ChatGPT

Проект «Всё о питомце»: поэтапный технический план

1. Планирование и анализ требований

- **Сбор требований**. Провести встречу с заказчиком/стейкхолдерами, подробно обсудить цели проекта и сценарии использования: учёт питомцев, роли пользователей (владельцы, ветеринары, волонтёры, админы, гости) и их права.
- **Определение MVP и приоритетов**. Выделить критичные функции для первой версии (например, регистрация/авторизация, CRUD-профили питомцев и пользователей, календарь вакцинаций) и спланировать этапы реализации.
- **Технические требования**. Зафиксировать стек технологий (Node.js, PostgreSQL, React), требования по масштабируемости, безопасности, локализации (i18n на русском и английском), интеграции OAuth (Google, позже Telegram и соцсети).
- **Оценка рисков и ограничений**. Определить потенциальные проблемы: например, моделирование родительских связей у животных, обеспечение приватности медицинских данных, синхронизация передачи питомца между владельцами, зависимости между модулями (календарь, блог и т.п.).

2. Проектирование архитектуры

- **Определение архитектурного стиля**. Выбрать слоистую (monolith) или микросервисную архитектуру. Для первого этапа можно использовать монолит на Node.js с чётким выделением слоёв (MVC/Service/DAO), чтобы ускорить разработку и упростить сопровождение [1] [2].
- **Структура проекта**. Разбить код на доменные модули (авторизация, управление питомцами, блог, события, товары и т.д.). Придерживаться принципа единой ответственности (SRP) – каждый модуль обрабатывает строго свою область.
- **Схема взаимодействия**. Фронтенд (React) общается с сервером через REST API. Сервер на Node.js обрабатывает запросы, взаимодействует с базой данных и файловым хранилищем (для фото/документов). Все конфигурации и секреты вынести в `.env` (см. ниже) [3].
- **Управление ролями и правами**. Спроектировать ролевую модель: **гость** – чтение общедоступной информации; **владелец питомца** – полный доступ к данным своего питомца; **ветеринар** – доступ к истории здоровья и возможность вносить записи; **волонтёр** – редактирование информации о найденных/пропавших питомцах; **администратор** – управление всеми ресурсами. Использовать middleware для проверки ролей и прав доступа.
- **Безопасность**. Предусмотреть HTTPS для всего трафика, защиту от XSS/CSRF, CORS (ограничить домен фронтенда), rate limiting и логирование подозрительных запросов. Хранить пароли в виде хэшей (bcrypt), а сессии – в виде JWT с коротким сроком жизни (access token) и более длительным refresh token [4]. Секретный ключ JWT генерировать случайным, достаточно длинным, и хранить **только** в защищённой переменной окружения [5].

3. Проектирование базы данных

- **Выбор СУБД**. PostgreSQL обеспечивает реляционные связи (например, питомцы и их родители, владельцы и питомцы) и ACID-транзакции. Использовать ORM (Prisma или Sequelize) для удобства работы на уровне кода и миграции схемы. Prisma рекомендуется при использовании TypeScript благодаря автогенерации типов.
- **Моделирование сущностей**. Спроектировать таблицы (ER-диаграмму):
- `User` (id, имя, email, роль, хэш пароля, аватар и т.д.). Отдельные сущности или флаги для ролей (владелец, ветеринар и т.д.).
- `Pet` (id, имя, фото, дата_рождения, тип, пол, описание, место_рождения, адрес проживания, уровень публичности). Поля `отец_id`, `мать_id` с внешними ключами на `Pet` (само отношение родителя-ребенка).
- `Ownership` (многие ко многим): связь питомцев и пользователей (история владельцев, передача).
- `Vaccination` (вакцинации): тип вакцины, дата, привязка к питомцу, сделано ветврачом.
- `HealthRecord` (здоровье): текущие диагнозы, заметки врача, история приёмов (график веса, прививки).
- `Event` (календарь): тип события (приём, вакцинация, груминг, день рождения, лекарства), дата/время, описание, связанная сущность (питомец).
- `Post` (мини-блог): заголовок, текст, автор, статус (на модерации/опубликован), дата.
- `ProductSet` (набор товаров): категория, целевая аудитория (вид, возраст, уровень), список товаров (можно хранить просто набор тегов или описания).
- **Связи и индексы**. Добавить индексы по часто запрашиваемым полям (ID, внешние ключи, дата). Обеспечить целостность (ON DELETE RESTRICT/SET NULL для родительских полей, чтобы не потерять связь при удалении записей).
- **Миграции и seed**. Настроить систему миграций (Prisma Migrate или Sequelize migrations) и скрипты заполнения начальными данными (справочниками пород, типов событий и т.д.).

4. Разработка бэкенда (Node.js)

- **Настройка проекта**. Инициализировать Node.js (версия LTS, TypeScript), настроить `tsconfig.json`. Организовать структуру папок по доменам или слоям (например, `controllers/`, `services/`, `models/DAO/`, `routes/`, `middlewares/`, `config/`, `utils/`). По примеру чистой архитектуры отделить бизнес-логику (слой сервисов) от маршрутов и доступа к БД [1].
- **Конфигурация**. Использовать `.env` (dotenv) для секретов и настроек (база, OAuth ключи, токены). Не хранить чувствительные данные в репозитории [3] [5]. В `config/` вынести все настройки приложения (порты, лимиты, параметры кэширования, мульти-язычности).
- **Express / фреймворк**. Выбрать Express.js или более структурированный фреймворк (например, NestJS). Настроить CORS (разрешить домен фронтенда), Helmet (защита заголовков), логирование запросов (morgan/Winston). Реализовать централизованную обработку ошибок (error handling middleware).
- **Маршруты и контроллеры (API)**. Создать REST API по всем сущностям: `/auth` (регистрация, логин через email/пароль, OAuth Google), `/users` (управление профилями), `/pets`, `/vaccinations`, `/events`, `/posts`, `/products`, и т.д. Использовать версионирование (например, префикс `/api/v1/`). Документировать API с помощью Swagger (swagger-jsdoc + swagger-ui-express) и держать актуальную спецификацию.
- **Аутентификация и авторизация**. Реализовать JWT-аутентификацию: при логине генерировать access и refresh токены. Настроить middleware, который проверяет валидность JWT и подгружает пользователя. Для OAuth Google использовать passport.js

или аналог, связывать аккаунт Google с сущностью User. Позже добавить Telegram OAuth и другие соцсети.
- **Валидация и безопасность**. Для каждого API использовать валидацию входящих данных (Joi, express-validator или class-validator). Санитизация полей (например, на входе от пользователей). Ограничивать размеры загружаемых файлов. Реализовать RBAC – у каждого эндпоинта проверять роль пользователя и разрешать только соответствующие операции.
- **Хранение файлов**. Фото питомцев, документы и прочие файлы хранить не в базе, а в облачном хранилище (Amazon S3, Google Cloud Storage). API загружает файлы через Multer (или аналог) и сохраняет ссылки. Обеспечить права доступа: приватность записей отмечена в атрибутах, чтобы при запросе проверять, кто может посмотреть файл.
- **Отправка писем**. Для подтверждения email, уведомлений о событиях, сброса пароля – настроить почтовый сервис (например, SMTP через SendGrid или Mailgun).
- **Тестирование бэкенда**. Писать юнит-тесты для сервисов (Jest), интеграционные тесты для маршрутов (supertest). Покрытие должно быть достаточным для ключевых функций (аутентификация, CRUD операций).

5. Разработка фронтенда (React)

- **Настройка проекта**. Создать frontend-приложение (например, Create React App или Next.js для SSR, в зависимости от потребностей SEO и быстродействия). Использовать TypeScript для большей надёжности.
- **Структура фронтенда**. Организовать папки `components/`, `pages/` или `views/`, `services/api` (для запросов к бэкенду), `context/` или Redux хранилище (состояние пользователя, локализация, т.п.), `i18n/`. Настроить ESLint и Prettier для единого стиля кода.
- **Мобильная адаптация**. Делать дизайн адаптивным (Mobile-first). Использовать CSS Framework или UI-библиотеку (Material-UI, Ant Design, TailwindCSS) для быстрого создания интерфейсов. На этапе верстки проверять макеты как на десктопе, так и на мобильных.
- **Маршрутизация**. Настроить React Router для SPA (или систему маршрутов Next.js). Добавить приватные маршруты, доступные только авторизованным пользователям определённых ролей (например, в интерфейс ветеринара или админа).
- **Локализация (i18n)**. Подключить библиотеку (react-i18next или аналог) для перевода интерфейса и контента. Структурировать ресурсы переводов (`public/locales/ru/…, en/…`). При разработке компонентов использовать переводные ключи, чтобы поддерживать RU/EN версии.
- **UI компонентов**. Разработать интерфейсы:
- **Главная страница**: блоки блога, новости, ссылки, меню навигации.
- **Страница питомца**: профиль питомца, фото-галерея, история вакцинаций, здоровье, отзывы. Возможность загрузки/обновления фото через форму.
- **Профиль пользователя**: его питомцы, настройки аккаунта, переключение языка, привязки OAuth.
- **Admin Dashboard**: табличная и графическая аналитика по числу пользователей, активность питомцев (например, новые регистрации), тревожные события. Можно использовать библиотеки для графиков (Recharts, Chart.js).
- **Panel Veterinarian**: расписание приёмов (день/неделя), форма добавления заметок и диаграммы веса/здоровья питомцев. Использовать календарный компонент (например, react-big-calendar или fullcalendar-react) и графики.
- **Календарь событий**: общий календарь для владельца питомца, в котором видны все прививки, приёмы, дни рождения. Подсветка и напоминания.

- **Блог**: страница списка постов, форма создания поста (с базовым редактором rich-text), история своих постов. Новые посты отправляются на модерацию (помечаются статусом «на проверке»). Админ видит очередь модерации и может одобрять публикацию.
- **Каталог товаров**: страницы «наборов товаров» с фильтрацией (по возрасту, виду, уровню). Могут быть списком или карточками. Бренды не указываются, только тип товара и описание.
- **Интеграция с API**. Использовать Axios или fetch для взаимодействия с бэкендом. Настроить централизованный сервис API, где при каждом запросе автоматически добавляются токены (из хранилища состояния) и обрабатываются ошибки (401, 403 и др.).
- **Хранение токенов**. На клиенте сохранять access-token либо в `HttpOnly` cookie (безопасней, JS не видит) [6], либо в памяти/хранилище (localStorage с учётом рисков). Обязательно реализовать логику обновления токена по refresh-token.
- **Тестирование фронтенда**. Писать unit-тесты для компонентов (Jest + React Testing Library). Для критичных сценариев (авторизация, создание поста, календарь) сделать E2E-тесты (Cypress). Проверять корректность локализации (переключение языков).

6. Админ-панель и аналитика

- **Отдельный интерфейс**. Разработать раздел «Админ-панель» (можно как часть общего приложения с ограниченным доступом или отдельный роутинг). Отображать сводную статистику: число активных пользователей, количество питомцев по категориям, динамику новых записей (графики), ожидающие модерации посты.
- **Инструменты аналитики**. Использовать графические библиотеки (Charts) и таблицы. Реализовать фильтры (по дате, по ролям пользователей). При необходимости интегрировать сторонний сервис аналитики (Google Analytics для общего трафика, но данные проекта лучше держать «дома»).
- **Управление пользователями**. Возможность блокировки/удаления неактивных или подозрительных аккаунтов, настройки ролей.

7. Панель ветеринара

- **Расписание приёмов**. Раздел, где ветеринар видит свой календарь записей (интегрированный с общим календарём событий). Возможность добавления/редактирования приёма в конкретную дату/время.
- **Заметки и отчёты**. На странице каждого питомца врач может добавлять медицинские записи (диагнозы, назначения) и графики показателей (например, динамика веса, температуры). Все записи сохраняются в `HealthRecord`.
- **Права доступа**. Ветврач видит пациентов только через записи, где он указан (или всех питомцев конкретного владельца по поручению). Ограничить доступ к медицинским данным по GDPR/конфиденциальности (если применимо).

8. Календарь событий

- **Общие события**. Поддержать создание событий для питомца: вакцинация, приём у врача, стрижка (груминг), приём лекарств и день рождения. Эти события показываются в профиле питомца и в общем календаре пользователя.
- **Напоминания**. Реализовать систему нотификаций: отправлять email/SMS-уведомления перед важными событиями (настройку передать пользователю: напоминать за день/час). Можно использовать CRON-задачу или внешние сервисы (например, AWS SNS).

- **Редактирование**. Владельцы и админы могут создавать и редактировать события питомцев. Соответственно, корректно валидировать даты (невозможность вносить задним числом или перекрывать приёмы).
- **Интеграция с API**. Эндпоинты `/events` поддерживают операции CRUD. Клиентский календарь подписывается на обновления через веб-сокеты (опционально) или периодически запрашивает данные.

9. Мини-блог

- **Сущность «Пост»**. Пользователи могут писать короткие посты о питомцах или общую информацию. При создании пост попадает в статус «на проверке». Админ-панель показывает очередь модерации, где админ может одобрить или отклонить пост.
- **Rich Text**. На фронтенде использовать редактор (Quill, Draft.js) для удобства форматирования текста, но на бэкенде хранить чистый HTML или Markdown. Производить санитизацию (удалять скрипты).
- **Категории/теги**. Опционально, поддерживать теги (например, «здоровье», «питание»), чтобы на главной странице можно было фильтровать посты.
- **Публикация**. Опубликованные посты отображаются на главной странице и в списке блога, новые – только после одобрения.

10. Каталог товаров (наборы)

- **Модель данных**. Сущность `ProductSet` с полями: название набора, описание, категория (вид питомца, возраст, уровень активности). Не хранить бренды – информация обобщённая. Можно сделать таблицу категорий и связку many-to-many для товаров внутри набора (если потребуется разбиение на несколько вещей).
- **UI**. На сайте – страницы «Наборы для щенков», «Для собак активных», «Для пожилых кошек» и т.д. Реализовать фильтры по типам. Пока без оплаты – просто как справочник товаров.
- **Административное управление**. Админ-панель должна позволять добавлять/редактировать наборы товаров.

11. Главная страница и маркетинговый контент

- **Главная**. Красочное лендинг-окно с обзором функционала, последние новости и блог-посты. Ссылки на важные разделы (календарь, профиль, магазин).
- **Статические страницы**. «О нас», «Контакты», «Помощь». Поддержать возможность пополнять контент через админку.
- **Поисковая оптимизация**. Если это SPA, можно рассмотреть SSR (Next.js) или статическую сборку для улучшения SEO (например, индексация Google).

12. Хранение фото и документов

- **Файловое хранилище**. Использовать облако (S3 или аналог) для фото питомцев, родословных, сканов документов. На бэкенде хранить только URL и мета-данные (тип файла, к какому питомцу привязан).
- **Обслуживание**. Оптимизировать изображения (thumbnail, сжатие). Добавить возможность выбирать публичность: некоторые фото или записи могут быть видны всем, а некоторые – только владельцу и врачам.

- **Резервное копирование**. Настроить бэкап БД и бэкап файлов (например, версии в S3). Это часть финального этапа операционной поддержки.

13. Документирование API

- **Swagger/OpenAPI**. Внедрить генерацию документации Swagger (с помощью yaml или аннотаций). Запустить Swagger UI на отдельном руте (`/api-docs`). Каждый новый эндпоинт добавлять в документацию.
- **README и wiki**. Описать проект в README (технологии, сборка, запуск). Для команды – подробные инструкции (установка dev окружения, миграции, деплой).

14. Тестирование и качество кода

- **Unit-тесты**. Backend: Jest+Supertest для сервисов и контроллеров. Frontend: Jest + React Testing Library для компонентов, функций. Покрытие критичных модулей (авторизация, CRUD операций, валидации).
- **Интеграционные тесты**. Проверка работы связок (например, регистрацию нового пользователя с одновременным созданием профиля и PET, создание события в календаре).
- **E2E-тесты**. Инструмент наподобие Cypress для полного сценария: от регистрации до публикации поста. Тестирование пользовательского потока.
- **Линтинг и статический анализ**. ESLint и Prettier настроить единый стиль для фронта и бэка. Добавить проверки в CI: сборку проекта, линтер, прогон тестов. Для безопасности – можно подключить SonarQube или Snyk.
- **Регрессия i18n**. Проверить все страницы на корректность перевода, наличие ключей в обоих языках, отсутствие «битых» меток.

15. CI/CD и деплой

- **Система контроля версий**. GitFlow или GitHub Flow: develop/main ветви, feature-ветки, pull request. В комментариях указывать связанные задачи из трекера (GitHub Issues).
- **CI-пайплайн**. Настроить GitHub Actions (или аналог): на каждый PR — сборка, линт, тесты. Если всё ок, мерж в main – автодеплой на staging.
- **Docker и окружения**. Создать Dockerfile для бэкенда и (при желании) фронтенда, чтобы запускать сервисы в контейнерах. Для локальной разработки можно использовать `docker-compose` (с Node, Postgres, Redis). Переменные окружения подставлять из `.env.development/.env.production`.
- **Хостинг**. Выбрать провайдер: AWS (EC2/ECS + RDS), Google Cloud (GKE + Cloud SQL) или Heroku/Vercel для простоты. Фронтенд можно развернуть как статический (Netlify, Vercel) или на том же хосте с бэком.
- **CD-пайплайн**. После успешного мержда в main: создавать версию, запускать тесты ещё раз, автоматически деплоить на продакшен. Например, через GitHub Actions или Jenkins. Добавить уведомления команды при неудаче билда.
- **Секреты и конфигурация**. В продакшене хранить секреты в защищённом хранилище (AWS Secrets Manager, Vault) или переменных окружения платформы. Никогда не коммитить `.env.production` с настоящими ключами.

16. Мониторинг и поддержка

- **Логирование**. Внедрить централизованное логирование ошибок и запросов (Winston на Node.js). Логи можно отправлять в внешний сервис (Logstash, CloudWatch, ELK).

- **Мониторинг производительности**. Подключить мониторинг (Prometheus + Grafana или облачные аналоги) для отслеживания CPU, памяти, времени ответов API. Настроить алерты (CPU > 80%, ошибки 5xx).
- **Обновления**. Регулярно обновлять зависимости, особенно безопасность (dependabot). Ежемесячный аудит безопасности (npm audit, OWASP-тесты).
- **Документация пользователя**. При необходимости подготовить FAQ или раздел помощи (как загрузить фото, распланировать событие).

Используемые технологии и лучшие практики

- **Node.js & Express/NestJS** (TypeScript) на бэкенде – быстрый REST API.
- **PostgreSQL + ORM** (Prisma или Sequelize) – реляционная БД с миграциями.
- **JWT аутентификация** (access/refresh tokens) [4], OAuth 2.0 для Google/TG.
- **.env** (dotenv) для конфигурации [3] [5], все пароли/ключи в секретах.
- **ESLint, Prettier** – единый стиль кода; **CORS** – настроенный доступ между фронтом и API.
- **Clean/MVC архитектура** [1] [2]: разделение на контроллеры, сервисы, модели/DAO.
- **Middleware**: авторизация, валидация (Joi), логирование (morgan), защита (helmet, rate-limit).
- **i18n**: react-i18next на фронтенде, переводы в JSON-файлах. Сервер: возвращать сообщения на нужном языке (через заголовок Accept-Language или выбор профиля).
- **CI/CD**: GitHub Actions (или GitLab CI), автоматические тесты и деплой. Регулярные интеграционные сборки.
- **Тестирование**: TDD/BDD подход для ключевых функций, покрытие основных сценариев.
- **Сквозная безопасность**: HTTPS, HttpOnly cookies для токенов [6], периодическое обновление токенов, ревью кода на уязвимости.

Каждый этап разработки сопровождается постановкой конкретных задач в трекере: дизайн архитектуры, разработка функций, написание тестов, ревью, фикс багов, релиз. Такой поэтапный подход с постоянной интеграцией и контролем качества обеспечит создание надёжного, масштабируемого и удобного приложения «Всё о питомце», отвечающего всем заявленным требованиям.

Источники и best practices: основывано на руководствах по проектированию Node.js приложений и практике отрасли [1] [3] [5] [4] [7] [6].

---

[1] [2] [3] Node.js project architecture best practices - LogRocket Blog
https://blog.logrocket.com/node-js-project-architecture-best-practices/

[4] [5] [6] [7] 5 JWT authentication best practices for Node.js apps | Tech Tonic
https://medium.com/deno-the-complete-reference/5-jwt-authentication-best-practices-for-node-js-apps-f1aaceda3f81