**Техническое задание (ТЗ) на разработку платформы Pet Project**

**1. Введение**

Проект представляет собой онлайн-платформу для поиска, бронирования и управления услугами для домашних животных. Платформа объединяет три основные группы пользователей: владельцев питомцев, учреждения (клиники, груминг-салоны и т.д.) и внутренних администраторов.

**2. Цели и задачи**

* Быстрый выход на рынок (MVP) с базовым функционалом.
* Универсальная админ-панель для управления всеми ролями.
* Расширяемость и возможность добавления новых сервисов и регионов.

**3. Общая архитектура**

* **Монолитное** Django-приложение с REST API (Django REST Framework).
* **Frontend**: React для веб-клиента; React Native (или Flutter) для мобильного приложения.
* **База данных**: PostgreSQL.
* **Фоновые задачи**: Celery + Redis.
* **Хранение файлов**: S3-совместимое хранилище.
* **Контейнеризация**: Docker + docker-compose для разработки и тестирования. Обеспечивает изоляцию окружения, единообразие dev/prod, упрощает деплой и CI/CD.

**4. Функциональные требования**

**4.1. Регистрация и аутентификация**

* Регистрация по e-mail и паролю.
* Вход через социальные сети (опционально).
* Восстановление пароля по e-mail.

**4.2. Роли и права доступа**

* **Владельцы питомцев**: управление своими животными, бронирование услуг, отзывы, онлайн-оплата.
* **Администраторы учреждения**: CRUD-специалистов, расписание, прайс-листы, просмотр бронирований, отчёты.
* **Специалисты (ветврачи, грумеры)**: отмечают оказанные услуги, прикрепляют документы к карточке питомца.
* **Менеджеры по биллингу**: договоры, выставление счетов, контроль задолженностей.
* **Главный администратор (платформы)**: управление контрагентами, выдача прав администраторам учреждений и менеджерам.

**4.3. Клиентские приложения**

* **Веб-приложение** (React) для владельцев и внутренних админов.
* **Мобильное приложение** (React Native/Flutter) для владельцев и специалистов.

**4.4. Админ‑панель**

Для MVP можно использовать встроенную **Django Admin Site** без отдельного React‑интерфейса:

* **Единая панель** /admin/ для всех администраторов.
* Пользователи с флагом is\_staff=True видят админку, права настроены через группы (Group) и разрешения (Permission).
* **Настройка ModelAdmin**:
  + has\_module\_permission(self, request) — скрывает/показывает приложения в меню.
  + get\_queryset(self, request) — ограничивает записи, например, только по своему учреждению.
  + Переопределение has\_add\_permission, has\_change\_permission, has\_delete\_permission — для CRUD‑доступа.
* **Объектные права** (если нужно тонкое разграничение) реализуются через django‑guardian.
* **Несколько панелей** (опционально):
* from django.contrib.admin import AdminSite
* class ProviderAdminSite(AdminSite):
* site\_header = "Admin Provider"
* provider\_admin = ProviderAdminSite(name='provider\_admin')

# регистрируете только нужные модели

* **Пример групп и разрешений**:
  1. Группа Admin\_Provider: доступ к моделям Provider, Employee, Booking (с ограничением по объектам).
  2. Группа Admin\_Platform: доступ ко всем моделям, включая Contract, Invoice, User.

**4.5. Маркетплейс передержки (Pet Sitting)**

* **Модель PetSittingAd**:
  + pet: ForeignKey(Pet)
  + owner: ForeignKey(User)
  + start\_date, end\_date
  + description, status (активно/закрыто)
  + location, max\_distance\_km
  + compensation\_type (возмездно/безвозмездно) — выбор между платной и бесплатной передержкой.
* **Публикация объявления**: форма в веб/моб. приложении.
* **Подписка на объявления**:
  + Пользователь задаёт фильтр (типы питомцев, даты, радиус).
  + Система рассылает уведомления (push/email) новым объявлениям.
* **Обмен контактами и мессенджер**:
  + Встроенный чат между владельцем и потенциальным передержчиком.
  + Альтернативно, можно показывать только кнопку «Запрос контактов» и делиться номером телефона при согласии.
* **Отклики на объявление**:
  + Механизм отклика с уведомлением владельцу.
  + Подтверждение передержки и перевод объявления в статус «закрыто».

**4.6. Совместный доступ к карточке питомца**

* **Модель PetAccess**:
  + pet: ForeignKey(Pet)
  + granted\_to: ForeignKey(User)
  + granted\_by: ForeignKey(User)
  + token: UUID, expires\_at
  + permissions: JSONField (например: {"read": true, "book": false})
* **QR‑код**:
  + Ссылка вида /share/{token}/ превращается в QR‑код.
* **Уровни прав**:
  + read — просмотр карточки.
  + book — право создавать бронирования для питомца.
  + write — редактирование мед‑карты.
* **Управление доступом**:
  + Владелец может отзывать доступ и обновлять права.

**5. Нефункциональные требования. Нефункциональные требования Нефункциональные требования**

* Доступность: ≈99.5% (Heroku/PAAS-инфраструктура).
* Масштабируемость: начальные 1 000–5 000 активных пользователей.
* Бэкапы: ежедневный дамп БД и файлового хранилища, хранение 7 дней.
* Мониторинг и алерты: UptimeRobot + Telegram.

**6. Технологический стек**

* Backend: Django, Django REST Framework
* Database: PostgreSQL
* Frontend: React, React Native (или Flutter)
* Background jobs: Celery, Redis
* Storage: AWS S3 (или DigitalOcean Spaces)
* Контейнеризация: Docker + docker-compose
* CI/CD: GitHub Actions

**7. Компоненты системы и их взаимодействие**

1. **users** (аутентификация, RBAC)
2. **pets** (карточки питомцев, мед‑история)
3. **catalog** (услуги, категории)
4. **providers** (учреждения, специалисты, расписания)
5. **booking** (бронирование с учетом буферов)
6. **billing** (договоры, счета, комиссия)
7. **notifications** (push, email)
8. **geolocation** (интеграция с Google Maps)

**8. Модели данных (основные)**

* User (с ролями)
* Pet (атрибуты: вид, порода, возраст, мед‑карта)
* Service (название, категория, стоимость)
* Provider (учреждение, локация, услуги)
* Employee (специалист, профиль, расписание)
* Booking (клиент, услуга, время, статус)
* Contract (договор, условия, статус)
* Invoice (счет, сумма, дата выпуска, статус)

**9. API-эндпоинты (пример)**

* POST /api/auth/login/ — вход
* GET /api/pets/ — список питомцев
* POST /api/booking/ — создание бронирования
* GET /api/providers/?lat=&lng=&radius= — поиск поставщиков
* GET /api/admin/contracts/ — список договоров (для менеджера по биллингу)

**10. Администрирование и права доступа**

**Вопрос:** могут ли администраторы учреждения и платформенные администраторы работать в одной админ‑панели с разным функционалом?

**Ответ:** Да. Django + DRF + React позволяет использовать единую админ‑SPA с разграничением доступа:

* Настраиваем **RBAC** через django.contrib.auth.models.Group и Permission.
* Каждой роли (группе) выдаём набор разрешений на модели и объекты.
* При авторизации фронтенд запрашивает список разрешений и строит меню.
* Опционально: для более тонких настроек — django-guardian (object-level permissions).

Таким образом, в одном интерфейсе администратор учреждения видит только разделы по своему контрагенту (специалисты, расписание, отзывы), а главный админ платформы — всю систему.

**11. Развёртывание и инфраструктура**

* Единый VPS/PAAS (Heroku, Elastic Beanstalk, DigitalOcean App Platform).
* Docker Compose: сборка web, worker, redis, postgres.
* Авто‑деплой через GitHub Actions.
* Настройка health-check и автоматический рестарт.

**12. Мониторинг и логирование**

* UptimeRobot → Telegram
* Логи Django + Celery → Papertrail / Logentries
* Ошибки → Sentry

**13. Безопасность и соответствие**

* TLS 1.2+ для трафика
* Шифрование медданных в покое (AES‑256)
* Соответствие GDPR и локальным законам по медданным

**14. Словарь терминов**

* **MVP** — минимальный жизнеспособный продукт
* **RBAC** — Role-Based Access Control
* **SPA** — Single Page Application

**15. Пользовательские истории**

**15.1. Владелец питомца**

* Как владелец питомца, я хочу зарегистрироваться и авторизоваться, чтобы пользоваться сервисом.
* Как владелец питомца, я хочу добавить карточку своего питомца, чтобы отслеживать медицинскую историю.
* Как владелец питомца, я хочу найти ближайшие клиники на карте, чтобы записаться на услугу.
* Как владелец питомца, я хочу оплатить услугу онлайн, чтобы не тратить время на оплату в клинике.
* Как владелец питомца, я хочу получать напоминания о вакцинациях, чтобы не забывать о сроках.

**15.2. Администратор учреждения**

* Как администратор клиники, я хочу добавлять и редактировать специалистов, чтобы управлять командой.
* Как администратор клиники, я хочу настраивать расписание специалистов, чтобы обеспечить приём клиентов, в т.ч. автоматически:
  + Автоматизированное создание расписания на основе заданных правил (рабочие дни, часы, перерывы).
  + Приоритеты и ограничения (максимум приёмов в день, буферы между приёмами).
  + Ручная корректировка слотов через простой интерфейс (drag & drop).
  + Уведомления специалистам о любых изменениях в их расписании.
* Как администратор клиники, я хочу задавать список услуг и их стоимость, чтобы они отображались владельцам питомцев.

**15.3. Специалист**

* Как специалист, я хочу видеть своё расписание, чтобы знать, кого я принимаю.
* Как специалист, я хочу отмечать оказанные услуги и прикреплять документы к карточке питомца, чтобы вести медицинскую историю.

**15.4. Менеджер по биллингу**

* Как менеджер по биллингу, я хочу создавать и редактировать договоры, чтобы регламентировать условия сотрудничества.
* Как менеджер по биллингу, я хочу просматривать и выставлять счета, чтобы контролировать финансовые потоки.
* Как менеджер по биллингу, я хочу ограничивать доступ учреждений при просрочке оплаты, чтобы мотивировать своевременные платежи.

**15.5. Главный администратор платформы**

* Как главный администратор, я хочу добавлять и настраивать контрагентов, чтобы управлять сетью учреждений.
* Как главный администратор, я хочу выдавать права администраторам учреждений и менеджерам по биллингу.
* Как главный администратор, я хочу видеть общую статистику по бронированиям и доходам, чтобы принимать решения.

**15.6. Пользователь передержки**

* Как владелец питомца, я хочу создать объявление о передержке питомца, чтобы найти желающих присмотреть.
* Как потенциальный передержчик, я хочу указать свой профиль и настройки (типы питомцев, даты, радиус) и получать уведомления о новых объявлениях.
* Как пользователь, я хочу использовать встроенный чат для обсуждения деталей передержки.
* Как владелец питомца, я хочу указать в объявлении, будет ли оно возмездным или бесплатным, чтобы потенциальные передержчики понимали условия.
* Как владелец питомца, я хочу создать объявление о передержке питомца, чтобы найти желающих присмотреть.
* Как потенциальный передержчик, я хочу указать свой профиль и настройки (типы питомцев, даты, радиус) и получать уведомления о новых объявлениях.
* Как пользователь, я хочу использовать встроенный чат для обсуждения деталей передержки.

**15.7. Пользователь совместного доступа**

* Как владелец питомца, я хочу генерировать QR‑код для доступа к карточке питомца с заданным уровнем прав (чтение, бронирование, запись).
* Как приглашённый пользователь, я хочу получать доступ к карточке питомца в соответствии с выданными правами по ссылке или QR‑коду.

**16. Высокоуровневая архитектурная диаграмма**

flowchart LR

subgraph Frontend

A[React Web] -->|API calls| API

B[React Native Mobile] -->|API calls| API

end

subgraph Backend

API[API Gateway / Django DRF] --> U[user Service]

API --> P[pets Service]

API --> C[catalog Service]

API --> D[providers Service]

API --> Bk[booking Service]

API --> Bl[billing Service]

API --> N[notifications Service]

API --> G[geolocation Service]

API --> S[marketplace Service]

API --> A[access Service]

U & P & C & D & Bk & Bl & N & G & S & A --> DB[(PostgreSQL)]

N --> |Celery| Q[Redis]

end

**17. Детализированная дорожная карта (Roadmap)**

Ниже представлена подробная разбивка по двумнедельным спринтам на ~12 спринтов (около 6 месяцев) для разработки MVP и сопутствующих компонентов.

**Спринт 0 (1 неделя)**

* Настройка репозитория: • Инициализация Git, настройка ветвления (Gitflow) • Документирование процессов работы в README
* Docker и окружение разработчика: • Написание Dockerfile для Django и React • Конфигурация docker-compose (web, db, redis, worker) • Скрипты запуска и миграций
* CI/CD начального уровня: • Настройка GitHub Actions: линтинг, тесты, сборка контейнеров
* Деплой dev-среды на PAAS (Heroku/DigitalOcean App Platform)
* Мониторинг и логирование: • Подключение Sentry, UptimeRobot, базовые health checks

**Спринт 1 (2 недели)**

* **Auth & User**: • Регистрация / вход по email+паролю • Восстановление пароля (email) • Модели User + роли (Group, Permission)
* **Модуль Pets**: • CRUD карточки питомца (модель, API, админ)
* **Страницы React**: • Форма регистрации/логина • Интерфейс списка и создания питомцев
* Тесты: unit-тесты для User и Pet models

**Спринт 2 (2 недели)**

* **Catalog & Providers**: • Модель Service и Category • Модель Provider и связь с Service • API для списка услуг и клиник
* **Frontend**: • Страница каталога услуг (React) • Фильтры по категории, цене
* **Геолокация**: • Интеграция Google Maps JS API • Рассчет расстояния в запросах
* Тесты API и E2E-спецификации (Postman/Newman)

**Спринт 3 (2 недели)**

* **Booking (часть 1)**: • Модель Booking с полем time\_slot • Базовый CRUD: создание без буферов, просмотр • API-эндпоинты для создания и списка
* **Notifications**: • Настройка Celery+Redis • Отправка первого push/email (Welcome)
* **Frontend**: • Форма бронирования слота • Отображение статуса бронирования
* Тесты: интеграционные тесты Booking workflow

**Спринт 4 (2 недели)**

* **Booking (часть 2)**: • Добавление буферов между приёмами • Валидация конфликтов расписания • Оптимизированный поиск доступных слотов
* **Frontend**: • Обновление интерфейса выбора времени с учётом буферов
* **Админ-панель (Django Admin)**: • Настройка ModelAdmin для Booking с ограничениями по провайдеру

**Спринт 5 (2 недели)**

* **Billing & Contracts (часть 1)**: • Модели Contract, Invoice, Commission • API для менеджера по биллингу: CRUD контрактов и счетов • Excel-отчёты через Celery task
* **RBAC в Admin**: • Настройка групп Admin\_Provider и Admin\_Platform • Права CRUD для моделей billing в Admin
* **Frontend Admin**: • Страницы управления контрактами и счетами (React)

**Спринт 6 (2 недели)**

* **Pet Sitting Marketplace**: • Модель PetSittingAd с compensation\_type • API для создания и списка объявлений • Логика фильтров (дата, радиус)
* **Subscription Engine**: • Настройка подписки пользователей на объявления • Рассылка уведомлений о новых объявлениях
* **Frontend**: • Интерфейс создания объявления • Лента объявлений и фильтры

**Спринт 7 (2 недели)**

* **Chat Module**: • Выбор технологии (WebSocket/REST) • Модели Message, Conversation • API для чата между владельцем и передержчиком • Базовый UI чата в React
* Безопасность чата: проверка прав участников

**Спринт 8 (2 недели)**

* **Pet Access Sharing**: • Модель PetAccess с JSONField permissions • QR-код генерация и endpoint /share/{token} • Логика приёма и валидации токена • UI для управления доступом

**Спринт 9 (2 недели)**

* **Mobile App Setup (часть 1)**: • Инициализация React Native / Flutter проекта • Аутентификация и основной навигатор • Экран списка питомцев и профиля
* **Push-уведомления**: • Интеграция FCM/APNs для мобильных устройств • Тестовая отправка уведомления о бронировании

**Спринт 10 (2 недели)**

* **Mobile App (часть 2)**: • Экран бронирования слота • Экран просмотра и создания PetSittingAd • Встроенный чат в мобильном клиенте
* Синхронизация с внешним календарём (опционально)

**Спринт 11 (2 недели)**

* **Тестирование и покрытие**: • Покрытие Unit/Integration тестами (backend + frontend) • E2E-тесты пользовательских сценариев (Cypress/Appium) • Security Audit: проверка уязвимостей зависимостей
* Документация API (Swagger/Redoc)

**Спринт 12 (2 недели)**

* **Staging → Production**: • Настройка staging-среды • Миграция данных, smoke tests • Обучение команды, подготовка инструкций • Выкатка и мониторинг пострелизного состояния

**Итого:** 12 спринтов (~6 месяцев) охватывают полный цикл разработки MVP: от настройки окружения до продакшн‑релиза с тестированием и документацией.