Техническое задание на разработку партнёрской программы

Пояснительная записка

Партнёрская программа представляет собой платформу (далее – «сервис»), связывающую веб-мастеров (партнёров, издателей или рефереров) с рекламодателем. Сервис должен обеспечивать стабильную работу при пиковой нагрузке до 50 запросов в секунду. В ядро сервиса должен быть заложен стэк технологий позволяющий в дальнейшем выдерживать более существенные нагрузки с возможностью модульного обновления.

Структура сервиса должна включать:

- обработка трафика;
- интеграция с основным сайтом рекламодателя;
- личный кабинет партнёра;
- личный кабинет рекламодателя (администратор).

UI/UX должен быть интуитивно понятным, удобным в использовании дизайн которого должен соответствовать с брендбуком компании.

1. Общие сведения

1.1. Название проекта

Партнёрская программа для агрегатора авиабилетов.

1.2. Цель проекта

Создать онлайн-платформу (далее – «сервис»), которая связывает веб-мастеров (партнёров) с рекламодателем (агрегатор авиабилетов). Сервис должен:

1. Обрабатывать и распределять трафик.
2. Предоставлять инструменты для управления партнёрскими ссылками, кампаниями и отчётами.
3. Иметь удобный интерфейс личных кабинетов (как для партнёров, так и для рекламодателя/администратора).
4. Гарантировать масштабируемость и стабильность работы при пиковых нагрузках до 50 запросов в секунду (с возможностью дальнейшего роста).

1.3. Основание для разработки

- Пояснительная записка;
- Технические и бизнес-требования заказчика (агрегатор авиабилетов);
- Брендбук компании для соответствия дизайна.

1.4. Участники проекта

- **Заказчик (рекламодатель)**: Купибилет (агрегатор авиабилетов).
- **Исполнитель**: команда разработки RomyRom (Backend, Frontend, DevOps, QA).
- **Партнёры** (веб-мастера): пользователи сервиса, работающие на стороне издателей/рефереров.
- **Администратор** (со стороны Заказчика): управляет всей системой.

2. Требования к сервису

2.1. Функциональные требования

1. **Обработка трафика**
    - Принимать партнёрские запросы (клики, переходы по ссылкам и т.п.).
    - Выполнять идентификацию партнёра и рекламной кампании (по передаваемым параметрам).
    - Сохранять/логировать данные о переходах в базе для дальнейшей аналитики.
2. **Интеграция с основным сайтом рекламодателя**
    - Обеспечить автоматическую передачу данных (по кликам, заказам, бронированиям и т.д.) в систему учёта агрегатора (через API или напрямую в БД при согласованном доступе).
    - Возможность использования webhook'ов или очередей для асинхронной интеграции.
3. **Личный кабинет партнёра**
    - Регистрация и авторизация партнёров.
    - Управление рекламными кампаниями, ссылками, промо-материалами (баннеры, формы поиска и т.д).
    - Просмотр статистики кликов, заказов, доходов.
    - Редактирование личных данных, настройки выплат.
4. **Личный кабинет рекламодателя (администратор)**
    - Управление партнёрами (подтверждение регистрации, модерация).
    - Управление рекламными кампаниями и правилами.
    - Просмотр и выгрузка статистики по всем партнёрам.
    - Финансовая часть (расчёты, закрытие периодов, отчёты).
5. **UI/UX**
    - Интуитивно понятный интерфейс.
    - Соответствие брендбуку рекламодателя (цветовая гамма, лого, шрифты).
    - Адаптивный дизайн для корректной работы на мобильных и планшетах.

2.2. Нефункциональные требования

1. **Производительность**
    - Сервис должен обрабатывать до 50 запросов в секунду в пиковые моменты без деградации качества.
    - Среднее время отклика должно оставаться в пределах 200–300 мс (при условии нормальной сетевой доступности).
2. **Масштабируемость**
    - Возможность вертикального и горизонтального масштабирования.
    - Поддержка модульного обновления компонентов (версионность API, контейнеризация и т.д.).
3. **Надёжность и отказоустойчивость**
    - Использовать механизмы резервирования (кеширование, система очередей).

4. **Безопасность**
    - Безопасное хранение и передача данных (HTTPS, шифрование данных, защита конфигураций).
    - Защита от CSRF/XSS/SQL-инъекций и прочих распространённых уязвимостей.
5. **Логирование и мониторинг**
    - Сбор и хранение логов запросов и ошибок.
    - Отчётность по сбоям и критическим ошибкам.
6. **Документированность**
    - Техническая документация для разработчиков и администраторов.

3. Архитектура решения

Для обеспечения гибкости и масштабируемости планируется использовать **микросервисный** или **модульный** подход. В рамках данного ТЗ выделяются основные блоки:

1. **Сервис обработки трафика (Tracking Service)**
   - Принимает запросы, валидирует, ставит в очередь для последующей обработки или сразу записывает в БД.
   - Отвечает за быструю реакцию (может работать через кеш/очереди, чтобы не нагружать основное приложение).
2. **Сервис статистики и аналитики**
   - Считает клики, заказы, конверсии.
   - Реализует отчётность в личных кабинетах партнёра и администратора.
3. **Сервис личных кабинетов**
   - Личный кабинет партнёра (UI + API).
   - Личный кабинет рекламодателя/администратора (UI + API).
4. **Интеграционный сервис**
   - Обеспечивает взаимодействие с основным сайтом рекламодателя.
   - Может работать по Event-driven модели (RabbitMQ) или по REST-соединениям.
5. **Система очередей**
   - RabbitMQ (или альтернативы) для асинхронной передачи данных и балансировки нагрузки.
6. **Кеширующий слой**
   - Redis для быстрого доступа к часто запрашиваемой информации, сессиям, кешу статистики и т.д.

4. Технологический стэк

Ниже описаны два варианта стэка для серверной части с общими элементами для очередей и кеширования.

4.1. Вариант 1: Node.js + Nest.js

1. **Node.js**
   - Популярен в high-load проектах, большое комьюнити, множество библиотек.
2. **Nest.js**
   - Удобный модульный фреймворк на TypeScript для быстрой разработки и структурированного кода.
   - Встроенные средства для работы с WebSockets, GraphQL (при необходимости), валидации, аутентификации и т.д.
3. **Redis**
   - Для кеширования запросов, хранения сессий и быстрых операций.
4. **RabbitMQ**
   - Создание очередей для асинхронной обработки (например, записи статистики, уведомлений).
5. **Nuxt.js**
   - Фреймворк на базе Vue.js для фронтальной части.
   - Может работать как SPA или PWA, поддерживает SSR (при необходимости SEO).

4.2. Вариант 2: Franken (на базе Laravel) + PHP

1. **Franken (https://frankenphp.dev/)**
   - Новая среда, позволяющая запускать Laravel-проекты более эффективно (в режиме «сервер как go/php mix»).
   - Даёт преимущества по производительности за счёт использования «реактивной» модели поверх Swoole, RoadRunner или аналога.
2. **Laravel**
   - Популярный, хорошо документированный фреймворк PHP.
   - Быстрая разработка, большое комьюнити.
3. **RabbitMQ**
   - Та же система очередей, что и в первом варианте.
4. **Redis**
   - Аналогичные преимущества для кеширования.
5. **Nuxt.js**
   - Аналогично используется для фронта (SPA/PWA).

5. Детали реализации

5.1. Структура баз данных

- **Основная БД** (реляционная: PostgreSQL, MySQL или MariaDB):
    - Сущности: пользователи (партнёры, администраторы), кампании, клики, заказы/конверсии, выплаты.
    - Таблицы связей (m:n) для кампаний и партнёров (если партнёр может участвовать в нескольких кампаниях).
    - Логи для учёта активности (можно вынести в отдельную таблицу или БД).
- **Redis**: хранение сессий, кеш популярных запросов (статистику в реальном времени).

5.2. Взаимодействие компонентов

1. **Пользователь (партнёр)** →\rightarrow→ **Сервис обработки трафика**
    - GET/POST запрос с метками (utm_source, utm_campaign и т.д.).
    - Валидация и постановка в очередь (RabbitMQ) или запись в БД.
2. **Сервис статистики**
    - Считывает данные из очереди, обрабатывает, агрегирует.
    - Сохраняет в БД для формирования отчётов.
3. **ЛК партнёра / ЛК администратора**
    - Обращается к API статистики, выводу аналитики, управляет сущностями (кампании, выплаты).
4. **Интеграция с сайтом рекламодателя**
    - Передача данных о бронированиях/покупках авиабилетов для подтверждения конверсий.
    - Возможен webhook/событийная модель.

6. UI/UX

1. **Фронтенд** (Nuxt.js)
    - Прототипирование интерфейсов с учётом брендбука.
    - Адаптивная верстка (mobile-first).
2. **Навигация**
    - Панель навигации для партнёра: «Кампании», «Статистика», «Выплаты», «Настройки».
    - Панель навигации для админа: «Партнёры», «Статистика», «Управление кампаниями», «Финансы», «Настройки системы».
3. **Удобство использования**
    - Минимальное количество шагов для получения нужной информации.
    - Графики и диаграммы для аналитики.

7. Безопасность

1. **Авторизация и аутентификация**
    - JWT/Passport (Node.js) или Sanctum/Passport (Laravel).
    - Защита от брутфорса (ограничение на количество попыток логина).
2. **Политика доступа**
    - Разделение ролей: партнёр, администратор, менеджер.
    - Гранулярный доступ к CRUD-операциям.
3. **Шифрование**
    - TLS/SSL-сертификаты (HTTPS).
    - Hashing паролей (bcrypt, Argon2).
4. **Защита от уязвимостей**
    - Валидация данных на сервере (DTO или FormRequest).
    - Использование встроенных механизмов защиты (CSRF-токены, CORS-настройки).

8. Производительность и масштабирование

1. **Горизонтальное масштабирование**
    - Возможность запускать несколько экземпляров сервисов обработки трафика и статистики.
    - Балансировка нагрузки (NGINX, HAProxy).
2. **Мониторинг**
    - Метрики через Prometheus + Grafana или ELK-стек.
    - Уведомления о сбоях (Slack, email, PagerDuty).
3. **Тестирование на нагрузку**
    - Использование инструментов (JMeter, k6) для проверки производительности до 50 rps и выше.

9. План выполнения работ

1. **Аналитика и проектирование**
    - Уточнение бизнес-требований.
    - Проектирование архитектуры и БД.
2. **Подготовка окружения**
    - Настройка CI/CD, Docker/Kubernetes (при необходимости).
    - Выбор и настройка репозиториев (Git).
3. **Разработка MVP**
    - Базовая версия сервиса обработки трафика.
    - Простейший ЛК партнёра и ЛК администратора.
    - Интеграция с очередью (RabbitMQ) и кешем (Redis).
4. **Тестирование**
    - Юнит-тесты, интеграционные тесты, e2e-тесты.
    - Нагрузочное тестирование.
5. **Доработка и оптимизация**
    - Рефакторинг по результатам тестов.
    - Добавление недостающих модулей и отчётности.
6. **Внедрение и релиз**
    - Перенос на продуктивный сервер.
    - Настройка мониторинга и резервного копирования.
7. **Сопровождение**
    - Техническая поддержка, устранение ошибок.
    - Дальнейшее развитие функционала.

10. Тестирование

1. **Модульное тестирование**
    - Тесты на функции и методы (Jest для Node.js или PHPUnit/Pest для Laravel).
2. **Интеграционное тестирование**
    - Проверка взаимодействия микросервисов и корректности записи в БД.
3. **E2E-тестирование**
    - Поведение пользователя в ЛК (Cypress или аналог).
4. **Нагрузочное тестирование**
    - JMeter, k6 для проверки пиковых нагрузок.

11. Документация

1. **API-документация**
    - Swagger/OpenAPI (Nest.js) или Laravel/OpenAPI.
    - Описание конечных точек, форматов запроса/ответа, кодов ошибок.
2. **Техническая документация**
    - Архитектура, схемы БД, инструкции по развёртыванию.
3. **Пользовательская документация**
    - Гайды для партнёров и администраторов (вопрос-ответ, типовые действия).

12. Рекомендации по улучшению стэка

1. **Рассмотреть использование TypeScript (в случае Node.js)**
    - Nest.js официально ориентирован на TypeScript, что облегчает рефакторинг и поддержку в долгосрочной перспективе.
2. **Рассмотреть переход на Kubernetes**
    - Если предполагается масштабирование микросервисов и наличие распределённой инфраструктуры, Kubernetes может упростить управление контейнерами и автоматизацию деплоя.
3. **Использовать менеджер процессов (PM2, Supervisor или системный сервис)**
    - Для Node.js — PM2. Для Laravel/Franken — Supervisor или аналог. Это повышает надёжность и управляемость сервисов.
4. **Оптимизация логирования и мониторинга**
    - Выделить отдельный сервис для сбора логов (например, ELK-стек: Elasticsearch, Logstash, Kibana).
    - Это поможет быстрее реагировать на инциденты и собирать аналитику.
5. **Подумать о Kafka как альтернативе RabbitMQ**
    - Если объём данных и количество событий будут расти очень существенно (сотни – тысячи rps), Kafka может подойти лучше для «тяжёлой» событийной шины.
6. **CI/CD**
    - Настроить автоматическую сборку и тестирование (GitLab CI, Jenkins, GitHub Actions).
    - Ускоряет релизы и гарантирует, что в продакшн попадает проверенный код.
7. **Рассмотреть Next.js вместо Nuxt.js (при желании перейти на React)**
    - Если в команде больше опыта в React, то Next.js может быть предпочтительнее. Аналогичная архитектура (SSR, SSG), но экосистема React.

13. Заключение

Данное техническое задание описывает основные требования и архитектурно-технологические решения для разработки партнёрской программы агрегатора авиабилетов. Предложенный стэк (Node.js/Nest.js или Franken/Laravel, RabbitMQ, Redis, Nuxt.js) покрывает ключевые аспекты производительности, надёжности и удобства разработки.

Дополнительные рекомендации по улучшению стэка направлены на повышение отказоустойчивости, расширение возможностей масштабирования и упрощение процессов DevOps.

Следующие шаги: согласовать детали ТЗ с заказчиком, финализировать выбор технологии (Node.js или Franken/Laravel) в зависимости от предпочтений и экспертизы команды, утвердить план работ и приступить к разработке MVP.