УДК 004.588

РАЗРАБОТКА ВЕБ-ПЛАТФОРМЫ ДЛЯ ИНТЕРАКТИВНОГО ОБУЧЕНИЯ ПРОГРАММИРОВАНИЮ С АВТОМАТИЧЕСКОЙ ПРОВЕРКОЙ РЕШЕНИЙ В DOCKER-КОНТЕЙНЕРАХ

Литвин Д.Е., студент гр. ИТб-222, IV курс
Научный руководитель: Ванеев О.Н., к.т.н., доцент
Кузбасский государственный технический университет
имени Т.Ф. Горбачёва
г. Кемерово

Когда преподаватель проверяет код студентов вручную, он неизбежно сталкивается с несколькими проблемами. Проверка занимает много времени, оценка зависит от субъективного восприятия, а студент получает обратную связь с задержкой – иногда в несколько дней. На практике это означает, что количество попыток сдачи приходится ограничивать, а преподаватель тратит значительную часть рабочего времени не на объяснение материала, а на рутинный разбор решений.

Платформы вроде LeetCode, HackerRank и Codeforces отчасти решают проблему автоматической проверки, но они ориентированы на соревновательный формат. Создать курс с модулями, привязать задачи к конкретной теме, отследить, как группа справляется с материалом – всё это они не предоставляют. Нужен другой инструмент, спроектированный именно для учебного процесса.

Поэтому была поставлена задача – спроектировать и реализовать веб-платформу для обучения программированию, которая совмещала бы каталог задач, встроенный редактор кода, автоматическую проверку решений и аналитику для преподавателя. Платформа разрабатывается по заказу АНО «РКЦ Кузбасса» (Автономная некоммерческая организация «Региональный координационный центр Кузбасса»). Принципиальное требование – безопасное исполнение пользовательского кода, что достигается запуском решений в изолированных Docker-контейнерах [2].

Роли и функции. В системе выделены две роли: студент и преподаватель. Студент может авторизоваться, просматривать каталог задач, решать задачи во встроенном редакторе кода, отправлять решения на автоматическую проверку, просматривать результаты и рейтинг. Преподаватель, помимо этого, получает панель управления: создание курсов, модулей и задач с тестами, настройка ограничений по времени и памяти, а также доступ к аналитике по курсам и студентам. Диаграмма вариантов использования [5], отражающая распределение функций между актёрами, приведена на рисунке 1.

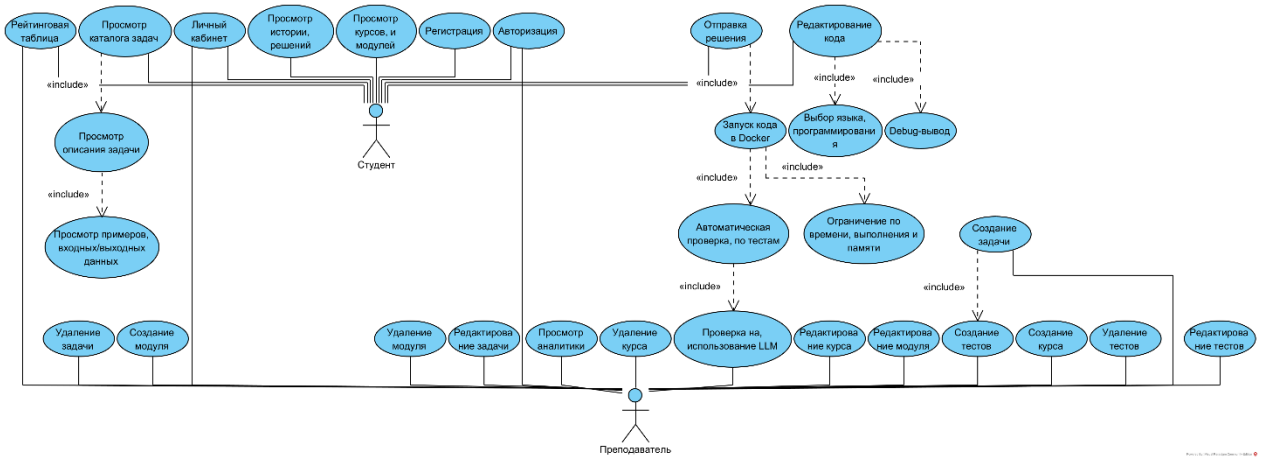

Рисунок 1 – Диаграмма вариантов использования платформы

Архитектура. Платформа построена по клиент-серверной схеме. Сервер написан на Node.js [1] с использованием фреймворка Fastify [7] – он обрабатывает REST API запросы. Данные хранятся в PostgreSQL [4]: эта СУБД выбрана за надёжность, поддержку сложных запросов и открытый исходный код. Клиентская часть – React [3], обеспечивающий реактивный интерфейс с редактором кода, каталогом задач и панелью преподавателя.

Авторизация реализована через JWT [6]. При входе сервер генерирует подписанный токен с идентификатором пользователя и ролью. Клиент передаёт этот токен в заголовке Authorization при каждом запросе, а сервер по нему определяет, кто обращается и какие права у него есть.

Структура данных включает шесть сущностей: Course (курс), Module (модуль), Task (задача), TestCase (тестовый набор), User (пользователь) и Solution (решение). Курс состоит из модулей, модуль – из задач. У каждой задачи есть тестовые наборы: входные данные и ожидаемый вывод. Решение связывает пользователя с задачей и хранит исходный код, язык, статус проверки и время отправки.

Механизм проверки. Когда студент отправляет решение, запускается цепочка: код сохраняется в базу, система подгружает тесты для задачи, поднимает Docker-контейнер с нужным компилятором, прогоняет код по каждому тесту – с контролем времени и памяти – и сравнивает вывод с эталоном. После этого решению присваивается финальный статус. Диаграмма деятельности [5], описывающая этот процесс, приведена на рисунке 2.

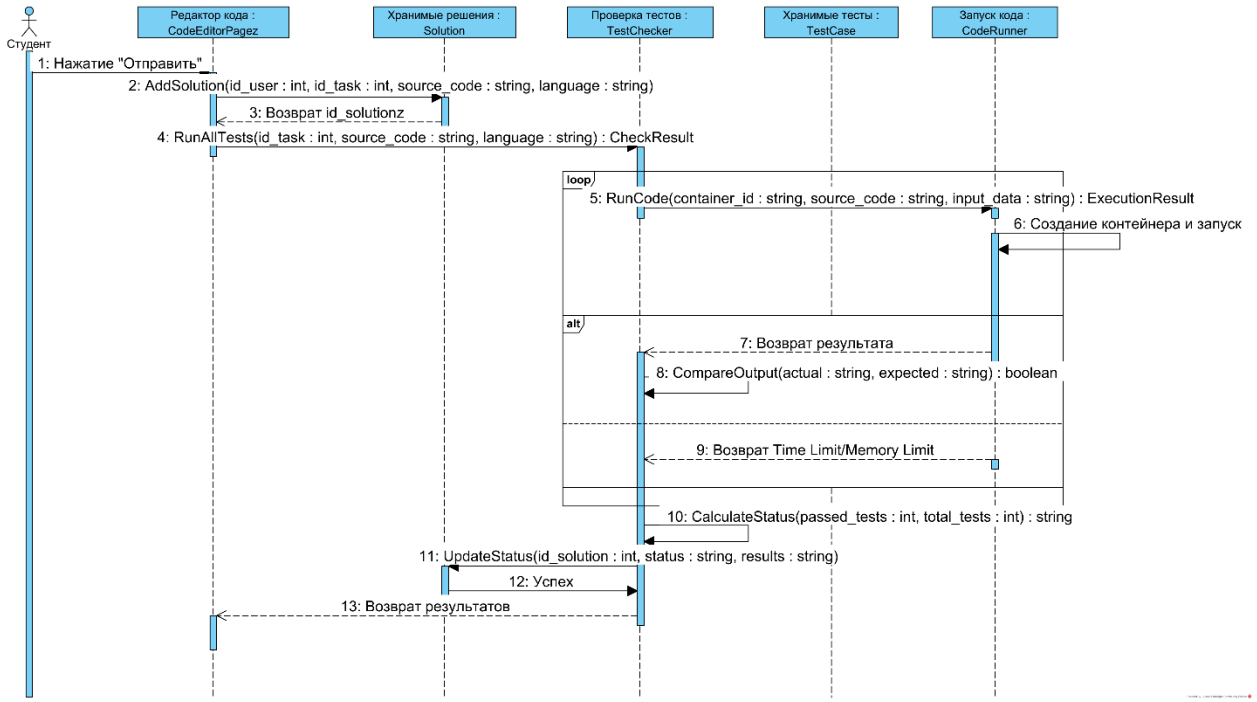

Рисунок 2 – Диаграмма последовательности: отправка решения на проверку

Почему именно Docker? Три причины. Первая – изоляция: код студента работает в отдельном контейнере с урезанными правами, и навредить серверу он не может. Вторая – контроль ресурсов: для каждой задачи можно задать свои лимиты по времени (TimeLimit) и памяти (MemoryLimit), и Docker их строго соблюдает. Третья – мультиязычность: под каждый язык программирования готовится свой Docker-образ с нужным компилятором или интерпретатором, так что добавление нового языка не требует переделки архитектуры.

За техническую часть проверки отвечают два компонента. CodeRunner управляет жизненным циклом контейнера: создаёт его, запускает код, следит за лимитами, удаляет контейнер после завершения. TestChecker последовательно прогоняет тесты – подаёт вход, забирает выход, сравнивает с эталоном. Если лимиты превышены – решение получает статус Time Limit Exceeded или Memory Limit Exceeded. Вывод не совпал – Wrong Answer. Все тесты пройдены – Accepted.

Аналитика. Преподавателю доступна статистика по курсам и модулям – процент выполнения, среднее время решения, распределение статусов. По этим данным можно понять, какие задачи вызывают затруднения, и скорректировать программу.

Развёртывание. Вся система поднимается через Docker Compose – один файл описывает контейнеры сервера, базы данных и клиентского приложения. Это заметно упрощает установку: не нужно вручную настраивать зависимости на каждой машине.

В итоге получилась платформа, которая закрывает основные потребности учебного процесса: автоматическая проверка кода в

изолированных контейнерах, поддержка нескольких языков, удобное управление курсами и задачами, аналитика. Архитектура модульная, что даёт возможность развивать систему дальше – например, добавить механизм обнаружения использования генеративного ИИ при решении задач.

Список литературы:

1. Node.js Documentation [Электронный ресурс]. – URL: https://nodejs.org/en/docs (дата обращения: 15.03.2026).

2. Docker Documentation [Электронный ресурс]. – URL: https://docs.docker.com/get-started/ (дата обращения: 15.03.2026).

3. React Documentation [Электронный ресурс]. – URL: https://react.dev/learn (дата обращения: 15.03.2026).

4. PostgreSQL 16 Documentation [Электронный ресурс]. – URL: https://www.postgresql.org/docs/16/ (дата обращения: 15.03.2026).

5. Буч, Г. Язык UML. Руководство пользователя / Г. Буч, Д. Рамбо, А. Якобсон. – 2-е изд. – Москва : ДМК Пресс, 2006. – 496 с.

6. JSON Web Token (JWT) [Электронный ресурс] : RFC 7519. – URL: https://datatracker.ietf.org/doc/html/rfc7519 (дата обращения: 16.03.2026).

7. Fastify Documentation [Электронный ресурс]. – URL: https://fastify.dev/docs/latest/ (дата обращения: 16.03.2026).