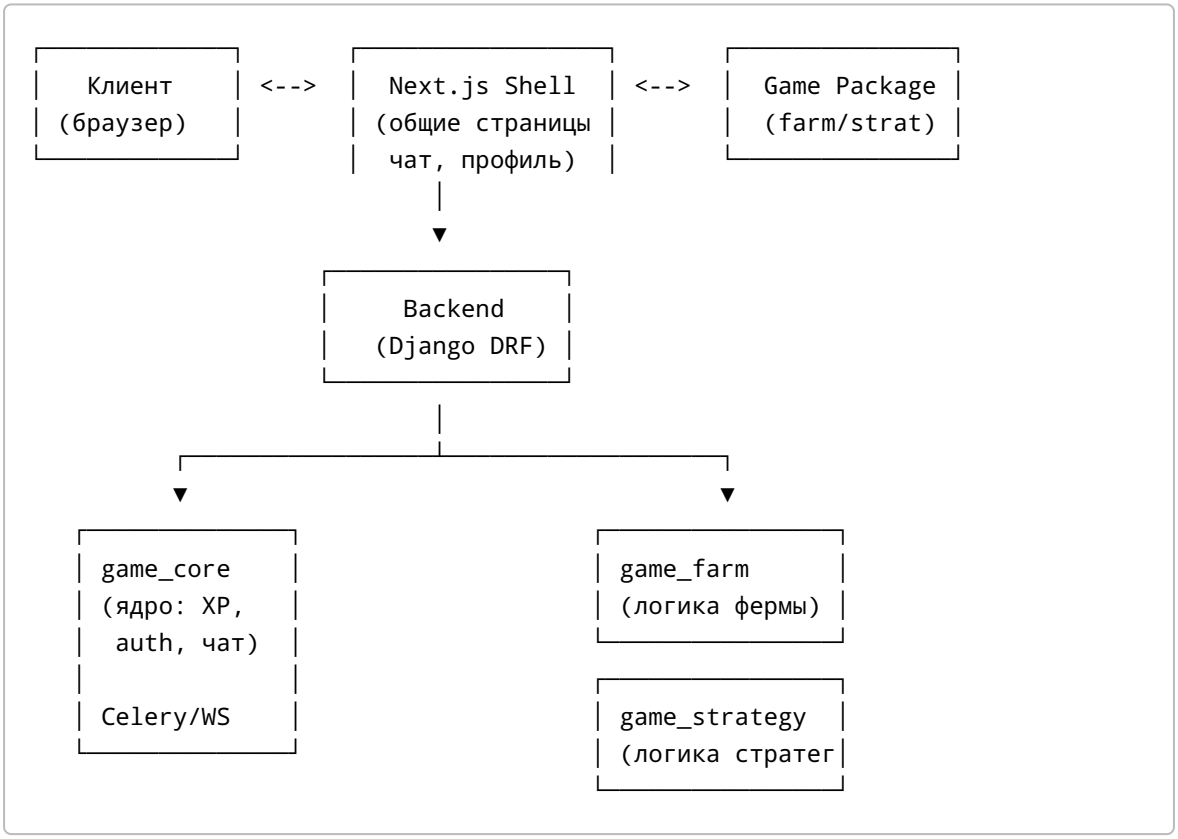
# Визуализация архитектуры бэка и фронта

Ниже — наглядные схемы и описания, которые помогут представить, как связаны ядро, плагины игр и фронт.

---

## 1. Общая картина




---

## 2. Потоки данных

### Простой сценарий (одиночная игра)

1. Игрок делает действие → фронт отправляет его на бек ( `/attempts` ).
2. Django-плагин игры проверяет, можно ли → возвращает `passed/score` .
3. Ядро обновляет прогресс: XP, бейджи.
4. Через WebSocket фронт сразу получает событие `progress.updated` .
5. Celery периодически пересчитывает лидерборды, streak и т.п.

### Кооп (Node.js)

1. Игроки соединены с Node-сервером (быстрая синхронизация).
2. После матча Node шлёт **webhook** → Django.
3. Django обновляет прогресс, вызывает Celery для XP/бейджей.

4. Django пушит обновления игрокам через WS.

---

## 3. Пример архитектуры фронта

```
frontend/
  app/                    # Next.js Shell (меню, профиль, чат)
    play/[gameSlug]/[id]  # роут для запуска конкретной игры
    profile/              # профиль пользователя
    catalog/              # каталог игр и сценариев
    auth/                 # страницы логина/регистрации/восстановления
    about/                # страница «О нас»
    terms/                # условия использования
    privacy/              # политика конфиденциальности
    pricing/              # тарифы/подписки
    blog/                 # блог/новости
    faq/                  # часто задаваемые вопросы
    contact/              # обратная связь/форма контакта
  games/
    farm/                 # пакет игры «ферма»
    strategy/             # пакет игры «стратегия»
  shared/
    sdk-game/             # общий интерфейс GameClient и WS-хуки
```

Shell загружает игру по slug, инициализирует её и подключает к WS.

Redux хранит все данные по проекту;
Состояние игры(пауза, инвентарь и прочее) хранит Zustand, тянет все это ReactQuery

---

## 4. Пример архитектуры бэка

```
backend/
  apps/
    game_core/        # ядро: пользователи, XP, чат, API /attempts
    game_farm/        # ферма: свои модели (Crop, Tile), правила
    game_strategy/    # стратегия: свои модели (Unit, Map, Order)
    users/            # аутентификация, профили
    content/          # лингвистическая библиотека: Phrase/GrammarPattern/
Intent/Lexeme; Game*-overrides для конкретных игр (перегрузки фраз/интентов/
лексем)
    analytics/        # события, outbox, экспорт бизнес-событий (как игрок
учится, как часто заходит, сколько ошибок)

  celery/             # задачи: начисления XP, лидерборды, рассылка
  channels/           # WebSocket consumers (чат, прогресс, игры)
```

```
payments/           # биллинг, тарифы (MVP-заглушка)
media/              # загрузка/хранение аудио, ассетов
```

- Для `content/` используем **гибрид**: общая библиотека фраз/интентов/грамматик + перегрузки на уровне игры.
- Правило разрешения: сначала Game*-overrides для `game_slug`, иначе — каноническая запись из общей библиотеки.

## 5. Где что считается

- **XP/бейджи/статистика** — ядро Django, хранится в Postgres.
- **Обновления в браузер** — WS (Channels) стримит события.
- **Тяжёлые расчёты** (лидерборды, streak) — Celery.
- **Игровой real-time** (кооп, стратегия) — Node.js сервер.

---

## 6. Итоговое сравнение

- **Ядро** = аккаунты, прогресс, чат, лидерборды, платежи.
- **Игровые плагины** = разные механики, свои правила.
- **Фронт** = shell + пакеты игр, динамически подгружаемые.
- **Celery** = асинхронные фоновые задачи.
- **WebSocket** = быстрые пуш-события игроку.
- **Node.js** = real-time сервер для тяжёлых кооп-игр.

---

# Визуализации архитектуры (BE/FE)

## A. Контекст (уровень C4-1)

```
[Пользователь]
   │
   ├─ HTTP(S) →  Next.js Shell (SSR/CSR)
   │                │
   │                ├─ REST → Django DRF  — Postgres/Redis/S3 → Данные
   │                ├─ WS   → Django Channels (чат/прогресс)
   │                └─ OAuth → Провайдеры входа (Google/GitHub…)
   │
   └─ WS (кооп) →  Node Game Server (опционально, для тяжёлых игр)
                        │
                        └─ Webhook/Queue → Django (фикс прогресса)
```

## B. Компоненты бэкенда (уровень C4-2)

```
[Django Project]
  ├─ apps/game_core      — ядро, прогресс, реестр игр, общие API
```

```
  ├─ apps/game_farm      — плагин фермы (валидация, состояние, WS consumer)
  ├─ apps/game_strategy  — плагин стратегии (свои правила/модели)
  ├─ apps/realtime       — Channels routing, chat, presence
  ├─ apps/users          — auth, профили
  ├─ apps/analytics      — события, outbox
  └─ Celery workers      — начисления, лидборды, почта, импорты

Infra: Postgres (OLTP), Redis (кеш/каналы/очередь), S3 (ассеты/аудио)
```

## C. Компоненты фронта (Next.js)

```
[Next.js Shell]
  ├─ app/… страницы (лендинг, профиль, чат, /play/[gameSlug]/[levelId])
  ├─ shared/sdk-game   — общий SDK: загрузчик игры, WS-коннектор
  └─ games/
      ├─ farm/        — пакет UI/логики фермы
      └─ strategy/    — пакет UI/логики стратегии
```

## D. Поток данных — одиночная игра (без коопа)

```
Игрок → (голос/текст) → Shell → POST /api/games/:slug/levels/:id/attempts → DRF
      ← результат (passed/score/feedback)
      → Celery (асинхронно начислить XP/бейджи)
      ← WS push от Channels: {type: progress.updated, xp_delta, badges}
```

## E. Поток данных — кооп с Node

```
Игрок ⇄WS ⇄ Node (игровая синхронизация)
                    │
                    ├─ по окончании: Webhook → Django /match-results
                    │                    → Celery: начислить прогресс
                    │                    → Channels WS push игрокам
                    └─ (опция) очередь сообщений: Kafka/Redis/NATS
```

## F. Схема данных прогресса (упрощённо)

```
User(id, email, …)
UserProfile(user_id, xp, streak, level)
Achievement(key, title, rule)
UserAchievement(user_id, key, earned_at)
Attempt(id, user_id, game_slug, level_id, score, passed, created_at)
LeaderboardSnapshot(period, game_slug, ranklist_json)
```

## G. Развёртывание (минимальная топология)

```
[Client]
  │
  ├─ HTTPS → Next.js (Vercel/Node) → статика + SSR
  ├─ HTTPS → Django (ASGI, gunicorn/uvicorn, autoscaling)
  ├─ WSS   → Channels (Daphne/Uvicorn) + Redis
  ├─ HTTPS/WSS → Node Game Server (если нужен)
  └─ Celery Workers ↔ Redis/RabbitMQ

DB: Postgres HA, Files: S3, Monitoring: Sentry + Prometheus/Grafana
```

Если захочешь, сгенерирую те же схемы в PNG/Mermaid/PlantUML и приложу файлы.

---

# Функции продукта и подсистем

## A. Функции продукта целиком (что видит пользователь)

- Регистрация/вход (email, OAuth).
- Профиль игрока (аватар, XP, streak, бейджи, словарь).
- Лендинг/маркетинг-страницы (о продукте, FAQ, цены, политика).
- Каталог игр и сценариев (в каждой игре свои «уровни/раунды/сессии»), поиск по тегам/навыкам, рекомендации.
- Мини-игры (соло, кооп): управление голосом/текстом/кнопками.
- Обратная связь (подсказки, проверка фраз, результат попытки).
- Геймификация (XP, бейджи, лидерборды, streaks).
- Чат (глобальный, групповой, личный).
- Уведомления (email/push).
- Оплата/подписки (позже).
- Настройки (язык интерфейса, голосовые функции, приватность).

## B. Функции бекенда (Django + Celery + Channels)

- Auth API: регистрация, логин, refresh токены, соц. вход.
- Users/Profiles API: чтение/редактирование профиля, прогресса.
- Content API: лингвистическая библиотека (Phrases, GrammarPatterns, Intents, Lexemes) + Game-overrides (GamePhrase/GameIntent/GameLexeme).
- Game API: `POST /attempts`, выдача целей/условий сценария.
- Realtime WS: чат, presence, игровые события.
- Progress сервис: начисление XP, streaks, выдача бейджей.
- Leaderboard сервис (через Celery, периодические задачи).
- Notifications сервис (почта, web push).
- Payments сервис (тарифы, статусы подписки).
- Analytics (event log, outbox events, экспорт в BI).
- Media (загрузка/хранение аудио/ассетов, S3).

## C. Функции фронтенда (Next.js Shell)

- SSR/CSR страниц (лендинг, профиль, каталог, play/*).
- UI для регистрации/авторизации.
- UI профиля (XP, бейджи, streak).
- UI каталога игр/сценариев.
- GameShell: загрузка пакета игры по slug.
- SDK-game: коннектор к WS/REST, интерфейс GameClient.
- Games пакеты: ферма, стратегия и др.
- UI чата (комнаты, лички, глобал).
- State management: React Query (API данные), Zustand (локальный UI).
- i18n: мультиязычность интерфейса.
- Уведомления в UI.

## D. Функции отдельного игрового сервера (Node, опционально)

- Реал-тайм синхронизация состояний игроков (тики).
- Валидация игровых действий в реальном времени.
- Расчёт результатов матча/сессии.
- Отправка результатов в ядро (webhook/очередь).
- Анти-чит и защита от флудов.

---

# Визуализация функций (высокоуровневая)

```
[Пользователь]
  │
  ├─ Веб-интерфейс (Next.js)
  │      ├─ Auth UI / Профиль / Лендинг
  │      ├─ Каталог игр / Игры
  │      ├─ Чат / Уведомления
  │      └─ GameShell → Game пакеты
  │
  ├─ REST API → Django DRF
  │      ├─ Auth / Users / Content
  │      ├─ Game Attempts / Progress
  │      └─ Payments / Analytics
  │
  ├─ WS → Django Channels
  │      ├─ Чат
  │      ├─ Обновления прогресса
  │      └─ Игровые события (простые игры)
  │
  └─ WS → Node Game Server (для кооп)
         └─ Webhook/Queue → Django (фикс прогресса)
```

---

# Карта функций продукта и распределение по частям

Ниже — полный перечень функций на уровне **продукта** и их разложение по **программным частям** (FE/BE/Realtime/Инфра). В конце — мини-визуализации.

## 1) Функции продукта целиком (что должен уметь сервис)

- **Онбординг и доступ**: регистрация по email/OAuth, подтверждение почты, восстановление пароля, базовые настройки, согласия (ToS/Privacy).
- **Каталог игр и сценариев**: список игр и их сценариев (в каждой игре свои «уровни/раунды/сессии»), поиск по тегам/навыкам, рекомендации.
- **Игры**: запуск сценария, голос/текст ввод, подсказки, пауза/рестарт, сохранения, туториал.
- **Оценка и фидбек**: валидация ответа, подсветка ошибок, объяснение правила, примеры.
- **Прогресс и геймификация**: XP, прогресс профиля, streak, бейджи, недельные/общие лидерборды, персональные цели.
- **Социалка**: чат (личный/группы/глобальный), статус online/typing, жалобы/мут.
- **Кооп (опционально)**: лобби/приглашения, матчмейкинг, синхронизация, результаты матча.
- **Контент-менеджмент**: создание/импорт фраз/интентов/лексем, предпросмотр, версии, публикация.
- **Лендинг и маркетинг**: публичные страницы, блог/новости, SEO/OG, UTM-треккинг, формы обратной связи.
- **Оплата (позже)**: подписки/планы, пробный период, биллинг, ограничения по тарифам.
- **Аналитика**: события (game.attempt, progress.updated), воронки, retention, A/B.
- **Безопасность/легал**: GDPR-features (export/delete), политика хранения аудио, модерация чатов.

## 2) Разложение по программным частям

### Frontend (Next.js Shell)

- Роутинг, SSR/CSR, лендинг/профиль/каталог, /play/[gameSlug]/[scenarioId].
- Auth-флоу (UI), хранение токена (httpOnly cookie), защищённые страницы.
- UI-кит: формы, таблицы, модальные окна, тосты; локализация (en/ru).
- Интеграция голосового ввода (Web Speech API), текстовый ввод, TTS ответ.
- Экран прогресса, бейджи, лидерборды; чат-клиент (WS), presence/typing.
- Телеметрия (Sentry), event-треккинг (нажатия, латентность распознавания).

### Game Packages (фронтовые пакеты игр)

- Рендер конкретной игры (ферма/стратегия/квест), локальный стейт (Zustand/XState).
- Протокол с бэком: команды/события, визуализация фидбека, таймеры.
- Встроенный туториал/хинты, минимальный офлайн-режим (опционально).

### Backend — DRF (ядро)

- Users/auth/profiles (JWT + refresh rotation), настройки, согласия.

- Content: Phrases/GrammarPatterns/Intents/Lexemes + Game-overrides; (если у конкретной игры