11. Список ADR (Architecture Decision Records)

ADR-1: Архитектурный стиль — Микросервисы

Контекст

- Приложение включает в себя функционал для учёта тренировок, социальной взаимодействия, геймификации, рекомендаций, интеграции с магазином.
- Нужна высокая масштабируемость и гибкость в развитии: предполагается глобальная аудитория, разные виды спорта, регулярные обновления.

Решение

- Принята **микросервисная архитектура**, в которой каждый ключевой блок (User Management, Workout & Activity, Social & Group, Gamification, Inventory, Promotions & News, Analytics & Recs и т.д.) функционирует как независимый сервис.

Обоснование

1. Локализация функциональных изменений: можно развивать каждый сервис своей командой, не ломая другие компоненты.
2. Масштабируемость по нагрузке: если растёт нагрузка на социальные функции, увеличиваем лишь Social & Group Service, не затрагивая другие.
3. Устойчивость к отказам: сбой одного микросервиса не обрушит всё приложение (при условии грамотного проектирования взаимодействий).

Последствия

- Увеличивается сложность DevOps: требуются инструменты оркестрации (Kubernetes), мониторинга, логирования, согласования контрактов между сервисами.
- Повышенные требования к квалификации команды, умению работать с распределёнными системами.

ADR-2: Облачная стратегия — Мультиоблако / Гибридное решение

Контекст

- Компания уже использует несколько облачных провайдеров (Yandex, AWS, Azure, GCP) в разных подразделениях.
- Необходимо обеспечить гибкость и избежать привязки к одному вендору (vendor lock-in).
- Важно поддержать географическую экспансию (распределённые дата-центры, разные законодательные требования к хранению данных).

Решение

- Развёртывать микросервисы и инфраструктуру (Kubernetes, базы данных, Data Lake) в разных облаках, используя подход IaaC (Infrastructure as Code) и единый слой оркестрации (K8s + Terraform/Ansible).

Обоснование

1. Позволяет «раскидать» сервисы по разным регионам ближе к конечным пользователям.
2. Минимизирует риск полного даунтайма, если у одного провайдера проблемы.
3. Можно выбирать конкретного облачного провайдера для конкретных задач (лучший ML-стек, лучшие цены на хранение и т.д.).

Последствия

- Усложнение инфраструктуры: нужно соблюдать сетевую связность, единый pipeline деплоя, учитывать разницу в сервисах провайдеров.
- Увеличение расходов на специалистов и DevOps-инструменты, чтобы поддерживать много облачных сред.

ADR-3: Модель данных — Гибридный подход (SQL, NoSQL, Time Series)

Контекст

- Данные разнородны: профили пользователей и транзакционные операции (SQL), социальные посты и лайки (NoSQL), временные ряды тренировок и сенсоров (Time Series).
- Высокие объёмы данных (активная социальная лента, IoT-метрики).

Решение

- Использовать **реляционные базы (SQL)** для критичных транзакций (User Management, заказы, инвентарь).
- **NoSQL** (MongoDB, DynamoDB или аналог) для хранения больших объёмов активности (ленты, комментариев, лайков).
- Специализированные **Time Series**-базы (InfluxDB, TimescaleDB) для телеметрии тренировок (пульс, шаги, темп) и других временных рядов.

Обоснование

1. Реляционная БД удобна для связных транзакций и жёсткой схемы (профили, магазин).
2. NoSQL подходит для «живых» данных соцсети, обладающих нестрогой структурой и большим объёмом.
3. Time Series-решения оптимизированы под запросы, связанные со временем (агрегации, фильтрация, downsampling).

Последствия

- Увеличивается архитектурная сложность: надо управлять несколькими типами хранилищ, синхронизировать их резервное копирование.
- Команда должна владеть несколькими СУБД и знать, как правильно организовывать миграции данных.

ADR-4: Подход к аналитике — Комбинированное решение (Batch + Streaming)

Контекст

- Необходимы персональные рекомендации в реальном времени (аналитика онлайн), а также долгосрочные отчёты и ML-модели, основанные на больших объёмах исторических данных.
- Приложение собирает события от пользователей (тренировки, соцактивность) и от IoT-устройств.

Решение

- Использовать **потоковую обработку (Streaming)** (Kafka, Flink/Spark Streaming) для оперативных метрик, моментальных уведомлений, лидербордов и геймификации.
- Параллельно организовать **Batch-процессы** на базе Spark/Hadoop (или облачных сервисов), которые будут обрабатывать исторические данные и обучать ML-модели.

Обоснование

1. Streaming позволяет мгновенно реагировать на события (прогресс пользователя, достижение целей, калькуляция рейтингов).
2. Batch даёт возможность формировать предиктивные модели, сложные отчёты, сегментацию пользователей.

Последствия

- Нужно обеспечить надёжную шину сообщений (Kafka) и настроить конвейеры ETL/ELT.

- Возникают дополнительные затраты на инфраструктуру Big Data, специалистов по Data Engineering и ML.

ADR-5: IoT-интеграция — Шлюз (IoT Hub)

Контекст

- Приложение работает с фитнес-трекерами (Garmin, Polar, Apple Watch и др.), которые передают различные форматы данных.
- Необходимо стандартизировать сбор данных в режиме реального времени и быстро масштабироваться при росте числа подключённых устройств.

Решение

- Использовать **IoT Hub** (возможно, готовое решение у облачного провайдера либо собственный слой) как единый шлюз, куда устройства отправляют метрики.
- IoT Hub отвечает за авторизацию устройств, преобразование форматов данных, первичную фильтрацию и передачу в микросервис Workout & Activity или стриминговую систему.

Обоснование

1. Облегчает подключение новых типов устройств, не нужно «патчить» сервис Workout & Activity при каждой интеграции.
2. Масштабируется независимо от остальной логики.

Последствия

- Повышается сложность в плане DevOps и мониторинга: IoT Hub требует отдельного контура для высокой пропускной способности, надёжности и безопасности.
- Необходимо поддерживать адаптеры/коннекторы под разные производители устройств.