_app).**

**📌 Шаг 2.3 – Добавление страницы списка перевалов**

✅ **Цель:** Отобразить все отправленные перевалы пользователя.  
✅ **Действия:**  
🔲 Добавить страницу /my-submits.  
🔲 Подключить API (GET /submitData/?user\_\_email=<email>/).  
🔲 **Закоммитить в GitHub (feature/pwa\_app).**

**📌 Шаг 2.4 – Добавление редактирования перевалов**

✅ **Цель:** Разрешить редактирование перевалов в статусе new.  
✅ **Действия:**  
🔲 Добавить страницу /edit/:id.  
🔲 Подключить API (PATCH /submitData/<id>/).  
🔲 **Закоммитить в GitHub (feature/pwa\_app).**

**📌 Шаг 2.5 – Карта перевалов**

✅ **Цель:** Отобразить перевалы на карте.  
✅ **Действия:**  
🔲 Подключить Google Maps API.  
🔲 Отобразить точки перевалов на карте.  
🔲 **Закоммитить в GitHub (feature/pwa\_app).**

**📌 Шаг 2.6 – Push-уведомления**

✅ **Цель:** Уведомлять пользователя о модерации.  
✅ **Действия:**  
🔲 Подключить Firebase Cloud Messaging.  
🔲 **Закоммитить в GitHub (feature/pwa\_app).**

**📌 Шаг 2.7 – Слияние feature/pwa\_app в master**

✅ **Цель:** Завершить этап работы над PWA.  
✅ **Действия:**  
🔲 Создать Pull Request.  
🔲 **Слить ветку feature/pwa\_app в master**.  
🔲 **Закоммитить финальные изменения в master**.

**🚀 Этап 3: Публикация API в облаке**

🔹 **Цель** – задеплоить API в облако **(Yandex Cloud или аналог)**.  
🔹 **Ветка для работы** – feature/deployment.

**📌 Шаг 3.1 – Настройка переменных окружения**

✅ **Цель:** Использовать .env для конфигурации БД.  
✅ **Действия:**  
🔲 Подключить переменные окружения (FSTR\_DB\_HOST, FSTR\_LOGIN, FSTR\_PASS).  
🔲 **Закоммитить в GitHub (feature/deployment).**

**📌 Шаг 3.2 – Развёртывание API в Yandex Cloud**

✅ **Цель:** Запустить API в облаке.  
✅ **Действия:**  
🔲 Настроить PostgreSQL в Yandex Cloud.  
🔲 Залить код на сервер.  
🔲 **Закоммитить в GitHub (feature/deployment).**

**📌 Шаг 3.3 – Документация и тестирование**

✅ **Цель:** Оформить документацию.  
✅ **Действия:**  
🔲 Добавить README.md.  
🔲 Настроить Swagger.  
🔲 Покрыть API тестами.  
🔲 **Закоммитить в GitHub (feature/deployment).**

📌 **Отчёт о выполнении проекта** 🚀

**🎯 Цель проекта**

Разработка и развёртывание REST API для системы учёта горных перевалов с возможностью добавления, редактирования и просмотра данных.

**✅ Реализованные возможности API**

🔹 **Добавление нового перевала** (POST /api/submitData/)

* Принимает JSON с данными о перевале.
* Записывает данные в базу.
* Возвращает id созданного объекта.

🔹 **Получение информации о перевале по ID** (GET /api/submitData/<id>/)

* Возвращает полные данные по указанному id.

🔹 **Редактирование данных о перевале** (PATCH /api/submitData/<id>/)

* Доступно только для перевалов в статусе new.
* Запрещено редактирование пользовательских данных (email, ФИО, телефон).

🔹 **Получение списка всех перевалов пользователя** (GET /api/submitData/?user\_\_email=<email>/)

* Позволяет получить все перевалы, добавленные указанным пользователем.

🔹 **Загрузка изображений** (POST /api/uploadImage/)

* Поддерживает загрузку изображений к перевалам.

**🔧 Технологический стек**

✅ **Django REST Framework** – для реализации API  
✅ **PostgreSQL** – в качестве базы данных  
✅ **Gunicorn** – для работы в продакшене  
✅ **Railway** – для развертывания API в облаке  
✅ **Swagger (drf-yasg)** – для автоматической генерации документации

**🚀 Развёртывание API в облаке**

🔹 API развернуто на **Railway**  
🔹 **Доступные эндпоинты:**

* **Swagger UI**: <https://mountain-pass-application-production.up.railway.app/swagger/>
* **Redoc**: <https://mountain-pass-application-production.up.railway.app/redoc/>

**🛠 Тестирование**

✅ **Unit-тесты** покрывают ключевые API-эндпоинты  
✅ Тесты успешно выполнены в pytest  
✅ API проверено через **Postman**

**📄 Документация**

✅ **README.md** включает:

* Описание проекта
* Инструкции по установке и запуску
* Примеры API-запросов

**🔚 Итог**

✅ API полностью разработано, протестировано и развернуто в облаке.  
✅ Все основные функции реализованы и работают стабильно.  
✅ Проект задокументирован и готов к использованию.

🎯 **Проект успешно завершён!** 🚀🎉

**Отчёт о текущем состоянии проекта Mountain Pass Application**

**1. Общая информация о проекте**

Mountain Pass Application — это REST API-сервис для системы учёта горных перевалов, позволяющий добавлять, редактировать и просматривать информацию о перевалах. Проект разработан с использованием Django и Django REST Framework (DRF), поддерживает хранение данных в базе PostgreSQL и развернут в облаке Railway.

**2. Архитектура проекта**

Проект построен на Django с DRF и включает в себя следующие компоненты:

* **REST API** для работы с перевалами.
* **База данных PostgreSQL** для хранения информации о пользователях, перевалах, изображениях и уровнях сложности.
* **Система аутентификации через email** (регистрация пользователя через email, без пароля).
* **Обработчик изображений** для загрузки и привязки фотографий к перевалам.
* **Документация API** с использованием Swagger и Redoc.
* **Тестирование** покрывает основные функциональные возможности API.

**3. Функциональность API**

**Основные эндпоинты:**

* **Добавление нового перевала** (POST /api/submitData/)
  + Принимает JSON с данными о перевале.
  + Создаёт запись в базе данных.
  + Возвращает id созданного объекта.
* **Получение информации о перевале по ID** (GET /api/submitData/<id>/)
  + Возвращает полные данные по указанному id.
* **Редактирование данных о перевале** (PATCH /api/submitData/<id>/)
  + Доступно только для перевалов в статусе new.
  + Запрещено редактирование пользовательских данных (email, ФИО, телефон).
* **Получение списка всех перевалов пользователя** (GET /api/submitData/?user\_\_email=<email>/)
  + Позволяет получить все перевалы, добавленные указанным пользователем.
* **Загрузка изображений** (POST /api/uploadImage/)
  + Поддерживает загрузку изображений к перевалам.

**4. База данных**

Используется **PostgreSQL**. Основные модели:

* User (пользователи, email как уникальный идентификатор).
* Coords (координаты перевалов).
* PerevalAdded (данные о перевалах, включая статус new, pending, accepted, rejected).
* PerevalImages (фотографии перевалов).
* PerevalDifficulty (уровни сложности по сезонам).
* WeatherInfo (данные о погоде в районе перевала).

**5. Тестирование**

Тесты написаны с использованием **pytest** и **Django TestCase**.

* **Тесты базы данных** (test\_db\_service.py):
  + Проверка создания пользователей, перевалов, изображений.
  + Проверка работы с GPS-треками (пока отключено).
* **Тесты API** (test\_api.py):
  + Тестирование основных эндпоинтов.
  + Проверка валидации данных.
  + Тесты загрузки изображений и получения перевалов по email.

**6. Развертывание**

* **Облачная платформа:** Railway.
* **Веб-доступ:**
  + Swagger: <https://mountain-pass-application-production.up.railway.app/swagger/>
  + Redoc: <https://mountain-pass-application-production.up.railway.app/redoc/>
* **WSGI и ASGI конфигурации** находятся в wsgi.py и asgi.py.

**7. Документация**

Файл README.md содержит:

* Описание проекта.
* Инструкции по установке и запуску.
* Примеры API-запросов.

**8. Критические моменты**

* **Редактирование перевалов ограничено статусом new**, но нет механизма отклонения изменений модераторами.
* **Обработка ошибок в API** требует доработки (некоторые исключения не обрабатываются корректно, возможны 500 Internal Server Error).
* **Поддержка GPX/KML-файлов** (GPS-треков) пока не реализована, но код частично подготовлен.

**9. Потенциальные улучшения**

* **Добавить возможность одобрения/отклонения заявок модераторами**.
* **Расширить обработку ошибок**, чтобы API возвращал более информативные сообщения.
* **Доработать поддержку GPX/KML-треков** для интеграции с картографическими сервисами.
* **Реализовать уведомления пользователям** при изменении статуса их заявок.

**Вывод:** Проект **успешно развернут**, API работает и тестируется, но есть несколько областей для улучшений, которые могут повысить его функциональность и устойчивость.