УДК 004.421:004.4'22

РАЗРАБОТКА ПЛАТФОРМЫ ДЛЯ АВТОМАТИЗИРОВАННОЙ ПРОВЕРКИ АЛГОРИТМИЧЕСКИХ РЕШЕНИЙ С ПРИМЕНЕНИЕМ КОНТЕЙНЕРИЗАЦИИ И АСИНХРОННОЙ ОБРАБОТКИ ЗАДАЧ

Худницкий А.В., студент

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Жвакина А.В. – канд. техн. наук, доцент

Аннотация. В статье представлено комплексное исследование и практическая реализация распределённой веб-платформы PinkCode для автоматизированной проверки решений алгоритмических задач. Проведён систематический анализ существующих решений (LeetCode, Codewars). Обоснован выбор технологического стека на основе Django REST Framework, Celery и Docker для обеспечения масштабируемости и безопасности системы. Реализована система асинхронной обработки задач на основе паттерна Task Queue с использованием Celery workers и Redis в качестве брокера сообщений. Разработана модульная архитектура бэкенда с разделением на четыре независимых Django-приложения (auth_, main, code_interpreter, users), реализующих принципы Domain-Driven Design и Separation of Concerns. Внедрена многоуровневая система кэширования (Redis, query optimization, database indexing), снижающая нагрузку на PostgreSQL для операций чтения. Система развёрнута с использованием Docker Compose, обеспечивающего воспроизводимость окружения и упрощение процесса деплоя.

Ключевые слова. Автоматизированная проверки решений алгоритмических задач, PinkCode, LeetCode, Codewars, Django.

Введение. Современная индустрия информационных технологий характеризуется растущим спросом на специалистов, владеющих как теоретическими знаниями в области алгоритмов и структур данных, так и практическими навыками их применения.

Традиционные методы обучения программированию, основанные на статических учебных материалах и отложенной проверке решений преподавателем, демонстрируют низкую эффективность в условиях массового образования. Средняя продолжительность проверки одного решения преподавателем составляет 15-20 минут, что при группе из 25 студентов создаёт задержку обратной связи до нескольких дней. Этот факт существенно снижает мотивацию обучающихся и замедляет процесс освоения материала.

Для понимания современного состояния области проведён систематический анализ двух ведущих мировых платформ для решения алгоритмических задач.

LeetCode является наиболее масштабным решением с аудиторией более 15 миллионов пользователей из 120+ стран[1]. База платформы содержит 3700+ задач различной сложности с поддержкой 20+ языков программирования (Python, Java, C++, JavaScript, Go, C#, Ruby, Swift, Kotlin, Rust и др.). Также для платформы характерна полная закрытость исходного кода и стоимость премиум-подписки $39/месяц или $179/год.

Codewars реализует геймифицированный подход через систему рангов кю/дан, заимствованную из восточных единоборств [2]. Платформа насчитывает около 3 миллионов зарегистрированных пользователей и поддерживает 25+ языков программирования.

Уникальной особенностью является community-driven подход к созданию контента: пользователи могут создавать собственные задачи и тест-кейсы, которые проходят модерацию сообществом.

Анализ существующих решений выявил следующие ключевые требования к разработке платформы: безопасность выполнения пользовательского кода, масштабируемость при обработке множества одновременных проверок, быстрая обратная связь и простота расширения функциональности.

Технологический стек и его обоснование. Выбор Python и Django был обоснован следующими факторами:

1 Высокая производительность разработки;

2 Встроенная ORM для работы с БД;

3 Развитая экосистема (Django REST Framework, Celery integration, extensive middleware);

4 Наличие опыта использования в образовательных проектах.

По сравнению с FastAPI, Django обеспечивает лучшую структурированность и полноту решения, несмотря на небольшую потерю производительности в микросекундах [3].

PostgreSQL выбрана как надёжная объектно-реляционная система управления данными с поддержкой сложных запросов, индексов и транзакций. Для часто запрашиваемых данных (лидерборды, списки задач) применено кэширование через Redis, что снижает нагрузку на основную БД и улучшает время ответа на 60-70% [4].

Celery с Redis в качестве брокера сообщений реализует паттерн Task Queue [5]. Этот выбор критичен для системы, так как проверка пользовательского кода может занимать 2-10 секунд, и синхронное выполнение привело бы к блокировке основного потока. Celery позволяет масштабировать

обработку независимо от фронтенда: при увеличении нагрузки добавляются новые workers без изменения кода приложения.

Docker используется для изоляции пользовательского кода [6]. Вместо опасных подходов (ограничение прав пользователя, системные call filters), каждое решение выполняется в отдельном контейнере, полностью отделённом от хост-системы. Это исключает любую возможность escape-атак и повреждения основной инфраструктуры.

Архитектурные паттерны и принципы проектирования. Система реализует Model-View-Serializer (MVS) паттерн, адаптированный Django REST Framework для API-first разработки [7]. Модели содержат бизнес-логику и валидацию, сериализаторы отвечают за преобразование данных между Python-объектами и JSON, представления обрабатывают HTTP-запросы и координируют взаимодействие компонентов.

Применён принцип Separation of Concerns (SoC): система разделена на четыре независимых Django-приложения с чётким распределением ответственности [8]. Модуль auth_ инкапсулирует всю логику аутентификации и авторизации на основе JWT, main содержит сущности задач и тестов, code_interpreter координирует асинхронную проверку, users управляет профилями и рейтингами [9]. Такое разделение позволяет разрабатывать и тестировать каждый модуль независимо.

Реализована трёхуровневая система кэширования: на уровне приложения: использование @cached_property для вычисляемых полей, на уровне БД: индексирование часто используемых колонок (user_id, problem_id, passed), на уровне Redis: кэширование списков задач и лидербордов [10]. Для безопасности взаимодействия клиента и сервера применена JWT-аутентификация с дополнительной проверкой на блэклист (используется django-rest-framework-simplejwt с поддержкой token blacklist). При выходе пользователя токен добавляется в чёрный список, предотвращая его повторное использование.

Система автоматической проверки решений. Критическая часть платформы – безопасное выполнение и проверка пользовательского кода. Процесс состоит из следующих этапов:

1 Пользователь отправляет решение (POST /api/interpreter/submit-code/{problem_id}/);

2 Django-представление создаёт Celery-задачу с передачей кода и тест-кейсов;

3 Celery worker формирует Docker-команду: docker run --rm --network leetcode_network code_interpreter_image python executor.py;

4 Внутри контейнера executor.py загружает код пользователя, вызывает функцию и проверяет вывод против ожидаемого результата;

5 Результаты сохраняются в PostgreSQL через ORM;

6 Фронтенд получает статус через polling.

Dockerfile для executor минимален и содержит только Python 3.10 и необходимые библиотеки, что ограничивает атаковую поверхность. Запрет на импорты в пользовательском коде предотвращает доступ к файловой системе или сетевым ресурсам.

Интеграция компонентов и развёртывание. Docker Compose конфигурация оркеструирует шесть контейнеров: PostgreSQL – основное хранилище данных; — Redis: брокер сообщений и кэш, Django – основное приложение, Nginx – reverse proxy и доставка статики, Celery worker – асинхронная обработка, Celery Beat – планировщик задач.

Две пользовательские сети (leetcode_network и frontend_network) обеспечивают сегментацию трафика: фронтенд имеет доступ только к Nginx/Django, а Celery worker может запускать Docker контейнеры для выполнения кода.

Развёртывание сводится к одной команде: docker-compose up --build, что гарантирует воспроизводимость среды независимо от хост-системы.

Результаты и выводы. Реализованная система демонстрирует следующие характеристики производительности: время ответа API: 100-200 мс (с кэшированием), время проверки одного решения: 1-3 секунды (в зависимости от сложности кода), затраты памяти на одного контейнера: ~50 МБ.

Основные выводы исследования:

1 Контейнеризация критична: использование Docker исключает необходимость в сложных системах изоляции и гарантирует безопасность.

2 Асинхронная обработка масштабируется: Celery + Redis позволяют обрабатывать всплески нагрузки без усложнения основного приложения.

3 Модульная архитектура упрощает поддержку: разделение на Django-приложения облегчает добавление новых функций (система достижений, интеграция с IDE и т.д.).

4 JWT предпочтительнее session-based auth: масштабируемость и независимость от состояния сервера критичны для распределённых систем.

Платформа PinkCode может служить основой для образовательных учреждений и может быть расширена для поддержки: дополнительных языков программирования, системного дизайна и задач на проектирование, интеграции с GitHub Classroom, персонализированных рекомендаций на основе ML.

Список использованных источников:

1. LeetCode Platform Statistics [Электронный ресурс]. – Режим доступа: https://leetcode.com/discuss/general-discussion/

2. Codewars Ranking System Documentation [Электронный ресурс]. – Режим доступа: https://docs.codewars.com/gamification/ranks

3. Web Framework Benchmarks [Электронный ресурс]. – Режим доступа: https://www.techempower.com/benchmarks/

4. PostgreSQL Performance Optimization [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/current/performance-tips.html

5. Celery Distributed Task Queue [Электронный ресурс]. – Режим доступа: https://docs.celeryproject.dev/

6. Docker Best Practices for Security [Электронный ресурс]. – Режим доступа: https://docs.docker.com/develop/security-best-practices/

7. Django REST Framework Documentation [Электронный ресурс]. – Режим доступа: https://www.django-rest-framework.org/

9. JWT Token-Based Authentication [Электронный ресурс]. – Режим доступа: https://tools.ietf.org/html/rfc7519

8. Separation of Concerns in Software Architecture [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/software-engineering/separation-of-concerns-soc

10. Redis as Message Broker [Электронный ресурс]. – Режим доступа: https://redis.io/topics/introduction

UDC 004.8:004.7

DEVELOPMENT OF A PLATFORM FOR AUTOMATED CHECKING OF ALGORITHMIC SOLUTIONS USING CONTAINERIZATION AND ASYNCHRONOUS PROCESSING OF TASKS

Khudnitsky A.V., student

Belarusian State University of Informatics and Radioelectronics, Minsk, Republic of Belarus

Zhvakina A.V. – PhD in Technical Sciences

Annotation. This article presents a comprehensive study and practical implementation of the PinkCode distributed web platform for automated verification of solutions to algorithmic problems. A systematic analysis of existing solutions (LeetCode, Codewars) is conducted. The choice of a technology stack based on the Django REST Framework, Celery, and Docker is justified to ensure system scalability and security. An asynchronous task processing system based on the Task Queue pattern is implemented using Celery workers and Redis as a message broker. A modular backend architecture is developed, divided into four independent Django applications (auth_, main, code_interpreter, users), implementing the principles of Domain-Driven Design and Separation of Concerns. A multi-tier caching system (Redis, query optimization, database indexing) is implemented, reducing the load on PostgreSQL for read operations. The system is deployed using Docker Compose, ensuring a reproducible environment and simplifying the deployment process.

Keywords. Docker, Celery, Django REST Framework, JWT, containerization, process isolation, asynchronous processing, educational platforms, Redis, PostgreSQL, RESTful API.