УДК 004.7

РАЗРАБОТКА МИКРОСЕРВИСНОЙ ВЕБ-ПЛАТФОРМЫ ОБЪЯВЛЕНИЙ О ПРОДАЖЕ ТОВАРОВ

Гук Д.Ю., студент гр. ПИб-222, IV курс,
Крутский Д.Л., старший преподаватель
Кузбасский государственный технический университет имени Т.Ф. Горбачева
г. Кемерово

Площадки объявлений о продаже товаров, такие как Avito, OLX, Юла — стали неотъемлемой частью рынка вторичных продаж. За каждой из них стоит сложная техническая инфраструктура, способная одновременно обслуживать миллионы запросов. При разработке аналогичной платформы в рамках данной работы встал закономерный вопрос: какая архитектура и какой технологический стек позволят реализовать систему, которая будет не только выполнять поставленные сейчас задачи, но и поддаваться расширению в будущем.

В качестве основного архитектурного подхода выбрана микросервисная архитектура. Её главное преимущество перед монолитом — возможность разрабатывать, тестировать и масштабировать отдельные части системы независимо [1]. Платформа реализует полный цикл работы с объявлениями: регистрацию и аутентификацию пользователей, публикацию и поиск объявлений, переписку покупателя с продавцом и автоматическую ML-модерацию.

Система разбита на шесть сервисов: auth-service отвечает за аутентификацию и выпуск JWT-токенов; ad-service управляет объявлениями, категориями и избранным; chat-service обеспечивает переписку между пользователями; moderation-service координирует очередь на модерацию и сохраняет её результаты; gateway-service выступает единой точкой входа для всех внешних запросов; ml-service запускает модель оценки качества текста. Сервисы взаимодействуют через REST поверх HTTP с передачей данных в JSON.

Выбор языка для серверных сервисов оказался одним из ключевых решений проекта. Рассматривались три варианта: Go, Java и Python. Java была отброшена из-за значительного потребления памяти при старте JVM — запускать шесть экземпляров одновременно на учебном сервере нецелесообразно. Python проигрывает по производительности в IO-нагрузке. В итоге остановились на Go версии 1.21: язык компилирует каждый сервис в единственный бинарный файл без внешних зависимостей, потребляет минимум оперативной памяти и запускается за миллисекунды [2]. Встроенная модель горутин (gorutine) и каналов при этом позволяет без лишних усилий писать конкурентный код.

Среди HTTP-фреймворков для Go выбор пал на Gin версии 1.10. Стандартная библиотека net/http хороша, но требует много шаблонного кода для группировки маршрутов и middleware-цепочек. Echo и Gin примерно одинаковы по возможностям, однако Gin имеет большую аудиторию и более ста-

бильный API [3]. На практике это выразилось в удобной группировке эндпоинтов по версиям и простом добавлении middleware для логирования и проверки токенов.

ML-сервис вынесен в отдельный контейнер на Python 3.11 — здесь смена языка оправдана. Весь пайплайн от загрузки данных до сохранения модели покрывается тремя библиотеками: pandas для предобработки датасета, scikit-learn для обучения классификатора и joblib для сериализации готовой модели [4]. В качестве веб-фреймворка выбран FastAPI версии 0.115: в сравнении с Flask он изначально асинхронный, автоматически генерирует OpenAPI-документацию и не требует дополнительной настройки ASGI-сервера — uvicorn запускается одной командой.

Модель классификации объявлений строится на TF-IDF-векторизаторе с 15 000 признаками и n-граммами от 1 до 3, поверх которого обучается логистическая регрессия с балансировкой классов (class_weight='balanced'). Такой выбор продиктован спецификой задачи: объявления — короткие тексты на русском языке, а трансформерные модели вроде BERT требуют на порядок больше вычислительных ресурсов при сопоставимом качестве на подобных задачах [5]. На тестовой выборке модель уверенно различает три класса: approved, needs_review и rejected.

Фронтенд написан на React версии 18.2 с маршрутизацией через React Router 6.22. От Redux или Zustand намеренно отказались — объём глобального состояния в приложении невелик, и Context API справляется без лишних абстракций. Webpack в проекте заменён на Vite 5.1: в режиме разработки он не собирает весь бандл заново, а отдаёт модули напрямую через нативные ES-модули браузера, что сокращает время перезапуска с нескольких секунд до долей. В production-сборке итоговые файлы раздаёт Nginx, он же проксирует /api на gateway-сервис.

В роли СУБД используется PostgreSQL версии 16. Рассматривалась и MongoDB — у неё гибкая схема, что удобно для объявлений с разными наборами атрибутов. Тем не менее выбор остановился на PostgreSQL: полнотекстовый поиск, поддержка JSON-полей и ACID-транзакции закрывают все потребности проекта, а реляционная модель лучше подходит для связанных сущностей — пользователи, объявления, сообщения, категории [6]. В текущей реализации все сервисы работают с одной базой через отдельные схемы, каждый управляет своими миграциями самостоятельно.

Контейнеризация выполнена через Docker с многоэтапной сборкой: на первом этапе golang:1.21-alpine компилирует бинарник, на втором — образ на базе alpine:3.19 его принимает. Итоговый размер образа для Go-сервиса составляет около 15 МБ против нескольких сотен у JVM-варианта. Docker Compose связывает все сервисы в единую сеть, прописывает healthcheck-зависимости и пробрасывает наружу три порта: gateway (8000), фронтенд (3000) и PostgreSQL (5432) — последний только для отладки. Таблица 1 обобщает принятые технологические решения.

Применение JWT для аутентификации позволило избежать хранения сессий на сервере: каждый запрос несёт подписанный токен, который gateway проверяет локально перед проксированием во внутренние сервисы. Это упрощает горизонтальное масштабирование — любой дополнительный экземпляр gateway проверяет токены без синхронизации состояния с остальными [7].

У принятой архитектуры есть осознанные ограничения. Единая база данных для всех сервисов нарушает канонический принцип изоляции микросервисов и делает её узким местом при масштабировании. Синхронное межсервисное взаимодействие через REST означает, что сбой в одном сервисе может каскадно затронуть другие. В рамках проекта эти компромиссы оправданы — они снижают сложность реализации; в производственной системе логичен переход к паттернам saga, circuit breaker и message broker (RabbitMQ или Apache Kafka).

В ходе работы спроектирована и частично реализована микросервисная веб-платформа объявлений. Стек формировался исходя из трёх приоритетов: эффективное использование ресурсов на сервере, зрелая экосистема для ML-задач и быстрый цикл разработки на фронтенде. Go решает первую задачу, Python с FastAPI и scikit-learn — вторую, React с Vite — третью. PostgreSQL и Docker замыкают инфраструктуру. Сделанные архитектурные компромиссы понятны и при необходимости устранимы без переписывания системы с нуля.

Таблица 1 – Технологический стек платформы

Слой системы	**Технологии**	**Обоснование выбора**
Клиентская часть	React 18.2, React Router 6.22, Vite 5.1	Компонентный подход, быстрая пересборка через HMR, развитая экосистема
Серверная часть	Go 1.21, Gin 1.10	Малый расход памяти, один бинарник на сервис, удобные middleware-цепочки
ML-сервис	Python 3.11, FastAPI 0.115, scikit-learn 1.5	Полный ML-пайплайн тремя библиотеками, встроенный ASGI, автодокументация
База данных	PostgreSQL 16	ACID, полнотекстовый поиск, JSON-поля, привычная реляционная модель
Инфраструктура	Docker, Docker Compose, Nginx	Воспроизводимое окружение, изоляция сервисов, healthcheck-зависимости
Аутентификация	JWT (golang-jwt/jwt 5.2)	Stateless-проверка на шлюзе, не требует хранения сессий
API	REST, JSON, HTTP	Простота интеграции и отладки, нет лишних зависимостей

Список литературы:

1. Ньюмен С. Создание микросервисов / С. Ньюмен; пер. с англ. П. Тимофеев. – 2-е изд. – Санкт-Петербург : Питер, 2022. – 624 с.
2. Донован А., Керниган Б. Язык программирования Go / А. Донован, Б. Керниган; пер. с англ. – Москва : ООО «И.Д. Вильямс», 2016. – 432 с.
3. Gin Web Framework [Электронный ресурс] // GitHub : [сайт]. – URL: https://github.com/gin-gonic/gin (дата обращения: 17.03.2026).
4. Рамирес С. FastAPI [Электронный ресурс] // Официальная документация FastAPI : [сайт]. – URL: https://fastapi.tiangolo.com (дата обращения: 17.03.2026).
5. Мэннинг К., Рагхаван П., Шюце Х. Введение в информационный поиск / К. Мэннинг, П. Рагхаван, Х. Шюце; пер. с англ. – Москва : Вильямс, 2011. – 528 с.
6. Момджян К. PostgreSQL. Основы языка SQL / К. Момджян. – Санкт-Петербург : БХВ-Петербург, 2018. – 336 с.
7. Jones M., Bradley J., Sakimura N. RFC 7519: JSON Web Token (JWT) [Электронный ресурс] // IETF : [сайт]. – URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата обращения: 17.03.2026).