

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

УТВЕРЖДАЮ

Зав.кафедрой,

доцент, к. ф.-м. н.

_____ Л. Б. Тяпаев

ОТЧЕТ О ПРАКТИКЕ

студента 2 курса 221 группы факультета КНиИТ

Мусатова Фёдора Алексеевича

Васильевой Софии Алексеевны

Блохина Артёма Романовича

Беляева Владислава Александрович

вид практики: Учебная

кафедра: дискретной математики и информационных технологий

курс: 2

семестр: 4

продолжительность: с 8.02.2025 г. по 12.06.2025 г.

Руководитель практики от университета,

старший преподаватель

А. А. Трунов

Тема практики: «Разработка веб-приложения для компиляции и проверки тестов на языке программирования C++»

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	5
ВВЕДЕНИЕ	7
1 Теоретические основы разработки веб-приложений	9
1.1 Обзор архитектуры клиент-серверных приложений	9
1.2 Технологии разработки	10
1.2.1 JavaScript	11
1.2.2 Go	11
1.2.3 Docker	11
1.2.4 MinIO	12
1.2.5 Dragonfly	12
1.2.6 React	12
1.2.7 g++	13
1.2.8 REST API	13
1.3 Принципы тестирования программного кода	14
2 Реализация веб-приложения	16
2.1 Структура клиентской части приложения	16
2.2 Структура серверной части приложения	17
2.3 Интеграция приложения	21
ЗАКЛЮЧЕНИЕ	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	25
Приложение А Код, отвечающий за взаимодействие с MinIO	27
Приложение Б Взаимодействие с отправленными решениями	29
Приложение В Описание структуры решений пользователя	35
Приложение Г Описание структуры тестовых значений	36
Приложение Д Точка вхождения программы	37
Приложение Е Использование механизма Pub/Sub в микросервисе	44
Приложение Ж Компиляция решения	46
Приложение З Тестирование программы	50
Приложение И Создание изолированного контейнера для компиляции решений	58
Приложение К Ограничения по компиляции	64
Приложение Л Взаимодействие с песочницей	68
Приложение М Форма на клиентской части приложения	77

Приложение Н	Список решений	81
Приложение О	Вывод статуса проверки задачи	83
Приложение П	Docker-compose, при помощи которого развертываем при- ложение	86

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Amazon S3 — облачное объектное хранилище компании Amazon Web Services для хранения и управления данными.

REST API — программный интерфейс, использующий HTTP-запросы для взаимодействия с сервисами.

Бакеты (buckets) — основной контейнер для хранения данных в сервисе Amazon S3. Бакеты используются для организации и управления объектами (файлами) в облачном хранилище Amazon Web Services.

MinIO — распределённая система хранения объектов с поддержкой S3, обеспечивающая масштабируемое и надёжное хранение данных в облаке и на локальных серверах.

Фронтенд — часть веб-приложения, с которой взаимодействует пользователь: интерфейс, дизайн, анимации.

Бэкенд — серверная часть приложения, которая обрабатывает логику, взаимодействует с базами данных, обеспечивает работу API.

JavaScript — это легковесный интерпретируемый язык программирования с функциями первого класса. Наиболее широкое применение находит как язык сценариев веб-страниц.

Go (или Golang) — это компилируемый многопоточный язык программирования, разработанный внутри компании Google. Использует объектно-ориентированный (структурный) стиль с поддержкой функциональных элементов.

Props — это данные, передаваемые от родительского компонента к дочернему для настройки его поведения или внешнего вида.

Эндпоинты — URL-адрес, по которому клиент может получить доступ к функционалу сервера.

Docker — платформа, позволяющая упаковывать приложения и их зависимости в контейнеры для простоты развертывания и запуска в любой среде.

Dockerfile — это конфигурационный файл, в котором описаны инструкции, которые будут применены при сборке Docker-образа и запуске контейнера.

Docker Compose — это инструмент, который упрощает развертывание и управление многоконтейнерными приложениями Docker. Он позволяет

определить и запустить несколько Docker-контейнеров, взаимодействующих друг с другом, используя один файл конфигурации

Контейнеры — изолированные и легковесные среды выполнения приложений, содержащие всё необходимое для их работы: код, библиотеки, зависимости.

Хуки — механизм в React, позволяющий использовать состояние и другие возможности без написания классов.

Буферизация данных — процесс временного хранения данных, чтобы ускорить их последующее получение.

Горутины — легковесные потоки, управляемые средой выполнения Go. Горутины позволяют выполнять функции асинхронно, что делает параллелизм в Go очень эффективным и простым.

Pub/Sub — архитектурный паттерн, при котором отправитель (publisher) не отправляет сообщения напрямую получателю (subscriber), а рассылает их всем подписчикам на определённую тему.

Продакшн — финальная стадия разработки, где приложение работает в реальной среде и используется конечными пользователями.

SPA — веб-приложение, которое загружается один раз, после чего динамически обновляет содержимое без перезагрузки страницы

API — Application Programming Interface.

HTTP — HyperText Transfer Protocol.

Tailwind CSS — CSS-фреймворк с открытым исходным кодом, позволяющий вносить изменения в оформление сайтов и приложений, не покидая HTML-разметку.

React — это JavaScript-библиотека, предназначенная для создания пользовательских интерфейсов. Она позволяет разработчикам строить интерактивные элементы веб-страниц и приложений, такие как кнопки, виджеты, чаты и многое другое.

ВВЕДЕНИЕ

Современные информационные технологии стремительно развиваются, предоставляя новые возможности для автоматизации процессов, связанных с разработкой программного обеспечения. Одной из актуальных задач в этой области является создание систем, позволяющих компилировать код и проверять его корректность с помощью автоматического тестирования. Такие системы находят широкое применение в образовательных учреждениях, где они помогают студентам осваивать языки программирования, а также в профессиональной среде, где разработчики используют их для быстрого анализа и отладки программ. Существуют многочисленные платформы, такие как Stepik, LeetCode и Codeforces, которые предоставляют пользователям возможность компилировать код онлайн и выполнять тесты, однако они часто ориентированы на определённые языки программирования или имеют ограничения в гибкости настройки. В этом контексте разработка веб-приложения, способного компилировать код на различных языках и обеспечивать проверку тестов с использованием современных технологий, становится важным шагом для удовлетворения потребностей пользователей, стремящихся к удобству и универсальности. Использование современных технологий, таких как клиент-серверная архитектура, контейнеризация и распределённое хранение данных, открывает новые перспективы для создания масштабируемых и производительных приложений.

Актуальность работы заключается в комплексном подходе к разработке: приложение объединяет клиентскую часть на React, обеспечивающий интуитивно понятный интерфейс, и серверная часть на Go, использующий компилятор g++ для обработки кода, а также интегрирует Docker для изоляции процессов, MinIO для хранения данных и Dragonfly для кэширования. Такой подход позволяет не только обеспечить высокую производительность, но и заложить основу для дальнейшего расширения функциональности, включая поддержку дополнительных компиляторов и улучшение интерфейса.

Цель работы — это разработать веб-приложение для компиляции кода и проверки тестов.

Задачи, которые необходимы для разработки нашего веб-приложения:

- Реализовать клиентскую часть приложения с использованием компонентов React для ввода кода и отображения результатов

- Разработать серверную часть приложения на Go для обработки запросов, компиляции и тестирования
- Интегрировать клиентскую и серверную части через REST API
- Устранить проблему несоответствия полей при отображении результатов тестов

1 Теоретические основы разработки веб-приложений

1.1 Обзор архитектуры клиент-серверных приложений

Архитектура клиент-сервер представляет собой фундаментальную модель разработки распределённых систем, широко применяемую в современных веб-приложениях, включая систему компиляции кода и проверки тестов [1]. Данная архитектура предполагает разделение функциональности между двумя основными компонентами: клиентским и серверным. Клиентская часть отвечает за предоставление пользовательского интерфейса и обработку взаимодействий с пользователем, тогда как серверная часть выполняет задачи обработки данных, хранения информации и выполнения вычислений. Такое разделение обеспечивает эффективное распределение нагрузки и упрощает масштабирование системы.

Основной принцип архитектуры заключается в том, что клиент инициирует запросы к серверу, которые передаются через сетевые протоколы, такие как HTTP или HTTPS. Сервер, располагающийся на удалённом узле с достаточными вычислительными ресурсами, принимает запросы, выполняет необходимые операции и возвращает результаты клиенту. В контексте рассматриваемого проекта клиент отправляет исходный код на языке C++ через интерфейс, сервер компилирует его с использованием g++, проводит тестирование и возвращает результаты в формате JSON. Этот процесс демонстрирует чёткое распределение функций, где клиент фокусируется на визуализации, а сервер — на вычислениях.

Архитектура обладает рядом преимуществ. Централизованное управление данными на сервере облегчает администрирование и резервное копирование. Независимость разработки клиентской и серверной частей позволяет обновлять серверную логику без изменения клиентского интерфейса, что повышает гибкость системы. Кроме того, поддержка различных клиентских платформ, включая настольные компьютеры и мобильные устройства, расширяет область применения модели. Однако архитектура имеет и ограничения. Зависимость от сетевого соединения может привести к недоступности системы при его отсутствии. При значительном увеличении числа запросов требуется внедрение механизмов балансировки нагрузки. Также возрастает необходимость обеспечения безопасности передачи данных между клиентом и сервером.

Примером реализации данной архитектуры в проекте является взаимодействие фронтенда, разработанного с использованием React, и бэкенда, реализованного на языке Go. Клиентский интерфейс предоставляет пользователю возможность ввода кода, а сервер обрабатывает запросы, выполняя компиляцию и тестирование. Такая организация структуры обеспечивает надёжность и готовность системы к расширению в зависимости от пользовательской нагрузки.

Архитектура клиент-сервер является ключевым подходом в разработке веб-приложений, обеспечивая эффективное разделение функций между клиентом и сервером. Её применение в проекте компиляции кода демонстрирует преимущества в гибкости и масштабируемости, несмотря на вызовы, связанные с сетевой зависимостью и необходимостью обеспечения безопасности. Данная модель создаёт прочную основу для дальнейшего развития системы.

1.2 Технологии разработки

Для создания веб-приложения, которое компилирует код и проверяет тесты, мы использовали несколько современных технологий. Каждая технология отвечает за свою часть работы: от красивого интерфейса до обработки кода и хранения данных.

В связи с большим количеством решений в сфере разработки веб-приложений, мы выбрали следующие технологии: JavaScript, Go, Docker, MinIO, Dragonfly. В качестве языка для фронтенда можно было выбрать TypeScript или Dart, но мы использовали JavaScript из-за его простоты, широкой распространённости и знания нашей командой. Для бэкенда рассматривались Rust или Python, но Go был выбран благодаря его высокой производительности, лаконичному синтаксису и встроенной поддержке многопоточности, быстрому написанию веб-приложений и хорошему знанию данного языка командой. Вместо Docker можно было использовать Podman или LXC, но Docker остаётся самым популярным инструментом для контейнеризации с удобным управлением и богатой экосистемой. Для объектного хранилища подходили Seph или Amazon S3, но MinIO был взят из-за его лёгкости развёртывания, S3-совместимости, открытой лицензии, отсутствия дополнительных затрат на аренду сервера. В качестве кэш-сервера могли быть Redis или Memcached, но Dragonfly был выбран из-за его высокой скорости, эффективного исполь-

зования памяти и совместимости с Redis API, а также большей совместимостью с нашим проектом. Таким образом, каждая технология в стеке сочетает простоту, широкое знание среди команды разработчиков, популярность и оптимальную производительность для быстрой и масштабируемой разработки. Ниже мы подробно объясняем, что это за технологии, как они работают и зачем нужны в проекте.

1.2.1 JavaScript

JavaScript — это язык программирования, который делает веб-страницы живыми и интерактивными. С его помощью можно добавлять кнопки, формы, всплывающие окна и обновлять данные без перезагрузки страницы. Например, когда пользователь нажимает кнопку "Отправить код JavaScript быстро отправляет код на сервер и показывает результат. В проекте JavaScript отвечает за всю интерактивность: он обрабатывает действия пользователя, такие как ввод кода или выбор задачи, и отправляет данные на сервер через технологию AJAX. AJAX позволяет общаться с сервером в фоновом режиме, чтобы всё работало быстро и без задержек. Без JavaScript приложение было бы статичным и неудобным, так как пользователю пришлось бы каждый раз перезагружать страницу.

1.2.2 Go

Go (или Golang) — это язык программирования от Google, который простой, быстрый и удобный для создания серверов. Он умеет одновременно обрабатывать много запросов от пользователей благодаря горутинам — это как лёгкие потоки, которые работают параллельно и не тормозят сервер [2]. В проекте Go отвечает за серверную часть: он принимает код от пользователя, компилирует его, запускает тесты и отправляет результаты обратно. Например, когда студент отправляет код на C++, сервер на Go запускает компилятор и проверяет, всё ли работает. Go выбрали, потому что он быстрый и надёжный, даже если много людей используют приложение одновременно.

1.2.3 Docker

Docker — это инструмент, который упаковывает приложение и всё, что ему нужно для работы, в специальные "контейнеры". Контейнер — изолированная среда, где лежат сервер, компилятор и настройки, для быстрого

развертывания и одинаковой работы на любом компьютере. Это удобно, потому что не нужно тратить время на настройку окружения [3]. В проекте Docker создаёт контейнеры для компиляции кода и запуска тестов. Например, в одном контейнере работает сервер на Go и компилятор g++. Контейнеры также делают приложение безопасным: код пользователя запускается в изолированной среде и не может навредить серверу. Ещё Docker упрощает запуск приложения в облаке, например, на серверах Amazon или Google.

1.2.4 MinIO

MinIO — это программа для локального хранения данных, похожая на облако, например, Amazon S3. Она сохраняет файлы, такие как код, тесты или результаты, в виде объектов, которые легко найти и использовать. MinIO удобна, потому что может хранить много данных и масштабируется: если пользователей станет больше, она справится [4]. В проекте MinIO сохраняет код, который отправляют пользователи, тесты для проверки и результаты выполнения. Например, если студент хочет посмотреть свой старый код или результаты тестов, MinIO быстро найдёт эти данные. Это также помогает сохранять историю решений, чтобы можно было анализировать прогресс.

1.2.5 Dragonfly

Dragonfly — это система кэширования, работающая в оперативной памяти, похожая на Redis полностью совместимым API, но быстрее и проще в использовании. Кэш нужен, чтобы хранить данные, которые часто запрашиваются, например, результаты тестов. Вместо того чтобы каждый раз заново компилировать код, сервер берёт готовый результат из Dragonfly. Это ускоряет работу приложения и снижает нагрузку на сервер [5]. В проекте Dragonfly сохраняет результаты компиляции и тестов, чтобы, например, не проверять один и тот же код много раз. Это особенно важно, когда много пользователей отправляют похожие запросы, и приложение должно работать быстро.

1.2.6 React

React — это библиотека, созданная Facebook, которая помогает строить удобные и быстрые интерфейсы. Она разбивает интерфейс на маленькие кусочки — компоненты, которые можно использовать повторно, как конструктор. Например, один компонент отвечает за форму ввода кода, другой — за

список задач, а третий — за показ результатов. React использует виртуальный DOM (модель страницы), чтобы обновлять только те части интерфейса, которые изменились, а не всю страницу [6]. Это делает приложение быстрым, даже если на странице много данных. В проекте мы создали несколько компонентов:

1. TaskForm.jsx — форма, где пользователь вводит код и выбирает задачу;
2. TaskList.jsx — список задач, которые можно выбрать;
3. TaskStatus.jsx — показывает, прошёл ли код тесты или где ошибка.

Для оформления мы использовали Tailwind CSS — это набор готовых стилей, которые делают интерфейс красивым и удобным на любом устройстве: компьютере, планшете или телефоне. Tailwind CSS помогает быстро настроить цвета, размеры и расположение элементов, чтобы всё выглядело аккуратно.

1.2.7 g++

g++ — это компилятор из набора GNU Compiler Collection (GCC), который превращает код на C++ в программу, которую можно запустить. Он проверяет код на ошибки и создаёт исполняемый файл. g++ хорош тем, что поддерживает разные версии C++ и выдаёт понятные сообщения об ошибках, что помогает студентам исправлять код. В проекте g++ используется для компиляции кода на C++, который отправляет пользователь. Например, если студент решает задачу, g++ проверяет, правильно ли написан код, и создаёт программу, которую сервер тестирует. Если есть ошибка, g++ сообщает, где она, чтобы пользователь мог её исправить.

1.2.8 REST API

REST API — это способ, которым интерфейс (React) и сервер (Go) общаются друг с другом. Пользователь через браузер отправляет код и тесты на сервер с помощью HTTP-запросов, а сервер отвечает в формате JSON (это удобный способ передачи данных). Например, когда пользователь нажимает "Проверить код React" отправляет код на сервер, Go компилирует его в Docker-контейнере с g++, сохраняет данные в MinIO, кэширует результат в Dragonfly и отправляет ответ обратно в браузер. REST API делает приложение чётким: интерфейс отвечает за внешний вид, а сервер — за обработку. Это также позволяет легко добавлять новые функции, не меняя всю систему.

1.3 Принципы тестирования программного кода

Тестирование программного кода представляет собой критически важный этап разработки, направленный на проверку соответствия программного обеспечения заданным требованиям и выявление ошибок. В контексте веб-приложения для компиляции кода и проверки тестов тестирование приобретает особую значимость, обеспечивая надежность системы при обработке пользовательского ввода и выполнении вычислительных задач. Основной целью тестирования является подтверждение корректности функциональности, производительности и безопасности приложения, что достигается через систематический анализ поведения программы в различных условиях.

Одним из ключевых принципов тестирования является охват всех возможных сценариев использования. Это включает проверку компиляции корректного кода, обработку синтаксических ошибок, а также тестирование граничных случаев, таких как превышение лимитов памяти или времени выполнения. В рассматриваемом проекте сервер, реализованный на Go, использует компилятор `g++` для преобразования кода на C++ и последующего запуска тестов, результаты которых возвращаются клиенту в формате JSON. Для обеспечения полноты тестирования применяются как ручные, так и автоматизированные подходы, включая модульное тестирование компонентов фроненда на React и интеграционное тестирование взаимодействия клиентской и серверной частей.

Еще одним важным принципом является воспроизводимость результатов. Тестирование должно проводиться в контролируемой среде, что в проекте достигается с использованием Docker для изоляции контейнеров, содержащих сервер и компилятор. Это позволяет устранить влияние внешних факторов и гарантировать стабильность результатов при повторных запусках. Кроме того, тестирование должно учитывать производительность системы, включая время ответа сервера и нагрузку при одновременной обработке множества запросов, что особенно актуально для масштабируемых приложений с кэшированием через Dragonfly и хранением данных в MinIO.

Принцип раннего обнаружения ошибок подчеркивает необходимость проведения тестирования на всех этапах разработки. В проекте это реализовано через поэтапную проверку: от тестирования отдельных компонентов, таких как `TaskStatus.JSx`, до финальной проверки всей системы. Такой под-

ход минимизирует затраты на исправление дефектов и повышает качество продукта. Также важна независимость тестов, что исключает взаимное влияние между проверяемыми модулями, обеспечивая точность оценки каждого элемента системы. Наконец, тестирование должно учитывать требования безопасности, включая защиту от уязвимостей, таких как инъекции кода или превышение ресурсов. В данном случае серверная логика на Go включает валидацию входных данных перед компиляцией, что снижает риски эксплуатации. Совокупность этих принципов формирует основу для создания надежного и устойчивого программного обеспечения, способного эффективно функционировать в реальных условиях.

Принципы тестирования программного кода, такие как охват сценариев, воспроизводимость, раннее обнаружение ошибок, независимость тестов и безопасность, обеспечивают высокую надежность веб-приложения для компиляции и проверки тестов. Их реализация с использованием современных технологий, включая Docker, Go и g++, подтверждает соответствие системы заданным требованиям и создает основу для ее дальнейшего совершенствования.

2 Реализация веб-приложения

2.1 Структура клиентской части приложения

Структура клиентской части веб-приложения для компиляции кода и проверки тестов разработана с использованием библиотеки React, что обеспечивает модульность и эффективность обновления пользовательского интерфейса [6]. Фронтенд представляет собой совокупность компонентов, взаимодействующих между собой для реализации основных функций системы. Основным файлом является App.JSx, который служит корневым компонентом и определяет общую структуру приложения, включая маршрутизацию и интеграцию с другими модулями. Дополняет его Main.JSx, обеспечивающий начальную инициализацию и рендеринг ключевых элементов интерфейса. Для управления вводом данных используется компонент TaskForm.JSx, который позволяет пользователю вводить исходный код и параметры тестов, отправляя их на сервер через REST API. Компонент TaskList.JSx отвечает за отображение списка задач, предоставляя пользователю обзор всех отправленных запросов с возможностью их идентификации по уникальным идентификаторам. Наконец, компонент TaskStatus.JSx отображает результаты выполнения задач, включая состояние компиляции и итоги тестирования, такие как количество пройденных тестов, основываясь на данных, полученных в формате JSON.

Для стилизации интерфейса применяется Tailwind CSS — CSS фреймворк для упрощенного создания интерфейса, что обеспечивает адаптивный и современный дизайн, совместимый с различными устройствами, что представлено на рисунке 1.

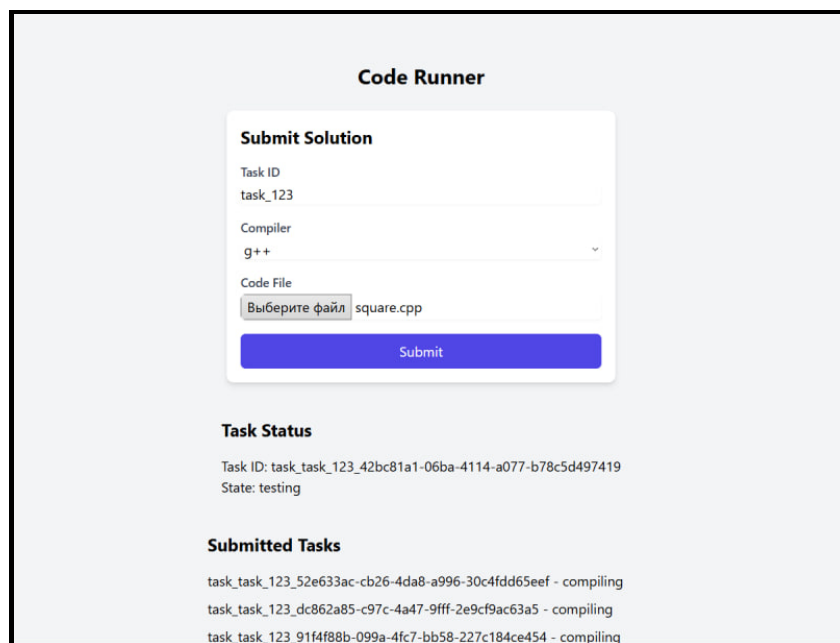


Рисунок 1 – Внешний вид веб-приложения

Компоненты связаны через пропсы и состояние, управляемое с помощью хуков React, таких как `useState` и `useEffect`, что обеспечивает динамическое обновление данных при получении ответов от сервера. Структура фронтенда оптимизирована для поддержки масштабируемости, позволяя легко добавлять новые функции, такие как визуализация результатов тестов или улучшение формы ввода. Такая организация компонентов обеспечивает чёткое разделение ответственности и упрощает процесс разработки и сопровождения системы.

2.2 Структура серверной части приложения

Серверная часть веб-приложения для компиляции кода и проверки тестов реализована на языке программирования Go, что обеспечивает высокую производительность и поддержку конкурентных операций. Она распределяется на серверную часть, позволяющую произвести валидацию и обработать данные с требованиями микросервиса и передать их на компиляцию и микросервис, компилирующий код и возвращающий данные обратно. Основная задача серверной части заключается в обработке запросов от клиента, компиляции исходного кода, выполнения тестов и возврате результатов в формате JSON. Сервер принимает запросы через REST API, используя эндпоинт `/api/tasks`, куда клиент отправляет исходный код и параметры тестов. Пример данных в формате JSON, отправляемых с клиента на наш микросервис

показан на листинге 1.

Листинг 1 – Пример данных перед обработкой микросервиса

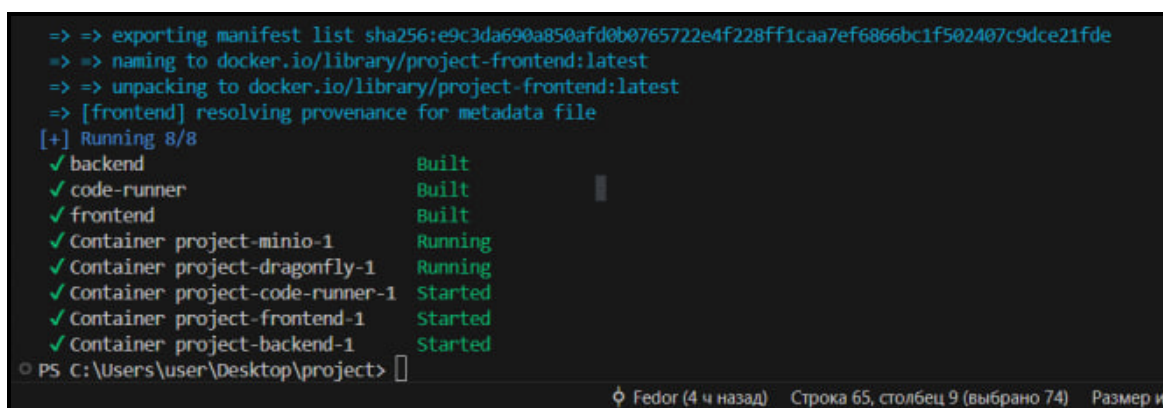
```
{
  "id": "task_task_123_2ce16d93-c0fc-4143-bede-
    b3667b871187",
  "codeLocation": {
    "bucketName": "code",
    "objectName": "2ce16d93-c0fc-4143-bede-
      b3667b871187.cpp"
  },
  "testsLocation": {
    "bucketName": "tests",
    "objectName": "task_123.JSON"
  },
  "executableLocation": {
    "bucketName": "",
    "objectName": ""
  },
  "compiler": "g++",
  "state": "compiling",
  "testResults": null
},
```

После получения запроса программа извлекает данные, выполняет валидацию входных параметров и передаёт текст кода в бакет MinIO под названием code. Параллельно он посылает ID задачи — уникальный идентификатор проверяемого решения, CodeLocation — путь к .cpp файлу, хранимого в MinIO, TestLocation — путь к JSON файлу, хранимого в MinIO, Compiler — строка с названием необходимого для компиляции кода в брокер сообщений Dragonfly. Далее наш микросервис забирает данные из брокера сообщений и плоского хранилища на компиляцию с использованием компилятора g++, который преобразует код на C++ в исполняемый файл .exe. В случае успешной компиляции сервер запускает тесты, сравнивая выходные данные программы с ожидаемыми результатами, хранимыми в JSON, и формирует ответ, содержащий статус задачи и результаты тестов, как например на листинге 2.

Листинг 2 – Пример данных после обработки микросервисом

```
[{task_task_123_5bb91db7-d544-4012-94f6-497
  b7058c62f 1 true} {task_task_123_5bb91db7-d544
-4012-94f6-497b7058c62f 2 true} {
  task_task_123_5bb91db7-d544-4012-94f6-497
  b7058c62f 0 true}]
```

Для изоляции процессов компиляции и выполнения тестов используется Docker, который создаёт контейнеры с необходимым окружением [3], включая g++ и временные файлы. Так же Docker играет немаловажную роль в процессе развертывания всего приложения. Был описан Dockerfile для каждой из частей нашего приложения — клиентской и серверной части приложения, а также микросервиса. Так же, для большей автоматизации, мы описали Docker-compose.yaml. Он позволяет развертывать наше приложение одной командой. Ниже можно наблюдать процесс развертывания докер-контейнеров, при помощи инструкции, написанной в Docker-Compose.yaml.



```
=> => exporting manifest list sha256:e9c3da690a850afd0b0765722e4f228ff1caa7ef6866bc1f502407c9dce21fde
=> => naming to docker.io/library/project-frontend:latest
=> => unpacking to docker.io/library/project-frontend:latest
=> [frontend] resolving provenance for metadata file
[+] Running 8/8
✓ backend                               Built
✓ code-runner                           Built
✓ frontend                               Built
✓ Container project-minio-1             Running
✓ Container project-dragonfly-1         Running
✓ Container project-code-runner-1       Started
✓ Container project-frontend-1          Started
✓ Container project-backend-1           Started
PS C:\Users\user\Desktop\project>
```

Рисунок 2 – Развертывание приложение при помощи Docker

Листинг 3 – Часть кода из Docker-compose.yaml

```
services:
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    ports:
      - "8080:8080"
    environment:
      - SERVER_PORT=8080
      - MINIO_ENDPOINT=minio:9000
      - MINIO_ACCESS_KEY=minioadmin
      - MINIO_SECRET_KEY=minioadmin
      - MINIO_USE_SSL=false
      - DRAGONFLY_HOST=dragonfly
      - DRAGONFLY_PORT=6379
      - DRAGONFLY_PASSWORD=
    depends_on:
      - minio
      - dragonfly
    networks:
      - app-network
    restart: unless-stopped
```

Данные, такие как исходный код, тесты и результаты, сохраняются в объектное хранилище MinIO, обеспечивающее долговременное хранение и доступ через API, совместимое с S3.

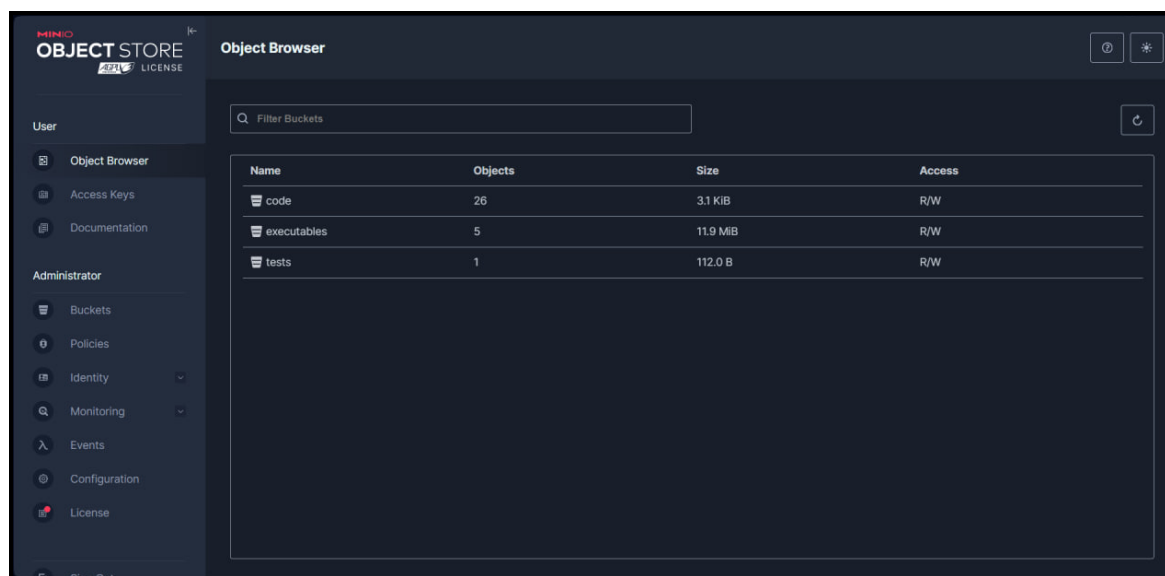


Рисунок 3 – Бакеты в MINIO

Для оптимизации производительности серверная часть приложения применяет кэширование данных с помощью Dragonfly, что позволяет сохранять результаты часто выполняемых задач и снижать нагрузку на сервер при повторных запросах. Логирование операций, включая ошибки компиляции и выполнения, реализовано для диагностики и отладки, что упрощает сопровождение системы. Структура бэкенда организована модульно, с разделением логики обработки запросов, компиляции и тестирования, что обеспечивает гибкость и возможность дальнейшего расширения функциональности [5]. Также Dragonfly используется в качестве брокера сообщений для общения между серверной частью приложения и микросервисом при помощи механизма Pub/Sub.



Рисунок 4 – Использование Dragonfly

2.3 Интеграция приложения

Интеграция веб-приложения для компиляции кода и проверки тестов осуществляется через протокол REST API, обеспечивая взаимодействие между клиентской частью, реализованной на React с использованием JavaScript, и серверной частью, разработанной на языке Go. Основной точкой взаимодействия является эндпоинт `/api/tasks`, через который фронтенд отправляет POST-запросы, содержащие исходный код и параметры тестов, в формате JSON. Бэкенд принимает запросы, выполняет валидацию данных и присваивание уникального идентификатора файлу решения пользователя, после чего передаёт код на компиляцию при помощи Dragonfly и MinIO с использованием `g++` в изолированном Docker-контейнере, что гарантирует безопасность и стабильность выполнения, благодаря валидации файла и изолируемости контейнера. После компиляции сервер запускает тесты, сравнивая результа-

ты с ожидаемыми значениями, и формирует ответ, который включает идентификатор задачи, статус выполнения и результаты тестов. На листинге 3, в упрощённой форме, показан пример обработки тестовых значений задачи, взятых из бакета и получаемый в результате микросервиса результат. Все данные логируются для дальнейшего вывода в консоль.

Листинг 4 – Пример работы тестов

```
test #0: Sandbox started

test #2: Output read from sandbox

test #2:      6

test #2: Testing completed with exit code 0

test #2: Test passed

test #0: Output read from sandbox

test #0:      3

test #0: Testing completed with exit code 0

test #0: Test passed

test #1: Sandbox removed

test #2: Sandbox removed

test #0: Sandbox removed
```

Листинг 5 – Пример итогового ответа

```
Done! Closing testResultsCh and testsCh

All tests completed!

[{"task_task_123_5bb91db7-d544-4012-94f6-497b7058c62f": 1, "true": true}, {"task_task_123_5bb91db7-d544-4012-94f6-497b7058c62f": 2, "true": true}, {"task_task_123_5bb91db7-d544-4012-94f6-497b7058c62f": 0, "true": true}]
```

Этот ответ возвращается клиенту в формате JSON, где данный компонент `TaskStatus.JSx` на фронтенде обрабатывает данные, полученные после заполнения пользователем формы и отображает результаты пользователю, включая количество пройденных тестов и статус тестирования. Для управления асинхронными запросами фронтенд использует JavaScript-метод `fetch` в сочетании с хуками `React`, такими как `useEffect`, что позволяет динамически обновлять интерфейс при получении ответа от сервера. Данные, такие как исходный код, тесты и результаты, сохраняются в объектное хранилище `MinIO`, а кэширование ответов и взаимодействие между серверной частью приложения и микросервисам осуществляется через брокер сообщений `Dragonfly`, что снижает нагрузку на сервер при повторных запросах. Такая организация интеграции обеспечивает надёжное взаимодействие между клиентской и серверной частями, поддерживая масштабируемость и гибкость системы.

ЗАКЛЮЧЕНИЕ

В результате работы нами было создано веб-приложение GoRunner, которое позволяет компилировать код и проверять его с помощью автоматических тестов. Предполагается, что этот инструмент будет использоваться в учебных целях, где важна проверка результата для понимания усвоения учебного материала. GoRunner помогает получать обратную связь, что особенно важно в процессе обучения.

Выполнение задач, поставленных в начале работы, позволили нам реализовать наше веб-приложение. Была выполнена правильная организация взаимодействия между клиентом и сервером. На основе полученных знаний выполнена разработка интерфейса на React, который позволяет пользователям удобно работать с приложением, например: вводить код в специальной форме, выбирать задачи из списка и видеть результат тестов. Серверная часть на языке программирования Golang обрабатывает запросы, компилирует код и запускает тесты, что обеспечивает быструю и надёжную работу. Разделение кода на модули, слои сервисов и репозиториев обеспечило удобство сопровождения, повторного использования компонентов и возможности параллельной работы нескольких команд разработки. Микросервис Coderunner, отвечающий исключительно за компиляцию и тестирование C++ кода, обрабатывает каждый запрос независимо, без использования внутреннего состояния, что упрощает горизонтальное масштабирование и повышает отказоустойчивость.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Bass, L. Software Architecture in Practice / L. Bass, P. Clements, R. Kazman. - 3rd Edition. - Addison-Wesley, 2012. - 624 с.
- 2 Go Programming Language Documentation [Электронный ресурс] // The Go Project. - URL: <https://golang.org/doc/> (дата обращения: 20.04.2025) - Загл. с экрана - Яз. англ.
- 3 Docker Documentation [Электронный ресурс] // Docker. - URL: <https://docs.docker.com/> (дата обращения: 20.04.2025) - Загл. с экрана - Яз. англ.
- 4 MinIO Documentation [Электронный ресурс] // MinIO. - URL: <https://docs.min.io/> (дата обращения: 20.04.2025) - Загл. с экрана - Яз. англ.
- 5 Dragonfly Documentation [Электронный ресурс] // Dragonfly. - URL: <https://www.dragonflydb.io/docs> (дата обращения: 20.04.2025) - Загл. с экрана - Яз. англ.
- 6 React Documentation [Электронный ресурс] // React. - URL: <https://reactjs.org/docs/> (дата обращения: 20.04.2025) - Загл. с экрана - Яз. англ.
- 7 Алексеев, П.О. ОПТИМИЗАЦИЯ ПРОИЗВОДИТЕЛЬНОСТИ ВЕБ-ПРИЛОЖЕНИЙ С ИСПОЛЬЗОВАНИЕМ СОВРЕМЕННЫХ ФРЕЙМВОРКОВ // Universum: технические науки : электрон. науч. журн. 2025. 4(133). URL: <https://7universum.com/ru/tech/archive/item/19668>
- 8 Amazon Simple Storage Service (S3) [Электронный ресурс] // Amazon Web Services. - URL: <https://aws.amazon.com/ru/s3/> (дата обращения: 20.04.2025) - Загл. с экрана - Яз. англ.
- 9 Newman, S. Building Microservices / S. Newman - 2nd Edition. - O’Rielli, 2022. - 615 с.
- 10 Richardson, C. Microservices Patterns / C. Richardson - Manning, 2019. - 522 с.

Отчет по практике "Разработка веб-приложения для компиляции и проверки тестов на языке программирования C++" выполнена нами самостоятельно, и на все источники, имеющиеся в работе, даны соответствующие ссылки.

подпись, дата

инициалы, фамилия

ПРИЛОЖЕНИЕ А

Код, отвечающий за взаимодействие с MinIO

```
package filesctl

import (
    "bytes"
    "context"
    "io"

    "github.com/minio/minio-go/v7"
)

type MinioManager struct {
    client *minio.Client
}

func NewMinioManager(client *minio.Client) *MinioManager {
    return &MinioManager{client: client}
}

func (m *MinioManager) PutFile(ctx context.Context, bucket
    string, name string, data []byte) error {
    _, err := m.client.PutObject(ctx, bucket, name,
        bytes.NewReader(data), int64(len(data)), minio.
        PutObjectOptions{})
    return err
}

func (m *MinioManager) LoadFile(ctx context.Context,
    bucket string, name string) ([]byte, error) {
    object, err := m.client.GetObject(ctx, bucket,
        name, minio.GetObjectOptions{})
    if err != nil {
        return nil, err
    }
}
```

```
}
defer object.Close()

data, err := io.ReadAll(object)
if err != nil {
    return nil, err
}

return data, nil
}
```

ПРИЛОЖЕНИЕ Б

Взаимодействие с отправленными решениями

```
package handler

import (
    "encoding/json"
    "fmt"
    "io"
    "net/http"

    "github.com/Feedonya/Runner/backend/filesctl"
    "github.com/Feedonya/Runner/backend/model"
    "github.com/google/uuid"
    "github.com/gorilla/mux"
    "github.com/redis/go-redis/v9"
)

type TaskHandler struct {
    filesManager filesctl.Manager
    redisClient  *redis.Client
}

func NewTaskHandler(filesManager filesctl.Manager,
    redisClient *redis.Client) *TaskHandler {
    return &TaskHandler{
        filesManager: filesManager,
        redisClient:  redisClient,
    }
}

func (h *TaskHandler) CreateTask(w http.ResponseWriter, r
    *http.Request) {
    ctx := r.Context()
```

```

err := r.ParseMultipartForm(10 << 20) // 10 MB
    limit
if err != nil {
    http.Error(w, "Failed to parse form", http.
        .StatusBadRequest)
    return
}

file, _, err := r.FormFile("code")
if err != nil {
    http.Error(w, "Missing code file", http.
        .StatusBadRequest)
    return
}
defer file.Close()

codeData, err := io.ReadAll(file)
if err != nil {
    http.Error(w, "Failed to read code file",
        http.StatusBadRequest)
    return
}

taskID := r.FormValue("task_id")
if taskID == "" {
    http.Error(w, "Missing task_id", http.
        .StatusBadRequest)
    return
}

compiler := r.FormValue("compiler")
if compiler == "" {
    compiler = "g++"
}

```

```

attemptID := uuid.New().String()
codeObjectName := fmt.Sprintf("%s.cpp", attemptID)

// Upload code to MinIO
err = h.filesManager.PutFile(ctx, "code",
    codeObjectName, codeData)
if err != nil {
    http.Error(w, "Failed to upload code to
        MinIO", http.StatusInternalServerError)
    return
}

taskCommand := model.StartTaskCommand{
    ID: fmt.Sprintf("task_%s_%s", taskID,
        attemptID),
    CodeLocation: model.FileLocation{
        BucketName: "code",
        ObjectName: codeObjectName,
    },
    TestsLocation: model.FileLocation{
        BucketName: "tests",
        ObjectName: fmt.Sprintf("%s.json",
            taskID),
    },
    Compiler: compiler,
}

jsonBytes, err := json.Marshal(taskCommand)
if err != nil {
    http.Error(w, "Failed to marshal task",
        http.StatusInternalServerError)
    return
}
err = h.redisClient.Publish(ctx, "

```

```

        coderunner_task_channel", string(jsonBytes)).Err
    ()
    if err != nil {
        http.Error(w, "Failed to publish task",
            http.StatusInternalServerError)
        return
    }

    task := model.Task{
        ID:                taskCommand.ID,
        CodeLocation:      taskCommand.CodeLocation,
        TestsLocation:     taskCommand.TestsLocation,
        Compiler:          taskCommand.Compiler,
        State:             model.CompilingTaskState,
    }
    taskJSON, err := json.Marshal(task)
    if err != nil {
        http.Error(w, "Failed to marshal task",
            http.StatusInternalServerError)
        return
    }
    h.redisClient.Set(ctx, fmt.Sprintf("task:%s", task
        .ID), string(taskJSON), 0)

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(map[string]string{
        "task_id": task.ID})
}

func (h *TaskHandler) GetTask(w http.ResponseWriter, r *
    http.Request) {
    ctx := r.Context()
    vars := mux.Vars(r)
    taskID := vars["id"]

```



```

taskJSON, err := h.redisClient.Get(ctx, fmt.Sprintf("task:%s", taskID)).Result()
if err == redis.Nil {
    http.Error(w, "Task not found", http.StatusNotFound)
    return
} else if err != nil {
    http.Error(w, "Failed to retrieve task", http.StatusInternalServerError)
    return
}

var task model.Task
if err := json.Unmarshal([]byte(taskJSON), &task);
err != nil {
    http.Error(w, "Failed to unmarshal task", http.StatusInternalServerError)
    return
}

w.Header().Set("Content-Type", "application/json")
json.NewEncoder(w).Encode(task)
}

func (h *TaskHandler) ListTasks(w http.ResponseWriter, r *
http.Request) {
    ctx := r.Context()
    keys, err := h.redisClient.Keys(ctx, "task:*").
Result()
if err != nil {
    http.Error(w, "Failed to list tasks", http
.StatusInternalServerError)
    return
}

```

```

    }

    tasks := make([]model.Task, 0, len(keys))
    for _, key := range keys {
        taskJSON, err := h.redisClient.Get(ctx,
            key).Result()
        if err != nil {
            continue
        }
        var task model.Task
        if err := json.Unmarshal([]byte(taskJSON),
            &task); err != nil {
            continue
        }
        tasks = append(tasks, task)
    }

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(tasks)
}

```

ПРИЛОЖЕНИЕ В

Описание структуры решений пользователя

```
package model

const (
    CompilingTaskState = "compiling"
    TestingTaskState   = "testing"
    CompletedTaskState = "completed"
)

type StartTaskCommand struct {
    ID            string      `json:"id"`
    CodeLocation  FileLocation `json:"codeLocation"`
    TestsLocation FileLocation `json:"testsLocation"`
    Compiler      string      `json:"compiler"`
}

type Task struct {
    ID            string      `json:"id"`
    CodeLocation  FileLocation `json:"codeLocation"`
    TestsLocation FileLocation `json:"testsLocation"`
    ExecutableLocation FileLocation `json:"executableLocation"`
    Compiler      string      `json:"compiler"`
    State         string      `json:"state"`
    TestResults   []TestResult `json:"testResults"`
}
```

ПРИЛОЖЕНИЕ Г

Описание структуры тестовых значений

```
package model

import "encoding/json"

type TestDTO struct {
    Stdin  string `json:"stdin"`
    Stdout string `json:"stdout"`
}

type Test struct {
    ID      int    `json:"id"`
    Stdin   string `json:"stdin"`
    Stdout  string `json:"stdout"`
}

type TestResult struct {
    TestNumber int    `json:"test_number"`
    Passed      bool  `json:"passed"`
}

func ParseTestsJSON(data []byte) ([]TestDTO, error) {
    var tests []TestDTO
    err := json.Unmarshal(data, &tests)
    return tests, err
}
```

ПРИЛОЖЕНИЕ Д

Точка вхождения программы

```
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "log"
    "net/http"
    "os"

    "github.com/Feedonya/Runner/backend/filesctl"
    "github.com/Feedonya/Runner/backend/handler"
    "github.com/Feedonya/Runner/backend/model"
    "github.com/gorilla/mux"
    "github.com/redis/go-redis/v9"
    "github.com/rs/cors"

    "github.com/minio/minio-go/v7"
    "github.com/minio/minio-go/v7/pkg/credentials"
)

func main() {
    ctx := context.Background()

    minioClient, err := minio.New(os.Getenv("MINIO_ENDPOINT"),
        &minio.Options{
            Creds:  credentials.NewStaticV4(os.Getenv("MINIO_ACCESS_KEY"), os.Getenv("MINIO_SECRET_KEY"), ""),
            Secure: false,
        })
    if err != nil {
        log.Fatalf("Failed to initialize MinIO client: %v", err)
    }
}
```

```

}

buckets := []string{"code", "tests", "executables"}
for _, bucket := range buckets {
exists, err := minioClient.BucketExists(ctx, bucket)
if err != nil {
log.Fatalf("Failed to check bucket %s: %v", bucket, err)
}
if !exists {
err = minioClient.MakeBucket(ctx, bucket, minio.
    MakeBucketOptions{})
if err != nil {
log.Fatalf("Failed to create bucket %s: %v",
    bucket, err)
}
log.Printf("Created bucket %s", bucket)
}
}

filesManager := filesctl.NewMinioManager(minioClient)

redisClient := redis.NewClient(&redis.Options{
Addr:      fmt.Sprintf("%s:%s", os.Getenv("DRAGONFLY_HOST"),
    os.Getenv("DRAGONFLY_PORT")),
Password: os.Getenv("DRAGONFLY_PASSWORD"),
DB:        0,
})
_, err = redisClient.Ping(ctx).Result()
if err != nil {
log.Fatalf("Failed to connect to DragonFly: %v", err)
}

go func() {
pubsub := redisClient.Subscribe(ctx, "

```

```

    coderunner_completed_tasks_channel")
defer pubsub.Close()

for msg := range pubsub.Channel() {
log.Printf("Received raw message from channel: %s", msg.
    Payload)

var task model.Task
if err := json.Unmarshal([]byte(msg.Payload), &task); err
    != nil {
    log.Printf("Error unmarshaling into model.Task: %v
        ", err)
} else {
    log.Printf("Successfully unmarshaled task: %+v",
        task)
}

var rawData map[string]interface{}
if err := json.Unmarshal([]byte(msg.Payload), &rawData);
    err != nil {
    log.Printf("Error unmarshaling into generic map: %
        v", err)
    continue
}
log.Printf("Raw data structure: %+v", rawData)

if id, ok := rawData["id"].(string); ok {
    task.ID = id
}
if state, ok := rawData["state"].(string); ok {
    task.State = state
}
if rawData["testResults"] != nil {
    task.State = "completed"
}

```

```

        log.Printf("Overriding state to 'completed' due to
                    presence of testResults")
    }

    if results, ok := rawData["testResults"].([]interface{});
    ok {
        task.TestResults = make([]model.TestResult, len(
            results))
        for i, r := range results {
            resultMap, ok := r.(map[string]interface{}))
            if !ok {
                log.Printf("Invalid test result
                            format at index %d", i)
                continue
            }
            if testID, ok := resultMap["test_id"].(
                float64); ok {
                task.TestResults[i] = model.
                    TestResult{
                        TestNumber: int(testID),
                        Passed:      resultMap["
                            successful"].(bool),
                    }
            } else {
                if successful, ok := resultMap["
                    successful"].(bool); ok {
                    task.TestResults[i] =
                        model.TestResult{
                            Passed: successful
                        },
                    }
            }
        }
    }
}

```



```

    }
}

existingTaskJSON, err := redisClient.Get(ctx, fmt.Sprintf(
    "task:%s", task.ID)).Result()
if err == nil {
    var existingTask model.Task
    if err := json.Unmarshal([]byte(existingTaskJSON),
        &existingTask); err == nil {
        if task.CodeLocation.BucketName == "" {
            task.CodeLocation = existingTask.
                CodeLocation
        }
        if task.TestsLocation.BucketName == "" {
            task.TestsLocation = existingTask.
                TestsLocation
        }
        if task.ExecutableLocation.BucketName ==
            "" {
            task.ExecutableLocation =
                existingTask.ExecutableLocation
        }
        if task.Compiler == "" {
            task.Compiler = existingTask.
                Compiler
        }
    }
}

jsonBytes, err := json.Marshal(task)
if err != nil {
    log.Printf("Error marshaling task: %v", err)
    continue
}

```

```

err = redisClient.Set(ctx, fmt.Sprintf("task:%s", task.ID)
    , string(jsonBytes), 0).Err()
if err != nil {
    log.Printf("Error setting task in DragonFly: %v",
        err)
    continue
}
log.Printf("Updated task %s in DragonFly: %s", task.ID,
    string(jsonBytes))
}
}()

router := mux.NewRouter()
taskHandler := handler.NewTaskHandler(filesManager,
    redisClient)
router.HandleFunc("/api/tasks", taskHandler.CreateTask).
    Methods("POST")
router.HandleFunc("/api/tasks/{id}", taskHandler.GetTask).
    Methods("GET")
router.HandleFunc("/api/tasks", taskHandler.ListTasks).
    Methods("GET")

c := cors.New(cors.Options{
    AllowedOrigins:    []string{"http://localhost", "http
        ://127.0.0.1"},
    AllowedMethods:    []string{"GET", "POST", "OPTIONS"},
    AllowedHeaders:    []string{"Content-Type"},
    AllowCredentials: true,
})
handler := c.Handler(router)

port := os.Getenv("SERVER_PORT")
if port == "" {
    port = "8080"

```

```
}  
log.Printf("Starting server on port %s", port)  
if err := http.ListenAndServe(":"+port, handler); err !=  
    nil {  
log.Fatalf("Server failed: %v", err)  
}  
}
```

ПРИЛОЖЕНИЕ Е

Использование механизма Pub/Sub в микросервисе

```
package handler

import (
    "context"
    "encoding/json"
    "fmt"

    "github.com/redis/go-redis/v9"
    "github.com/t3m8ch/coderunner/internal/model"
)

func HandleStartTaskCommands(
    ctx context.Context,
    redisClient *redis.Client,
    tasksToCompile chan model.Task,
) {
    pubsub := redisClient.Subscribe(ctx, taskChannel)
    for msg := range pubsub.Channel() {
        var taskCommand model.StartTaskCommand
        err := json.Unmarshal([]byte(msg.Payload),
            &taskCommand)
        if err != nil {
            fmt.Printf("Error unmarshaling
                task: %v\n", err)
            continue
        }

        fmt.Printf("Received task: %+v\n",
            taskCommand)

        task := model.Task{
            ID: taskCommand.ID,
```

```

        CodeLocation: taskCommand.
            CodeLocation,
        TestsLocation: taskCommand.
            TestsLocation,
        Compiler: taskCommand.
            Compiler,
        State: model.
            CompilingTaskState,
    }
    jsonBytes, err := json.Marshal(task)
    if err != nil {
        fmt.Printf("Error marshaling task:
            %v\n", err)
        continue
    }

    redisClient.Set(ctx, fmt.Sprintf("task:%s
        ", taskCommand.ID), string(jsonBytes), 0)

    tasksToCompile <- task
    }
}

```

ПРИЛОЖЕНИЕ Ж

Компиляция решения

```
package handler

import (
    "context"
    "fmt"

    "github.com/t3m8ch/coderunner/internal/filesctl"
    "github.com/t3m8ch/coderunner/internal/model"
    "github.com/t3m8ch/coderunner/internal/sandbox"
)

func HandleTasksToCompile(
    ctx context.Context,
    filesManager filesctl.Manager,
    sandboxManager sandbox.Manager,
    tasksToCompile chan model.Task,
    tasksToTest chan model.Task,
) {
    for task := range tasksToCompile {
        handleTaskToCompile(ctx, filesManager,
            sandboxManager, task, tasksToTest)
    }
}

func handleTaskToCompile(
    ctx context.Context,
    filesManager filesctl.Manager,
    sandboxManager sandbox.Manager,
    task model.Task,
    tasksToTest chan model.Task,
) {
    fmt.Printf("Task to compile: %+v\n", task)
```

```

codeBinary, err := filesManager.LoadFile(ctx, task
    .CodeLocation.BucketName, task.CodeLocation.
    ObjectName)
if err != nil {
    fmt.Printf("Error loading code from file
        server: %v\n", err)
    return
}

sandboxID, err := sandboxManager.CreateSandbox(
ctx,
"gcc:latest",
[]string{"g++", sourceFilePath, "-o",
    compileExecPath, "-static"},
)
if err != nil {
    fmt.Printf("Error creating sandbox: %v\n",
        err)
    return
}

defer func() {
    err = sandboxManager.RemoveSandbox(ctx,
        sandboxID)
    if err != nil {
        fmt.Printf("Error sandbox removing
            : %v\n", err)
    }
}()

err = sandboxManager.CopyFileToSandbox(ctx,
    sandboxID, sourceFilePath, 0644, codeBinary)
if err != nil {

```

```

        fmt.Printf("Error copying code to sandbox:
            %v\n", err)
        return
    }

    err = sandboxManager.StartSandbox(ctx, sandboxID)
    if err != nil {
        fmt.Printf("Error starting sandbox: %v\n",
            err)
        return
    }

    statusCode, err := sandboxManager.WaitSandbox(ctx,
        sandboxID)
    if err != nil {
        fmt.Printf("Error waiting for sandbox: %v\n
            n", err)
        return
    }
    if statusCode != 0 {
        fmt.Printf("Compilation failed with exit
            code %d\n", statusCode)
        logs, err := sandboxManager.
            ReadLogsFromSandbox(ctx, sandboxID)
        if err != nil {
            fmt.Printf("Error reading logs
                from sandbox: %v\n", err)
        }
        fmt.Println(logs)
        return
    }

    executable, err := sandboxManager.
        LoadFileFromSandbox(ctx, sandboxID,

```



```

        compileExecPath)
    if err != nil {
        fmt.Printf("Error copying executable: %v\n", err)
        return
    }

    objectName := fmt.Sprintf("%s.out", task.ID)

    err = filesManager.PutFile(
        ctx,
        execBucketName,
        objectName,
        executable,
    )
    if err != nil {
        fmt.Printf("Error put object to file
            server: %v\n", err)
        return
    }

    task.State = model.TestingTaskState
    task.ExecutableLocation = model.FileLocation{
        BucketName: execBucketName,
        ObjectName: objectName,
    }
    tasksToTest <- task
}

```

ПРИЛОЖЕНИЕ 3

Тестирование программы

```
package handler

import (
    "context"
    "encoding/json"
    "fmt"
    "strings"
    "sync"

    "github.com/redis/go-redis/v9"
    "github.com/t3m8ch/coderunner/internal/filesctl"
    "github.com/t3m8ch/coderunner/internal/model"
    "github.com/t3m8ch/coderunner/internal/sandbox"
)

func HandleTasksToTest(
    ctx context.Context,
    filesManager filesctl.Manager,
    sandboxManager sandbox.Manager,
    tasksToTest chan model.Task,
    redisClient *redis.Client,
) {
    for task := range tasksToTest {
        handleTaskToTest(ctx, filesManager,
            sandboxManager, redisClient, task)
    }
}

func handleTaskToTest(
    ctx context.Context,
    filesManager filesctl.Manager,
    sandboxManager sandbox.Manager,
```

```

redisClient *redis.Client,
task model.Task,
) {
    fmt.Printf("Task to test: %+v\n", task)

    executable, err := filesManager.LoadFile(
ctx,
task.ExecutableLocation.BucketName,
task.ExecutableLocation.ObjectName,
)
    if err != nil {
        fmt.Printf("Error loading executable from
MinIO: %v\n", err)
        return
    }
    fmt.Println("Executable loaded")

    testsData, err := filesManager.LoadFile(
ctx,
task.TestsLocation.BucketName,
task.TestsLocation.ObjectName,
)
    if err != nil {
        fmt.Printf("Error loading tests from MinIO
: %v\n", err)
        return
    }
    fmt.Println("Tests loaded")

    tests, err := model.ParseTestsJSON(testsData)
    if err != nil {
        fmt.Printf("Error parsing tests JSON: %v\n
", err)
        return
    }
}

```

```

    }
    fmt.Println("Tests parsed")

    var wg sync.WaitGroup
    wg.Add(len(tests))

    testsCh := make(chan model.Test, 20)
    testsResultsCh := make(chan model.TestResult, len(
        tests))

    go func() {
        for i := range tests {
            testsCh <- model.Test{
                ID:      i,
                Stdin:    tests[i].Stdin,
                Stdout:   tests[i].Stdout,
            }
        }
    }()

    go func() {
        fmt.Println("Waiting...")
        wg.Wait()
        fmt.Println("Done! Closing testsResultsCh
            and testsCh")
        close(testsResultsCh)
        close(testsCh)
    }()

    for test := range testsCh {
        go func() {
            defer wg.Done()

            fmt.Printf("———— Test #%d ——— \n", test

```

```

.ID)

sandboxID, err := sandboxManager.
    CreateSandbox(
ctx,
"debian:bookworm",
[]string{"sh", "-c", fmt.Sprintf("%s < %s",
    testingExecPath, inputFilePath)},
)
if err != nil {
    fmt.Printf("test #%d: Error
        creating sandbox: %v\n", test.ID,
        err)
    return
}
fmt.Printf("test #%d: Sandbox created\n",
    test.ID)

err = sandboxManager.CopyFileToSandbox(ctx
    , sandboxID, testingExecPath, 0700,
    executable)
if err != nil {
    fmt.Printf("test #%d: Error
        copying executable to sandbox: %v
        \n", test.ID, err)
    return
}
fmt.Printf("test #%d: Executable copied to
    sandbox\n", test.ID)

err = sandboxManager.CopyFileToSandbox(ctx
    , sandboxID, inputFilePath, 0644, []byte(
    test.Stdin))
if err != nil {

```

```

        fmt.Printf("test #%d: Error
                    copying input data: %v\n", test.
                    ID, err)
        return
    }

    err = sandboxManager.StartSandbox(ctx,
        sandboxID)
    if err != nil {
        fmt.Printf("test #%d: Error
                    starting sandbox: %v\n", test.ID,
                    err)
        return
    }
    fmt.Printf("test #%d: Sandbox started\n",
        test.ID)

    statusCode, err := sandboxManager.
        WaitSandbox(ctx, sandboxID)
    if err != nil {
        fmt.Printf("test #%d: Error
                    waiting for sandbox: %v\n", test.
                    ID, err)
        return
    }

    output, err := sandboxManager.
        ReadLogsFromSandbox(ctx, sandboxID)
    if err != nil {
        fmt.Printf("test #%d: Error
                    reading logs from sandbox: %v\n",
                    test.ID, err)
        return
    }

```

```

fmt.Printf("test #%d: Output read from
    sandbox\n", test.ID)
fmt.Printf("test #%d: %s", test.ID, output
)

fmt.Printf("test #%d: Testing completed
    with exit code %d\n", test.ID, statusCode
)

output = strings.Trim(output, " ")
output = strings.Trim(output, "\n")
output = strings.Trim(output, "\t")

test.Stdout = strings.Trim(output, " ")
test.Stdout = strings.Trim(output, "\n")
test.Stdout = strings.Trim(output, "\t")

if statusCode == 0 && output == test.
    Stdout {
        fmt.Printf("test #%d: Test passed\n
            n", test.ID)
        testsResultsCh <- model.TestResult
            {TaskID: task.ID, TestID: test.ID
            , Successful: true}
    } else {
        testsResultsCh <- model.TestResult
            {TaskID: task.ID, TestID: test.ID
            , Successful: false}
        fmt.Printf("test #%d: Test failed\n
            n", test.ID)
        fmt.Printf("test #%d: Expected: %s
            \n", test.ID, test.Stdout)
        fmt.Printf("test #%d: Actual: %s\n
            ", test.ID, output)
    }

```

```

        fmt.Printf("test #%d: Expected
            bytes: %q\n", test.ID, []byte(
                test.Stdout))
        fmt.Printf("test #%d: Actual bytes
            :   %q\n", test.ID, []byte(output
                ))
    }

    err = sandboxManager.RemoveSandbox(ctx ,
        sandboxID)
    if err != nil {
        fmt.Printf("test #%d: Error
            sandbox removing: %v\n", test.ID,
                err)
    }
    fmt.Printf("test #%d: Sandbox removed\n",
        test.ID)
}()
}

task.TestsResults = make([]model.TestResult , 0,
    len(tests))
for test := range testsResultsCh {
    task.TestsResults = append(task.
        TestsResults , test)
    jsonBytes , err := json.Marshal(test)
    if err != nil {
        fmt.Printf("test #%d: Error
            marshaling test result: %v\n",
                test.TestID , err)
    }
    redisClient.Publish(ctx ,
        completedTestsChannel , string(jsonBytes))
}

```



```

fmt.Println("All tests completed!")
fmt.Println(task.TestsResults)

jsonBytes, err := json.Marshal(task)
if err != nil {
    fmt.Printf("Error marshaling task: %v\n",
        err)
}
redisClient.Publish(ctx, completedTasksChannel,
    string(jsonBytes))
}

```

ПРИЛОЖЕНИЕ И

Создание изолированного контейнера для компиляции решений

```
package sandbox
```

```
import (  
    "archive/tar"  
    "bytes"  
    "context"  
    "encoding/binary"  
    "fmt"  
    "io"
```

```
"github.com/docker/docker/api/types/container"  
docker "github.com/docker/docker/client"  
)
```

```
type DockerManager struct {  
    dockerClient *docker.Client  
}
```

```
func NewDockerManager(dockerClient *docker.Client) *  
    DockerManager {  
    return &DockerManager{dockerClient}  
}
```

```
func (m *DockerManager) CreateSandbox(ctx context.Context,  
    image string, cmd []string) (SandboxID, error) {  
    resp, err := m.dockerClient.ContainerCreate(  
        ctx,  
        &container.Config{  
            AttachStdin: true,  
            AttachStdout: true,  
            AttachStderr: true,  
            OpenStdin: true,
```

```

        StdinOnce:    true ,
        Image:        image ,
        Cmd:          cmd ,
    },
    nil ,
    nil ,
    nil ,
    "",
)
if err != nil {
    return "", err
}

return resp.ID, nil
}

func (m *DockerManager) StartSandbox(ctx context.Context,
    id SandboxID) error {
    return m.dockerClient.ContainerStart(ctx, id,
        container.StartOptions{})
}

func (m *DockerManager) AttachToSandbox(ctx context.
    Context, id SandboxID) (io.Reader, io.WriteCloser, error)
{
    resp, err := m.dockerClient.ContainerAttach(ctx,
        id, container.AttachOptions{
            Stream: true,
            Stdin:  true,
            Stdout: true,
            Stderr: true,
        })
    if err != nil {
        return nil, nil, err
    }

```

```

    }
    return resp.Reader, resp.Conn, nil
}

func (m *DockerManager) RemoveSandbox(ctx context.Context,
    id SandboxID) error {
    return m.dockerClient.ContainerRemove(ctx, id,
        container.RemoveOptions{Force: true})
}

func (m *DockerManager) CopyFileToSandbox(ctx context.
    Context, id SandboxID, path string, mode int64, data []
    byte) error {
    var buf bytes.Buffer
    tw := tar.NewWriter(&buf)
    hdr := &tar.Header{
        Name: path,
        Mode: mode,
        Size: int64(len(data)),
    }
    if err := tw.WriteHeader(hdr); err != nil {
        return err
    }
    if _, err := tw.Write(data); err != nil {
        return err
    }
    tw.Close()

    return m.dockerClient.CopyToContainer(
        ctx,
        id,
        "/",
        &buf,
        container.CopyToContainerOptions{},

```

```

    )
}

func (m *DockerManager) LoadFileFromSandbox(ctx context.
Context, id SandboxID, path string) ([]byte, error) {
    reader, _, err := m.dockerClient.CopyFromContainer
        (ctx, id, path)
    if err != nil {
        return nil, err
    }
    defer reader.Close()

    tarReader := tar.NewReader(reader)
    tarReader.Next()
    data, err := io.ReadAll(tarReader)
    if err != nil {
        return nil, err
    }

    return data, nil
}

func (m *DockerManager) WaitSandbox(ctx context.Context,
id SandboxID) (StatusCode, error) {
    statusCh, errCh := m.dockerClient.ContainerWait(
        ctx, id, container.WaitConditionNotRunning)
    select {
        case err := <-errCh:
            return -1, err
        case status := <-statusCh:
            return status.StatusCode, nil
    }
}

```

```

func (m *DockerManager) ReadLogsFromSandbox(ctx context.
Context, id SandboxID) (string, error) {
    reader, err := m.dockerClient.ContainerLogs(ctx,
        id, container.LogsOptions{
            ShowStdout: true,
            ShowStderr: true,
            Timestamps: false,
            Details:    false,
            Follow:    false,
        })
    if err != nil {
        return "", err
    }
    defer reader.Close()

    var buf bytes.Buffer

    // Это код, сгенерированный DeepSeek для очистки
    строки от всякого мусора.
    // Слава великой китайской аболе!

    // Полный ответ DeepSeek'а по ссылке: https://
    pastebin.com/UZQadXsf

    // Читаем логи с обработкой Docker-заголовков
    header := make([]byte, 8)
    for {
        // Читаем заголовок
        _, err := io.ReadFull(reader, header)
        if err != nil {
            if err == io.EOF {
                break
            }
            return "", fmt.Errorf("failed to

```

```

        read header: %w", err)
    }

    // Разбираем размер данных (последние 4
    // байта заголовка, big-endian)
    dataSize := binary.BigEndian.Uint32(header
        [4:8])

    // Читаем данные
    data := make([]byte, dataSize)
    _, err = io.ReadFull(reader, data)
    if err != nil {
        return "", fmt.Errorf("failed to
            read data: %w", err)
    }

    // Записываем данные в буфер
    buf.Write(data)
}

return buf.String(), nil
}

```

ПРИЛОЖЕНИЕ К

Ограничения по компиляции

```
package sandbox

import (
    "context"
    "io"
)

type ConcurrencyLimitDecorator struct {
    manager    Manager
    semaphore chan struct{}
}

func NewConcurrencyLimitDecorator(manager Manager,
    maxConcurrent int) Manager {
    return &ConcurrencyLimitDecorator{
        manager:    manager,
        semaphore: make(chan struct{}),
        maxConcurrent),
    }
}

func (d *ConcurrencyLimitDecorator) acquire(ctx context.
    Context) error {
    select {
        case d.semaphore <- struct{}{}:
            return nil
        case <-ctx.Done():
            return ctx.Err()
    }
}

func (d *ConcurrencyLimitDecorator) release() {
```



```

        <-d.semaphore
    }

func (d *ConcurrencyLimitDecorator) CreateSandbox(ctx
    context.Context, image string, cmd []string) (SandboxID,
    error) {
    if err := d.acquire(ctx); err != nil {
        return "", err
    }
    defer d.release()
    return d.manager.CreateSandbox(ctx, image, cmd)
}

func (d *ConcurrencyLimitDecorator) StartSandbox(ctx
    context.Context, id SandboxID) error {
    if err := d.acquire(ctx); err != nil {
        return err
    }
    defer d.release()
    return d.manager.StartSandbox(ctx, id)
}

func (d *ConcurrencyLimitDecorator) AttachToSandbox(ctx
    context.Context, id SandboxID) (io.Reader, io.WriteCloser
    , error) {
    if err := d.acquire(ctx); err != nil {
        return nil, nil, err
    }
    defer d.release()
    return d.manager.AttachToSandbox(ctx, id)
}

func (d *ConcurrencyLimitDecorator) RemoveSandbox(ctx
    context.Context, id SandboxID) error {

```

```

        if err := d.acquire(ctx); err != nil {
            return err
        }
        defer d.release()
        return d.manager.RemoveSandbox(ctx, id)
    }

func (d *ConcurrencyLimitDecorator) CopyFileToSandbox(ctx
    context.Context, id SandboxID, path string, mode int64,
    data []byte) error {
    if err := d.acquire(ctx); err != nil {
        return err
    }
    defer d.release()
    return d.manager.CopyFileToSandbox(ctx, id, path,
        mode, data)
}

func (d *ConcurrencyLimitDecorator) LoadFileFromSandbox(
    ctx context.Context, id SandboxID, path string) ([]byte,
    error) {
    if err := d.acquire(ctx); err != nil {
        return nil, err
    }
    defer d.release()
    return d.manager.LoadFileFromSandbox(ctx, id, path
    )
}

func (d *ConcurrencyLimitDecorator) WaitSandbox(ctx
    context.Context, id SandboxID) (StatusCode, error) {
    if err := d.acquire(ctx); err != nil {
        return -1, err
    }
}

```

```

    defer d.release()
    return d.manager.WaitSandbox(ctx, id)
}

func (d *ConcurrencyLimitDecorator) ReadLogsFromSandbox(
    ctx context.Context, id SandboxID) (string, error) {
    if err := d.acquire(ctx); err != nil {
        return "", err
    }
    defer d.release()
    return d.manager.ReadLogsFromSandbox(ctx, id)
}

```

ПРИЛОЖЕНИЕ Л

Взаимодействие с песочницей

```
package sandbox

import (
    "bytes"
    "context"
    "fmt"
    "io"
    "time"

    "github.com/docker/docker/api/types/container"
    docker "github.com/docker/docker/client"
    "github.com/docker/docker/pkg/stdcopy"
)

type TMPFSDockerManager struct {
    dockerClient *docker.Client
    cmd          []string
    execIDs      map[SandboxID]string
    execOutputs  map[SandboxID]string
    outputReady  map[SandboxID]chan struct{}
}

func NewTMPFSDockerManager(dockerClient *docker.Client)
    Manager {
    return &TMPFSDockerManager{
        dockerClient: dockerClient,
        cmd:          make([]string, 0),
        execIDs:      make(map[SandboxID]string),
        execOutputs:  make(map[SandboxID]string),
        outputReady:  make(map[SandboxID]chan
            struct{}),
    }
}
```

```

}

func (m *TMPFSDockerManager) CreateSandbox(ctx context.
Context, image string, cmd []string) (SandboxID, error) {
    m.cmd = cmd
    resp, err := m.dockerClient.ContainerCreate(
        ctx,
        &container.Config{
            AttachStdin:  true,
            AttachStdout: true,
            AttachStderr: true,
            OpenStdin:    true,
            StdinOnce:    true,
            Image:        image,
            Cmd:          []string{"tail", "-f", "/dev
                /null"},
        },
        &container.HostConfig{
            Tmpfs: map[string]string{
                "/app": "rw,exec,nosuid,size=65536
                    k",
                "/tmp": "rw,exec,nosuid,size=65536
                    k",
            },
            LogConfig: container.LogConfig{
                Type: "none",
            },
        },
        nil,
        nil,
        "",
    )
    if err != nil {
        return "", err
    }
}

```

```

    }

    err = m.dockerClient.ContainerStart(ctx, resp.ID,
        container.StartOptions{})
    if err != nil {
        return "", err
    }

    return resp.ID, nil
}

func (m *TMPFSDockerManager) StartSandbox(ctx context.
    Context, id SandboxID) error {
    execConfig := container.ExecOptions{
        AttachStdout: true,
        AttachStderr: true,
        Cmd:          m.cmd,
    }
    execResp, err := m.dockerClient.
        ContainerExecCreate(ctx, string(id), execConfig)
    if err != nil {
        return err
    }

    m.execIDs[id] = execResp.ID
    m.outputReady[id] = make(chan struct{})

    attachResp, err := m.dockerClient.
        ContainerExecAttach(
            ctx,
            execResp.ID,
            container.ExecAttachOptions{},
        )
    if err != nil {

```

```

        return err
    }

    go func() {
        var buf bytes.Buffer
        _, err := io.Copy(&buf, attachResp.Reader)
        if err != nil && err != io.EOF {
            // Логирование ошибки, если
            требуется
        }
        m.execOutputs[id] = buf.String()
        close(m.outputReady[id])
        attachResp.Close()
    }()

    return nil
}

func (m *TMPFSDockerManager) AttachToSandbox(ctx context.
Context, id SandboxID) (io.Reader, io.WriteCloser, error)
{
    resp, err := m.dockerClient.ContainerAttach(ctx,
        id, container.AttachOptions{
            Stream: true,
            Stdin: true,
            Stdout: true,
            Stderr: true,
        })
    if err != nil {
        return nil, nil, err
    }
    return resp.Reader, resp.Conn, nil
}

```

```

func (m *TMPFSDockerManager) RemoveSandbox(ctx context.
    Context, id SandboxID) error {
    return m.dockerClient.ContainerRemove(ctx, id,
        container.RemoveOptions{Force: true})
}

func (m *TMPFSDockerManager) CopyFileToSandbox(ctx context
    .Context, id SandboxID, path string, mode int64, data []
    byte) error {
    // Convert mode to octal string for chmod
    modeStr := fmt.Sprintf("%o", mode)

    // Create exec config
    execConfig := container.ExecOptions{
        AttachStdin: true,
        AttachStdout: false,
        AttachStderr: false,
        Tty: false,
        Cmd: []string{"/bin/sh", "-c", "
            mkdir -p \"$(dirname \"$1\")\" && cat >
            \"$1\" && chmod $2 \"$1\"", "-", path,
            modeStr},
        // Cmd: []string{"ls"},
    }

    // Create exec instance
    execResp, err := m.dockerClient.
        ContainerExecCreate(ctx, id, execConfig)
    if err != nil {
        return err
    }
    fmt.Println("Container exec created")

    // Attach to exec instance with stdin

```



```

attachResp, err := m.dockerClient.
    ContainerExecAttach(ctx, execResp.ID, container.
        ExecAttachOptions{
            Detach: false,
        })
if err != nil {
    return err
}
defer attachResp.Close()
fmt.Println("Container exec attached")

// Write the byte array to stdin
_, err = io.Copy(attachResp.Conn, bytes.NewReader(
    data))
if err != nil {
    return err
}
fmt.Println("Container exec data written")

// Close stdin to signal EOF
err = attachResp.Conn.Close()
if err != nil {
    return err
}
fmt.Println("Container exec stdin closed")

// Wait for exec to finish
for {
    inspect, err := m.dockerClient.
        ContainerExecInspect(ctx, execResp.ID)
    if err != nil {
        return err
    }
    if !inspect.Running {

```

```

        if inspect.ExitCode != 0 {
            return fmt.Errorf("failed
                to write file: exit code
                %d", inspect.ExitCode)
        }
        fmt.Println("Container exec
            finished")
        return nil
    }
    time.Sleep(100 * time.Millisecond)
}
}

```

```

func (m *TMPFSDockerManager) LoadFileFromSandbox(ctx
    context.Context, id SandboxID, path string) ([]byte,
    error) {
    execConfig := container.ExecOptions{
        AttachStdout: true,
        AttachStderr: true,
        Cmd:          []string{"cat", path},
    }
    execResp, err := m.dockerClient.
        ContainerExecCreate(ctx, string(id), execConfig)
    if err != nil {
        return nil, err
    }
    attachResp, err := m.dockerClient.
        ContainerExecAttach(ctx, execResp.ID, container.
            ExecStartOptions{})
    if err != nil {
        return nil, err
    }
    defer attachResp.Close()
    var stdout, stderr bytes.Buffer

```

```

_, err = stdcopy.StdCopy(&stdout, &stderr,
    attachResp.Reader)
if err != nil {
    return nil, err
}
inspectResp, err := m.dockerClient.
    ContainerExecInspect(ctx, execResp.ID)
if err != nil {
    return nil, err
}
if inspectResp.ExitCode != 0 {
    return nil, fmt.Errorf("failed to read
        file: exit code %d, stderr: %s",
        inspectResp.ExitCode, stderr.String())
}
return stdout.Bytes(), nil
}

func (m *TMPFSDockerManager) WaitSandbox(ctx context.
    Context, id SandboxID) (StatusCode, error) {
    execID, ok := m.execIDs[id]
    if !ok {
        return -1, fmt.Errorf("no exec ID found
            for container %s", id)
    }

    for {
        execInspect, err := m.dockerClient.
            ContainerExecInspect(ctx, execID)
        if err != nil {
            return -1, err
        }
        if !execInspect.Running {
            return int64(execInspect.ExitCode)

```

```

        , nil
    }
    time.Sleep(100 * time.Millisecond) // Wait
        a short time before checking again
    }
}

func (m *TMPFSDockerManager) ReadLogsFromSandbox(ctx
context.Context, id SandboxID) (string, error) {
    readyCh, ok := m.outputReady[id]
    if !ok {
        return "", fmt.Errorf("no output ready for
            container %s", id)
    }
    select {
    case <-readyCh:
        output, ok := m.execOutputs[id]
        if !ok {
            return "", fmt.Errorf("no output
                for container %s", id)
        }
        return output, nil
    case <-ctx.Done():
        return "", ctx.Err()
    }
}

```

ПРИЛОЖЕНИЕ М

Форма на клиентской части приложения

```
import React, { useState } from 'react';

const TaskForm = ({ setShowStatus, setCurrentTaskId }) => {
  const [taskId, setTaskId] = useState('');
  const [compiler, setCompiler] = useState('g++');
  const [codeFile, setCodeFile] = useState(null);
  const [error, setError] = useState('');
  const [submitting, setSubmitting] = useState(false);

  const handleSubmit = async (e) => {
    e.preventDefault();
    console.log('Submitting:', { taskId, compiler, codeFile });
    ;
    if (!taskId || !compiler || !codeFile) {
      setError('Please fill all fields and select a file .');
      return;
    }

    setError('');
    setSubmitting(true);

    const formData = new FormData();
    formData.append('task_id', taskId);
    formData.append('compiler', compiler);
    formData.append('code', codeFile);

    try {
      const response = await fetch('/api/tasks', {
        method: 'POST',
        body: formData,
```

```

    });
    if (!response.ok) throw new Error('Failed to
      submit');
    const data = await response.json();
    console.log('Response:', data);
    setCurrentTaskId(data.task_id);
    setShowStatus(true);
  } catch (err) {
    setError(err.message);
    console.error('Fetch error:', err);
  } finally {
    setSubmitting(false);
  }
};

return (
  <div className="p-4 max-w-md mx-auto bg-white rounded-lg
    shadow-md">
    <h2 className="text-xl font-bold mb-4">Submit Solution</h2>
    <
    <form onSubmit={handleSubmit}>
    <div className="mb-4">
    <label className="block text-sm font-medium text-gray
      -700">Task ID</label>
    <input
      type="text"
      value={taskId}
      onChange={(e) => setTaskId(e.target.value)}
      className="mt-1 block w-full border-gray-300 rounded-md
        shadow-sm"
      placeholder="task_123"
    />
    </div>
    <div className="mb-4">

```

```

<label className="block text-sm font-medium text-gray
  -700">Compiler</label>
<select
  value={compiler}
  onChange={(e) => setCompiler(e.target.value)}
  className="mt-1 block w-full border-gray-300 rounded-md
    shadow-sm"
>
<option value="g++">g++</option>
</select>
</div>
<div className="mb-4">
<label className="block text-sm font-medium text-gray
  -700">Code File</label>
<input
  type="file"
  accept=".cpp" // Restrict to .cpp files
  onChange={(e) => setCodeFile(e.target.files[0])}
  className="mt-1 block w-full border-gray-300 rounded-md
    shadow-sm"
/>
</div>
{error && <p className="text-red-500 text-sm mb-4">{error
  }</p>}}
<button
  type="submit"
  disabled={submitting}
  className="w-full bg-indigo-600 text-white py-2 px-4
    rounded-md hover:bg-indigo-700 disabled:bg-gray-400"
>
Submit
</button>
</form>
</div>

```

```
);  
};
```

```
export default TaskForm;
```


ПРИЛОЖЕНИЕ Н

Список решений

```
import React, { useState, useEffect } from 'react';

function TaskList() {
  const [tasks, setTasks] = useState([]);
  const [error, setError] = useState('');

  useEffect(() => {
    const fetchTasks = async () => {
      try {
        const response = await fetch('/api/tasks');
        if (!response.ok) throw new Error('Failed to fetch tasks');
        const data = await response.json();
        setTasks(data);
        setError('');
      } catch (err) {
        console.error('Fetch error:', err);
        setTimeout(fetchTasks, 2000);
      }
    };
    fetchTasks();
  }, []);

  if (error) {
    return <p className="text-red-500">{error}</p>;
  }

  return (
    <div className="mt-6">
```

```

<h2 className="text-xl font-bold mb-4">Submitted Tasks</h2
>
{error && <p className="text-red-500 text-sm mb-4">{error
  }</p>}}
<ul>
{tasks.map((task) => (
  <li key={task.id} className="mb-2">
    {task.id} - {task.state}
  </li>
))}
</ul>
</div>
);
};

export default TaskList;

```

ПРИЛОЖЕНИЕ О

Вывод статуса проверки задачи

```
import React, { useState, useEffect } from 'react';

const TaskStatus = ({ taskId }) => {
  const [status, setStatus] = useState(null);
  const [error, setError] = useState('');
  const [lastFetch, setLastFetch] = useState(null);

  useEffect(() => {
    let isMounted = true;

    const fetchStatus = async () => {
      try {
        const response = await fetch(`/api/tasks/${taskId}`);
        if (!response.ok) {
          if (response.status === 404) throw new
            Error('Task not found');
          throw new Error('Failed to fetch status');
        }
        const data = await response.json();
        if (isMounted) {
          setLastFetch(new Date().toISOString());
          if (data.state === 'completed') {
            if (!data.testResults) {
              console.warn('Completed
                but missing testResults
                at', lastFetch, '—
                retrying...');
            } else {
              setStatus(data);
              setError('');
            }
          }
        }
      }
    }
  }, [taskId]);
}
```

```

        } else {
            setStatus(data); // Update for
                               intermediate states
            setError('');
        }
    }
} catch (err) {
    console.error('Fetch error at', lastFetch, ':',
        err);
    if (isMounted && status && status.state === '
        completed') {
        setError('Failed to fetch status after
            completion');
    }
}
};

```

```

fetchStatus();
const interval = setInterval(fetchStatus, 2000); // Poll
    every 2 seconds
return () => {
    isMounted = false;
    clearInterval(interval);
};
}, [taskId]);

```

```

if (!status) return <div className="mt-6 p-4 bg-gray-100
    rounded-lg">Loading status...</div>;

```

```

return (
<div className="mt-6 p-4 bg-gray-100 rounded-lg">
<h2 className="text-xl font-bold mb-4">Task Status</h2>
{error && <p className="text-red-500 text-sm mb-4">{error
    }</p>}

```

```

<p>Task ID: {status.id}</p>
<p>State: {status.state}</p>
<p>Last Fetch: {lastFetch}</p>
{status.state === 'completed' && status.testResults && (
<p>Tests Passed: {status.testResults.filter(r => r.passed
    ).length} / {status.testResults.length}</p>
)}
</div>
);
};

export default TaskStatus;

```

ПРИЛОЖЕНИЕ П

Docker-compose, при помощи которого развертываем приложение

```
version: '3.8'

services:
  backend:
    build:
      context: ./backend
    dockerfile: Dockerfile
    ports:
      - "8080:8080"
    environment:
      - SERVER_PORT=8080
      - MINIO_ENDPOINT=minio:9000
      - MINIO_ACCESS_KEY=minioadmin
      - MINIO_SECRET_KEY=minioadmin
      - MINIO_USE_SSL=false
      - DRAGONFLY_HOST=dragonfly
      - DRAGONFLY_PORT=6379
      - DRAGONFLY_PASSWORD=
    depends_on:
      - minio
      - dragonfly
    networks:
      - app-network
    restart: unless-stopped

  frontend:
    build:
      context: ./frontend
    dockerfile: Dockerfile
    ports:
      - "80:80"
    depends_on:
```

```

— backend
networks:
— app-network
restart: unless-stopped

code-runner:
build:
context: ./code-runner
dockerfile: Dockerfile
environment:
— REDIS_HOST=dragonfly:6379
— REDIS_PASSWORD=
— REDIS_DB=0
— MINIO_ENDPOINT=minio:9000
— MINIO_ACCESS_KEY=minioadmin
— MINIO_SECRET_KEY=minioadmin
— USE_TMPFS=true
depends_on:
— minio
— dragonfly
networks:
— app-network
restart: unless-stopped
volumes:
— /var/run/docker.sock:/var/run/docker.sock

minio:
image: minio/minio:latest
ports:
— "9000:9000"
— "9001:9001"
environment:
— MINIO_ROOT_USER=minioadmin
— MINIO_ROOT_PASSWORD=minioadmin

```

```

command: server /data --console-address ":9001"
volumes:
- minio-data:/data
networks:
- app-network
restart: unless-stopped

dragonfly:
image: docker.dragonflydb.io/dragonflydb/dragonfly
ports:
- "6379:6379"
command: dragonfly --proactor_threads=4
networks:
- app-network
restart: unless-stopped
ulimits:
memlock: -1

networks:
app-network:
driver: bridge

volumes:
minio-data:

```