

СОДЕРЖАНИЕ

1 ПОСТАНОВКА И АНАЛИЗ ЗАДАЧИ.....	6
1.1 Анализ предметной области	6
1.2 Обоснование актуальности работы	8
1.2.1 Существующие решения	9
1.2.2 Сравнительная характеристика	12
1.3 Постановка задачи.....	13
1.4 Выбор языков программирования и технологий	14
2 ПРОЕКТИРОВАНИЕ ПЛАТФОРМЫ	18
2.1 Проектирование процесса проверки решения на серверной части	18
2.2 Инфологическое проектирование Базы Данных.....	19
2.3 Даталогическое проектирование Базы Данных	23
2.3.1 Формирование предварительных отношений по ER-диаграмме	23
2.3.2 Подготовка списка атрибутов. Распределение их по отношениям	24
2.4 Проектирование архитектуры платформы.	25
2.4.1 Архитектура клиента	27
2.4.2 Архитектура сервера.....	27
2.4.3 Итоговая архитектура платформы	31
3 РАЗРАБОТКА ПЛАТФОРМЫ	34
3.1 Разработка сервиса выполнения кода и проверки решения	35
3.2 Разработка сервиса задач и решений	39
3.2.1 Получение задачи с решениями пользователя по ID задачи.....	40
3.2.2 Получение списка задач доступных для решения.....	41
3.2.3 Создание задачи	43
3.2.4 Добавление тестов к задаче.....	44
3.2.5 Удаление тестов задачи	46
3.2.6 Получение собственных задач.....	47

3.2.7	Получение решений пользователя	49
3.2.8	Получение решения по ID	50
3.2.9	Создание решения	52
3.2.10	Обработка завершения решения	54
3.3	Разработка сервиса пользователей и аутентификации	56
3.3.1	Регистрация пользователя	58
3.3.2	Логин пользователя	59
3.4	Разработка сервиса соревнований.	61
3.4.1	Получение информации о соревновании по ID	61
3.4.2	Получение всех соревнований в которых участвовал пользователь	63
3.4.3	Участие в соревновании	65
3.4.4	Получение всех соревнований, которые организовал пользователь	66
3.4.5	Получения списка публичных соревнований	68
3.4.6	Создание соревнования	70
3.4.7	Получение результатов соревнования	72
3.4.8	Получения результата соревнования конкретного пользователя ...	74
3.5	Развертывание серверной части	76
3.5.1	Создание кластера Kubernetes	77
3.5.2	Сборка образов	78
3.5.3	Развертывание БД и Kafka	80
3.5.4	Развертывание сервисов	80
3.6	Разработка клиентской части	83
4	ТЕСТИРОВАНИЕ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ	89
4.1	Тестирование серверной части	89
4.2	Ручное тестирование клиентской части	90
	ОПИСАНИЕ ПРОГРАММЫ	95

РУКОВОДСТВО ОПЕРАТОРА	97
СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ	103

1 ПОСТАНОВКА И АНАЛИЗ ЗАДАЧИ

1.1 Анализ предметной области

В наше время информационных технологий отмечается внушительный рост интереса к сфере IT со стороны трудовых ресурсов. Большое количество желающих начать карьеру в этой сфере создает конкурентное окружение, в котором отличие и выделение требуют особых усилий и навыков от начинающих специалистов. Это обусловлено не только привлекательностью высоких заработных плат, но и осознанием растущего спроса на квалифицированных IT-профессионалов.

С ростом числа работников в IT-сфере увеличиваются и требования, предъявляемые к начинающим специалистам. Компании ожидают не только теоретических знаний, но и умения применять их на практике. Это обусловлено динамикой индустрии, постоянными технологическими изменениями и потребностью в сотрудниках, способных быстро адаптироваться к новым вызовам.

Одним из важных аспектов в арсенале начинающего IT-специалиста является умение эффективно решать алгоритмические задачи. В современной практике собеседований в технологических компаниях, классические алгоритмические задачи стали непременным этапом отбора. Подобные задания не только оценивают знание базовых алгоритмов и структур данных, но и позволяют судить о широте мышления и готовности к решению реальных проблем.

Решение алгоритмических задач представляет собой не просто формальное испытание, а возможность показать свою способность анализа, творческого мышления и глубины понимания основных принципов программирования. В этом контексте, алгоритмическое программирование становится неотъемлемой частью профессионального роста и успешного вступления в индустрию.

Сложность и разнообразие алгоритмических задач позволяют оценить не только техническую подготовку, но и гибкость мышления, стремление к

самосовершенствованию и умение быстро принимать решения в условиях ограниченного времени. Эти качества являются неотъемлемой частью успешной карьеры в области информационных технологий.

Постоянная тренировка и развитие в области алгоритмического программирования становятся неременной частью жизни IT-специалиста. Систематическое решение алгоритмических задач не только поддерживает высокий уровень технических компетенций, но и обеспечивает гибкость в принятии решений в условиях быстро меняющейся индустрии.

Для упрощения и более объективного отбора специалистов в сфере информационных технологий, многие компании предпочитают использовать различные форматы конкурсов и контестов, в рамках которых участники решают сложные алгоритмические задачи. Этот метод позволяет выделить наиболее перспективных кандидатов, предоставляя компаниям возможность более тщательно оценить не только технические навыки, но и креативность, способность к коллаборации и быстрое принятие решений.

Контесты по алгоритмическому программированию становятся своеобразным полигоном, где участники могут продемонстрировать себя и выделиться на фоне конкуренции. Сложность задач, предлагаемых в таких мероприятиях, поднимается настолько высоко, что только настоящие мастера своего дела могут успешно справиться.

Проведение контестов также подчеркивает важность не только знания алгоритмов, но и умение применять их в условиях ограниченного времени и неопределенности. Соревновательный характер задач позволяет выявить не только теоретическую готовность, но и способность к принятию быстрых и обоснованных решений в динамичной среде.

Такие мероприятия не только служат эффективным инструментом отбора талантливых специалистов, но и мотивируют кандидатов на саморазвитие и постоянное совершенствование. Успех в контесте становится наглядным доказательством высокого уровня компетенций и готовности к решению сложных задач.

1.2 Обоснование актуальности работы

В условиях стремительного развития информационных технологий и увеличивающейся конкуренции на рынке труда в сфере ИТ, актуальность разработки платформы для решения алгоритмических задач и автоматизированного тестирования кандидатов становится неоспоримой. С постоянным увеличением числа желающих присоединиться к этой перспективной области профессиональной деятельности, компании сталкиваются с необходимостью эффективного и объективного отбора кандидатов среди большого числа желающих.

В наше время успешный старт в карьере ИТ-специалиста требует не только теоретических знаний, но и демонстрации практических навыков, способности эффективно решать сложные алгоритмические задачи. Существующая практика собеседований в технологических компаниях подчеркивает важность алгоритмических задач в процессе отбора кандидатов. Это отражает реальные потребности индустрии и подчеркивает необходимость создания инструмента, который поможет компаниям более эффективно оценивать навыки алгоритмического программирования у потенциальных сотрудников.

Кроме того, важным аспектом в обосновании актуальности является удобство тренировки. Платформа, предназначенная для решения алгоритмических задач, может предложить пользователям самые актуальные задачи, доступные для решения из любой точки мира с использованием WEB-браузера с выполнением кода и проверки решения на стороне сервера.

Наконец, следует подчеркнуть, что постоянная практика в решении алгоритмических задач не только развивает умение быстро и эффективно находить оптимальные решения, но также формирует аналитическое мышление и готовность к промышленной разработке. Практическое владение алгоритмами и структурами данных позволяет использовать языки программирования и структуры более эффективно; библиотеки и

фреймворки не как зависимости, задающие весь процесс разработки, а как эффективные инструменты для решения сложных задач в реальных проектах.

1.2.1 Существующие решения

Существующие решения в области онлайн-платформ для решения алгоритмических задач и автоматизированного тестирования кандидатов представляют собой важный контекст для разработки новой платформы.

В данном разделе будет проведен детальный анализ ряда популярных онлайн-платформ, специализирующихся на решении алгоритмических задач. Этот обзор позволит выявить основные характеристики, функциональные возможности и особенности существующих решений, а также проанализировать их применимость и основные недостатки.

1) LeetCode

Основная функциональность LeetCode:

LeetCode — платформа, акцентирующая внимание на подготовке к техническим собеседованиям и предоставляющая обширную коллекцию алгоритмических задач. Ее функциональные возможности включают:

- обширный набор задач: платформа предлагает разнообразные задачи, охватывающие различные аспекты программирования и алгоритмов;
- система обсуждения: каждая задача снабжена системой комментариев, что позволяет пользователям обсуждать решения, давать советы и обмениваться опытом;
- решения от компаний: LeetCode предлагает задачи, разработанные или используемые компаниями при технических собеседованиях.
- различные курсы для обучения решению алгоритмических задач;

Почему LeetCode не подходит для наших целей:

- иностранная платформа: LeetCode преимущественно ориентирована на англоязычную аудиторию и базируется на английском языке. Это может создать языковые и культурные барьеры для ряда

пользователей, особенно в государственных предприятиях, где использование русскоязычных ресурсов предпочтительно;

- ограниченный доступ для государственных предприятий: в связи с ограничениями в использовании внешних иностранных платформ государственными предприятиями, LeetCode может оказаться недоступной для значительной части целевой аудитории;

- отсутствие возможности создавать компаниями закрытые соревнования по решению задач.

2) Codewars

Основная функциональность Codewars

Codewars – платформа, направленная на развитие и совершенствование навыков программирования через решение задач. Ее функциональные возможности включают:

- обширный набор задач различной сложности на множестве языков программирования;

- платформа акцентирует внимание на соревновательном процессе и позволяет участникам соревноваться между собой в решении задач;

- каждая задача снабжена системой комментариев, что создает пространство для обсуждения различных подходов к решению. Пользователи могут делиться своим опытом, предлагать советы и взаимодействовать с сообществом;

- помимо самих задач, Codewars предоставляет возможность написания тестового кода на популярном фреймворке для каждого языка, что делает систему тестирования более прозрачной.

Однако, Codewars имеет схожие с Leetcode недостатки, связанные с доступом в государственных и не только предприятиях. Так же данная платформа обладает следующими недостатками:

- перегруженный интерфейс, непонятный человеку, только что зашедшему на платформу;

- отсутствие возможности создавать компаниями закрытые соревнования по решению задач.

3) Codeforces:

Codeforces - популярная онлайн-платформа по программированию и соревнованиям в области информационных технологий. Ее функциональные возможности включают:

- большая коллекция задач: Codeforces предлагает обширный набор алгоритмических задач, от простых до сложных, включая различные категории, такие как графы, динамическое программирование, жадные алгоритмы и т. д;

- система соревнований: платформа позволяет пользователям организовывать и участвовать в соревнованиях по программированию. Участники могут соревноваться друг с другом, решая задачи в заданное время, и сражаться за рейтинговые позиции;

- система рейтинга: платформа имеет систему рейтинга, которая формируется при решении задач и участии в соревнованиях;

- создание соревнований: Codeforces предоставляет возможность организации соревнований, однако это недоступно рядовому пользователю;

- система оценивания: Codeforces имеет продуманную систему оценки решения по занимаемой памяти и времени выполнения программы.

Платформа имеет следующие недостатки:

- в задачах необходимо реализовывать ввод и вывод данных, что может отвлекать от процесса решения;

- устаревший и неудобный интерфейс.

4) Яндекс Контест

Яндекс Контест - платформа для онлайн-проверки заданий, позволяющая проводить состязания любого уровня сложности, от школьных олимпиад — и до соревнований международного класса. Позволяет устраивать как командные, так и личные соревнования. На ее базе также

проходят тренировки спортивных программистов и ежегодный чемпионат «Яндекса» по разработке решений. Поддерживает больше двадцати языков программирования. Ее главные преимущества:

- встроенный редактор кода;
- возможность самому создавать соревнования через специальную страницу;
- документация по созданию соревнований и задач;
- известность.

Несмотря на свою популярность, данная платформа не совсем подходит для наших целей в связи с следующими недостатками:

- усложненный процесс создания задач;
- в задачах необходимо реализовывать чтение из консоли и вывод в консоль, что не относится к самому алгоритму решения задачи.

1.2.2 Сравнительная характеристика

В таблице 1 представлена сравнительная характеристика существующих решений, а также желаемые характеристики платформы, предназначенной для тренировки в решении алгоритмических задач и автоматизированного тестирования кандидатов.

Таблица 1 – Сравнительная характеристика существующих решений в рамках задачи тренировки в решении алгоритмических задач и автоматизированного тестирования кандидатов

Платформа	Русский язык	Для решения только реализовать метод	Удобный интерфейс	Быстрая обратная связь	Быстрое создание соревнований	Быстрое создание задач
LeetCode	Нет	Да	Да	Да	Нет	Нет
CodeWars	Нет	Да	Нет	Да	Да	Нет
Codeforces	Да	Нет	Нет	Нет	Нет	Нет
Яндекс Контест	Да	Нет	Да	Нет	Частично	Частично
Желаемое	Да	Да	Да	Да	Да	Да

1.3 Постановка задачи

Исходя из сравнительной характеристики существующих решений, становится ясной задача создания платформы, спроектированной для улучшения процесса тренировки в решении алгоритмических задач и тестирования кандидатов.

Важная особенность будущей платформы — возможность пользователям легко формировать задачи и контесты, но основная цель заключается в предоставлении пользователям простого и эффективного инструмента для самостоятельного совершенствования навыков путем решения задач. Платформа призвана быть не только средством отбора кандидатов, но и пространством для обучения и тренировки. Главные задачи платформы включают:

- упрощенный процесс решения задач: участники контестов и просто желающие потренироваться в решении задач должны иметь возможность решать задачи, сводящиеся к реализации одной функции, без необходимости использования консольного ввода/вывода и приведения типов;
- автоматическая проверка правильности решения: процесс проверки решения пользователя должен быть полностью автоматическим и простым для ускорения обратной связи с пользователем;
- прозрачность и простота автоматического тестирования: процесс тестирования должен быть максимально понятен для пользователей, обеспечивая им возможность легко понимать причины возможных ошибок. Так же это упростит процесс добавления данных для тестирования к задаче;
- возможность создания задач и контестов: платформа должна предоставлять заинтересованным лицам удобный интерфейс для создания контестов и задач;
- оценка решения пользователей для рекомендации компаниям: поскольку процесс проверки решения не направлен на оценку решения по

времени выполнения, а проверяется только правильность решения, платформа не должна предлагать конкретные оценки и рекомендации компаниям по рассмотрению работ конкретных пользователей. Объективной оценкой, которую может предоставить платформа является количество решенных задач;

- различные форматы контестов: платформа должна поддерживать как открытые, так и закрытые форматы проведения контестов;

- организация удобного поиска задач: платформа должна облегчить пользователю процесс выбора задачи поиском по названию, сортировкой по сложности.

1.4 Выбор языков программирования и технологий

При принятии решений относительно технологий для разработки платформы в первую очередь было решено, что взаимодействие с платформой будет осуществляться через веб-приложение. Это решение обусловлено не только огромной популярностью, но и доступностью с любого устройства, что является важным фактором.

В качестве языка программирования для написания серверной части платформы, было решено использовать Kotlin [6]. Этот выбор обусловлен стремительным ростом популярности этого языка в сфере разработки, превращая его не только в основной инструмент для создания приложений под Android, но и в востребованную замену Java в промышленных приложениях. Kotlin предоставляет удобство и выразительность, а также отлично интегрируется с существующим Java-кодом, что является значимым преимуществом. График востребованности Kotlin представлен на рисунке 1.1.

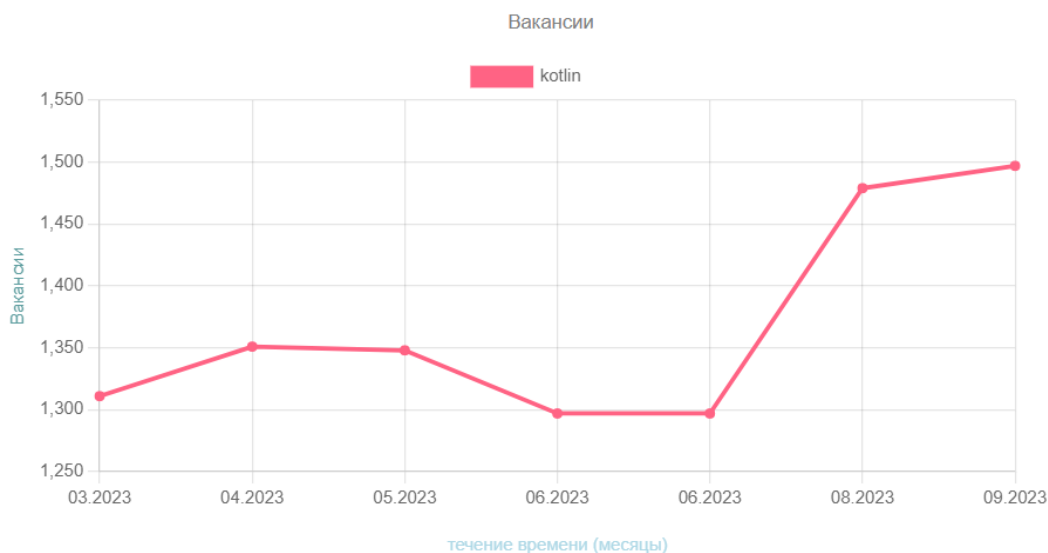


Рисунок 1.1 – Востребованность Kotlin на 09.2023

При разработке клиентской части веб-платформы было решено использовать Vue.js [29]. Этот современный JavaScript-фреймворк известен своей простотой и гибкостью, что делает его идеальным выбором для создания динамичных пользовательских интерфейсов. Vue.js пользуется широкой популярностью в сообществе разработчиков благодаря своей легкости в освоении и прекрасной документации. График востребованности Vue представлен на рисунке 1.2.

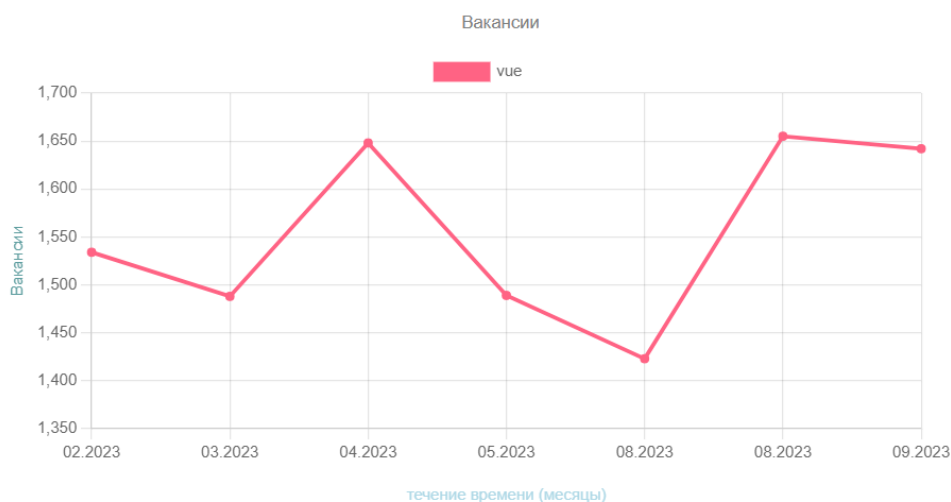


Рисунок 1.2 – Востребованность Vue на 09.2023

В качестве фреймворка для разработки серверной части будет использоваться Kora.

Kora [7] предоставляет все необходимые для современной Java / Kotlin разработки инструменты:

- внедрение и инверсию зависимостей на этапе компиляции посредством аннотаций;
- аспектно-ориентированное программирование посредством аннотаций;
- пре-конфигурируемые модули интеграций;
- легкое и быстрое тестирование с помощью JUnit5;
- простая, понятная и подробная документация, подкреплённая примерами сервисов.

Для достижения высокопроизводительного и эффективного кода, Kora стоит на таких принципах:

- использование обработчиков аннотаций;
- отказ от использования Reflection API;
- отказ от динамических прокси;
- реализацию тонких абстракций;
- реализацию бесплатных аспектов через наследование классов;
- использование наиболее эффективных реализаций для модулей;
- реализацию и предоставление наиболее эффективных принципов для разработки;
- отсутствие генерации байт-кода во время компиляции и исполнения.

Таким образом Kora отлично подходит для разработки высоконагруженных приложений с быстрым запуском для эффективного масштабирования.

Для управления инфраструктурой будет использоваться оркестратор контейнеров Kubernetes.

Kubernetes [21] - это открытое программное обеспечение для автоматизации развёртывания, масштабирования и управления контейнеризированными приложениями. Kubernetes управляет и запускает

контейнеры Docker на большом количестве хостов, а также обеспечивает совместное размещение и репликацию большого количества контейнеров. Проект был начат Google и теперь поддерживается многими компаниями, среди которых Microsoft, RedHat, IBM и Docker.

Основные возможности Kubernetes:

- мониторинг сервисов и распределение нагрузки: Kubernetes может обнаружить контейнер, используя имя DNS или IP-адрес. Также Kubernetes может сбалансировать нагрузку и распределить сетевой трафик, чтобы развертывание было более стабильно;
- оркестрация хранилища: Kubernetes позволяет автоматически смонтировать нужную систему хранения, например, локальное хранилище, провайдера облака и многое другое.
- автоматическое развертывание и откаты: Используя Kubernetes, можно менять фактическое состояние развернутых контейнеров, а также описывать желаемое состояние новых контейнеров.
- автоматическое распределение нагрузки: На основе предоставленных для Kubernetes кластеров, узлов, заданных данных по CPU и памяти для каждого контейнера, Kubernetes может разместить их на узлах так, чтобы наиболее эффективно использовать ресурсы.
- самоконтроль: Kubernetes автоматически перезапускает сбойные контейнеры. При необходимости он заменяет и завершает работу контейнеров, которые не проходят определенную пользователем проверку работоспособности и отключает доступ к ним, пока они не будут готовы к обслуживанию.

2 ПРОЕКТИРОВАНИЕ ПЛАТФОРМЫ

2.1 Проектирование процесса проверки решения на серверной части

Проверка решения задачи, отправленного пользователем, является ключевой функциональностью будущей платформы. Рассмотрим этот процесс подробнее:

1. Пользователь, взаимодействуя с клиентским веб-приложением пишет код на выбранном языке, после чего отправляет решение.
2. На сервис API приходит код пользователя, язык, на котором написан код и идентификатор задачи, по которому можно получить задачу из хранилища.
3. API получает драйверы и тесты из базы данных по ID задачи. Драйвер – код, который создается для каждого языка при создании задачи и служит основой для выполнения кода пользователя. Он состоит из методов для чтения из консоли, цикла, который читает тестовые данные и прогоняет их через решение пользователя, а также метода, который выводит результат в консоль. Код для выполнения получается путем подстановки кода пользователя в нужное место драйвера.
4. В базу данных сохраняется попытка без результата и получается ее ID.
5. В сервис executor асинхронно отправляется ID решения и код готовый к выполнению.
6. Executor компилирует код, если необходимо и запускает процесс с выполнением полученного кода. Взаимодействуя с процессом через консоль, сервис отправляет тестовые данные через консоль и считывает результаты. Полученные результаты сравниваются с ожидаемыми и асинхронно отправляется результат решения на сервис API.
7. API получает результат решения и обновляет его в базе данных.
8. Результат возвращается для отображения пользователю.

Диаграмма последовательности показана на рисунке 2.1

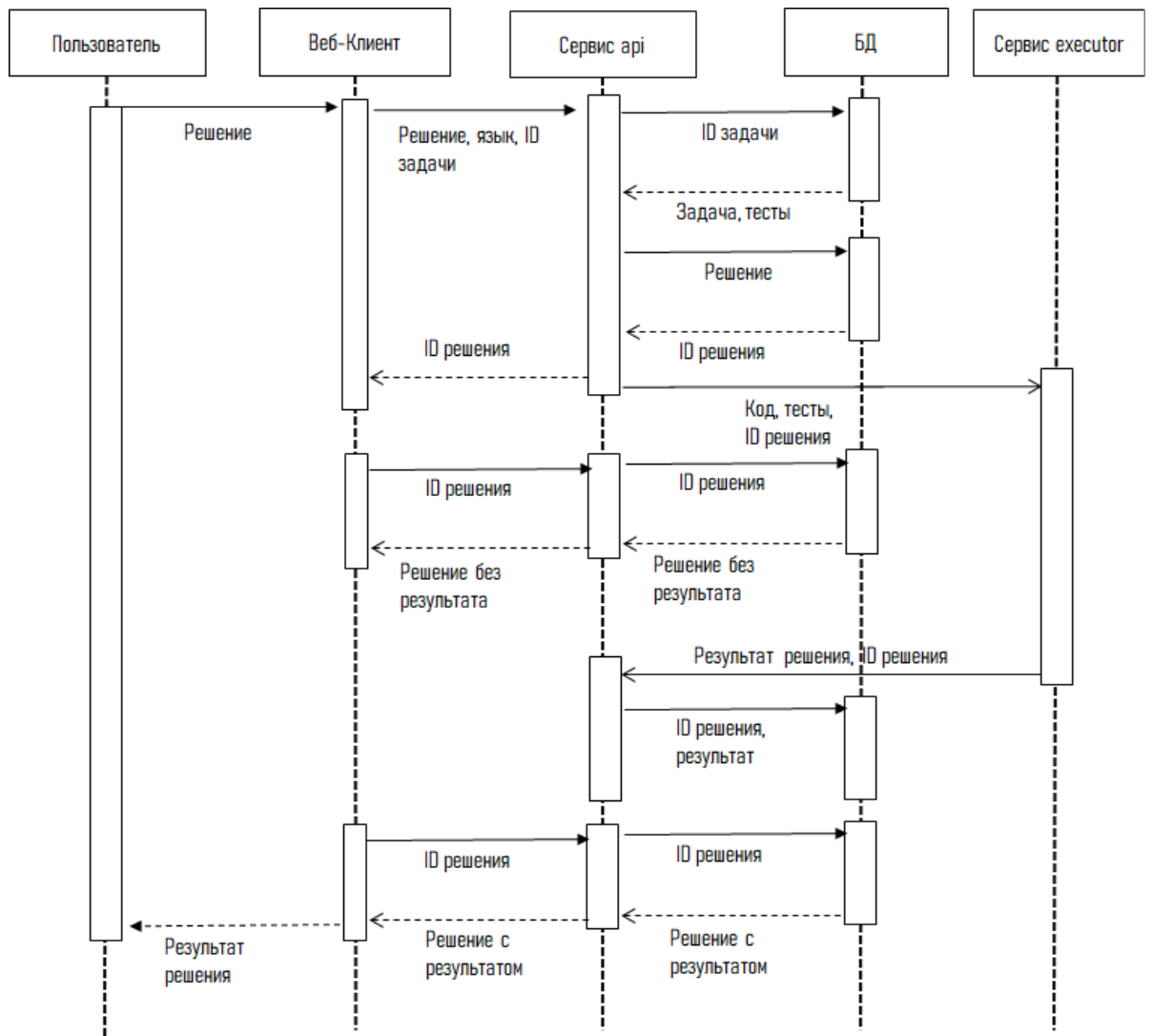


Рисунок 2.1 – Диаграмма последовательности для проверки решения

2.2 Инфологическое проектирование Базы Данных

В предметной области можно выделить следующие сущности:

- 1) Пользователь
- 2) Задача
- 3) Решение
- 4) Тест
- 5) Соревнование

В предметной области можно выделить следующие связи между сущностями:

- 1) Пользователь создает Решение
- 2) Решение решает Задачу

- 3) Решение учитывается в Соревновании
- 4) Задача содержит Тест
- 5) Задача включена в Соревнование
- 6) Пользователь организует Соревнование
- 7) Пользователь участвует в Соревновании
- 8) Пользователь создает задачу

Построение ER-Диаграмм

1. Связь Пользователь создает Решение

Для степени связи:

- Пользователь создает множество решений;
- Решение создано только одним пользователем;

Для класса принадлежности сущности к связи:

- Пользователь необязательно создает Решение;
- Решение обязательно создано пользователем;



Рисунок 2.2 – ER-Диаграмма связи Пользователь создает Решение

2. Связь Решение решает Задачу

Для степени связи:

- Задача имеет множество решений;
- Решение решает только одну задачу;

Для класса принадлежности сущности к связи:

- Задача необязательно имеет Решение;
- Решение обязательно решает Задачу;



Рисунок 2.3 – ER-Диаграмма связи Решение решает Задачу

3. Связь Решение учитывается в Соревновании

Для степени связи:

- Соревнование учитывает множество Решений;
 - Решение учитывается только в одном соревновании;
- Для класса принадлежности сущности к связи:
- Соревнование обязательно учитывает Решение;
 - Решение обязательно учитывается в Соревновании;



Рисунок 2.4 – ER-Диаграмма связи Решение решает Задачу

4. Связь Задача содержит Тест

Для степени связи:

- Задача содержит множество Тестов;
- Тест привязан к одной Задаче;

Для класса принадлежности сущности к связи:

- Задача обязательно содержит Тест;
- Тест обязательно привязан к Задаче;



Рисунок 2.5 – ER-Диаграмма связи Задача содержит Тест

5. Связь Задача включена в Соревнование

Для степени связи:

- Задача включена в множество Соревнований;
- Соревнование содержит множество Задач;

Для класса принадлежности сущности к связи:

- Задача обязательно включена Соревнование;
- Соревнование обязательно содержит Задачи;



Рисунок 2.6 – ER-Диаграмма связи Задача включена в Соревнование

6. Связь Пользователь организует Соревнование

Для степени связи:

- Пользователь организует множество Соревнований;
- Соревнование организуется одним Пользователем;

Для класса принадлежности сущности к связи:

- Пользователь необязательно организует Соревнование;
- Соревнование обязательно организуется Пользователем;



Рисунок 2.7 – ER-Диаграмма связи Пользователь организует Соревнование

7. Связь Пользователь участвует в Соревновании

Для степени связи:

- Пользователь участвует в множестве Соревнований;
- В Соревновании участвуют множество Пользователей;

Для класса принадлежности сущности к связи:

- Пользователь необязательно участвует в Соревновании;
- В Соревновании необязательно участвуют Пользователи;



Рисунок 2.8 – ER-Диаграмма связи Пользователь участвует в Соревновании

8. Связь Пользователь создает Задачу

Для степени связи:

- Пользователь создает множество Задач;
- Задачу создает один Пользователь;

Для класса принадлежности сущности к связи:

- Пользователь необязательно создает Задачу;
- Задачу обязательно создает Пользователь;

3) Решение учитывается в Соревновании:

- Соревнование (competition_id)
- Решение (attempt_id, competition_id)

4) Задача содержит Тест

- Задача (task_id)
- Тест (test_id, task_id)

5) Задача включена в Соревнование

- Задача (task_id)
- Соревнование (competition_id)
- Задача_Соревнование (task_id, competition_id)

6) Пользователь организует Соревнование

- Пользователь (user_id)
- Соревнование (competition_id, user_id)

7) Пользователь участвует в Соревновании

- Пользователь (user_id)
- Соревнование (competition_id)
- Пользователь_Соревнование (user_id, competition_id)

8) Пользователь создает Задачу

- Пользователь (user_id)
- Задача(task_id, user_id)

2.3.2 Подготовка списка атрибутов. Распределение их по отношениям

1) Пользователь (user) – user_id, username, password, created_at

2) Задача (task) - task_id, name, description, input_types, output_type, languages, drivers, is_enabled, method_name, is_private, level, created_at, user_id

3) Тест (test) - test_id, task_id, input_data, output_data

4) Решение (attempt) – attempt_id, user_id, task_id, competition_id, status, code, language, error, created_at

5) Соревнование (competition) – competition_id, title, description, start_at, created_at, is_private, user_id, end_at

8) Задача_Соревнование (task_competition) - task_id, competition_id, points

9) Пользователь_Соревнование (user_competition) – user_id, competition_id, started_at, ended_at

2.4 Проектирование архитектуры платформы.

Обобщенно архитектуру платформы можно представить, как клиент-серверное приложение. Клиент-серверная архитектура – это подход, в котором функциональность приложения разделена между клиентской (пользовательской) и серверной (бэкенд) частями. Клиент отправляет запросы серверу, а сервер обрабатывает эти запросы и возвращает результат клиенту. Общая схема архитектуры представлена на рисунке 2.10.

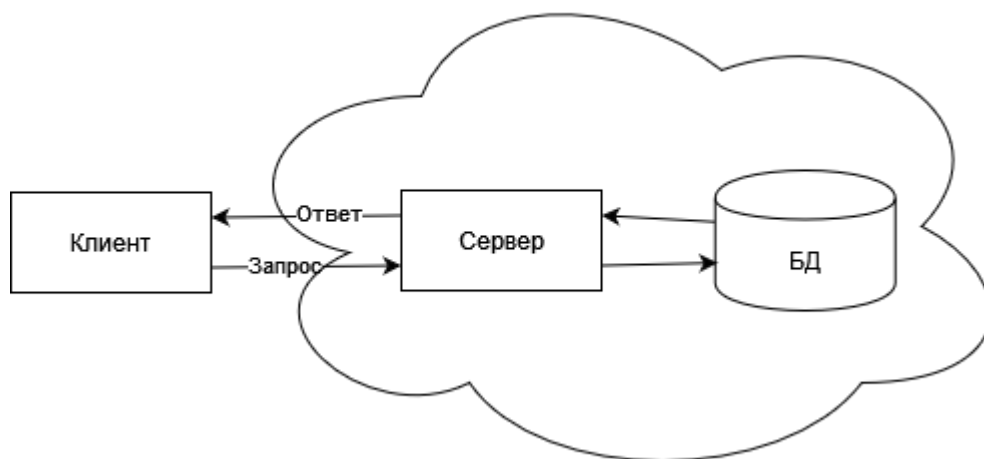


Рисунок 2.10 – Клиент-серверная архитектура

На клиент-серверной архитектуре построены все сайты и интернет-сервисы. Также ее используют десктоп-программы, которые передают данные по интернету.

Преимущества клиент-серверной архитектуры:

- масштабируемость: клиент-серверная архитектура позволяет распределить нагрузку на сервера и может масштабироваться по мере необходимости. Благодаря этому можно значительно улучшить

производительность системы и обрабатывать большое количество запросов от клиентов;

- централизованное управление: сервер является центральным узлом, который контролирует всю систему, обеспечивает безопасность и управление доступом к данным. Это позволяет легко обновлять и модифицировать систему, не задевая клиента;

- безопасность: централизованное управление сервером обеспечивает возможность контроля доступа и защиты данных.

Недостатки клиент-серверной архитектуры:

- зависимость от сервера: клиент не может работать без сервера. Если сервер(а) недоступен или имеет проблемы, все клиенты будут неработоспособны или испытывать проблемы с функциональностью;

- затраты на инфраструктуру: клиент-серверная архитектура требует наличия серверного и сетевого оборудования и поддержки, что может потребовать затрат на инфраструктуру и обслуживание;

- зависимость от сети: клиент-серверная архитектура требует постоянного подключения к сети. Если сеть недоступна или имеет проблемы, это может существенно ограничить возможности работы системы;

- ограниченность: при использовании клиент-серверной архитектуры возникают ограничения на количество одновременно подключенных клиентов и на пропускную способность сети. Это может привести к ограничениям в расширении системы и обработке большого количества запросов.

Платформа будет представлять собой интернет сервис, что позволит иметь доступ к функционалу с любого устройства, имеющего выход в интернет. При таком подходе взаимодействие между клиентом и сервером осуществляется через протокол HTTP.

HTTP (Hypertext Transfer Protocol) – это протокол прикладного уровня, используемый для передачи данных по сети. Основан на взаимодействии запрос – ответ. HTTP является основным строительным блоком веб-

взаимодействия и используется во многих приложениях для передачи данных между клиентами и серверами.

2.4.1 Архитектура клиента

Клиент будет представлять собой SPA (Single Page Application) - это тип веб-приложения, которое загружает единственную веб-страницу и динамически обновляет ее, вместо того чтобы загружать новые страницы с сервера. Это позволяет создавать более интерактивные и быстрые веб-приложения, так как большая часть ресурсов загружается один раз, а затем переиспользуется без полной перезагрузки страницы. То есть при переходе по URL адресу платформы, браузер будет делать HTTP запрос на web сервер Nginx, который вернет один HTML файл и ссылки на CSS и JavaScript файл, который является основным для клиентского приложения. Для упрощения разработки SPA используется фреймворк Vue.

2.4.2 Архитектура сервера

Существуют два основных подхода к архитектуре серверной части: монолитная архитектура и микросервисная архитектура.

Монолитная архитектура — это методология проектирования и построения приложений, при которой весь функционал приложения организован и интегрирован в одну программу или исполняемый модуль. В отличие от более распределенных подходов, таких как микросервисная архитектура, монолит объединяет все компоненты приложения в одной кодовой базе и обычно запускается на одном сервере.

Вот основные характеристики монолитной архитектуры:

1. Единая кодовая база: весь исходный код приложения находится в одном репозитории и компилируется в один исполняемый файл или пакет.
2. Одна база данных: обычно монолит использует одну базу данных для хранения данных приложения.
3. Монолитное развертывание: приложение развертывается как единое целое. Все изменения вносятся и разворачиваются вместе.

4. Единый язык программирования: в монолитных приложениях используется один язык программирования и технологический стек для всех компонентов.

5. Производительность: из-за отсутствия необходимости сетевого взаимодействия между компонентами, монолитные приложения могут быть более производительными внутри.

Преимущества монолитной архитектуры:

- простота разработки и тестирования: отладка и тестирование приложения проще, поскольку все компоненты находятся в одном месте;
- производительность при малых объемах: внутренние вызовы происходят в пределах одного процесса, что обычно более эффективно с точки зрения производительности;
- проще масштабирование на начальных этапах: на ранних этапах развития приложения простота масштабирования может быть важнее, чем более сложные модели. Достаточно запустить несколько экземпляров за балансировщиком.

Недостатки монолитной архитектуры:

- сложность масштабирования на более поздних этапах: при увеличении объема функционала и нагрузки масштабирование становится сложнее;
- сложность поддержки: изменения в одном компоненте могут затрагивать другие, что усложняет поддержку при росте приложения;
- единовременное развертывание: необходимость разворачивания всего приложения целиком может быть проблемой при внесении изменений.

Монолитная архитектура обычно выбирается на начальных этапах разработки, когда требования к масштабированию еще не так велики, а простота и быстрота разработки являются приоритетом. Позже, при росте проекта и возрастании нагрузки, команды могут решить перейти к более современным моделям, таким как микросервисная архитектура. Пример приложения с монолитной архитектурой представлен на рисунке 2.11.

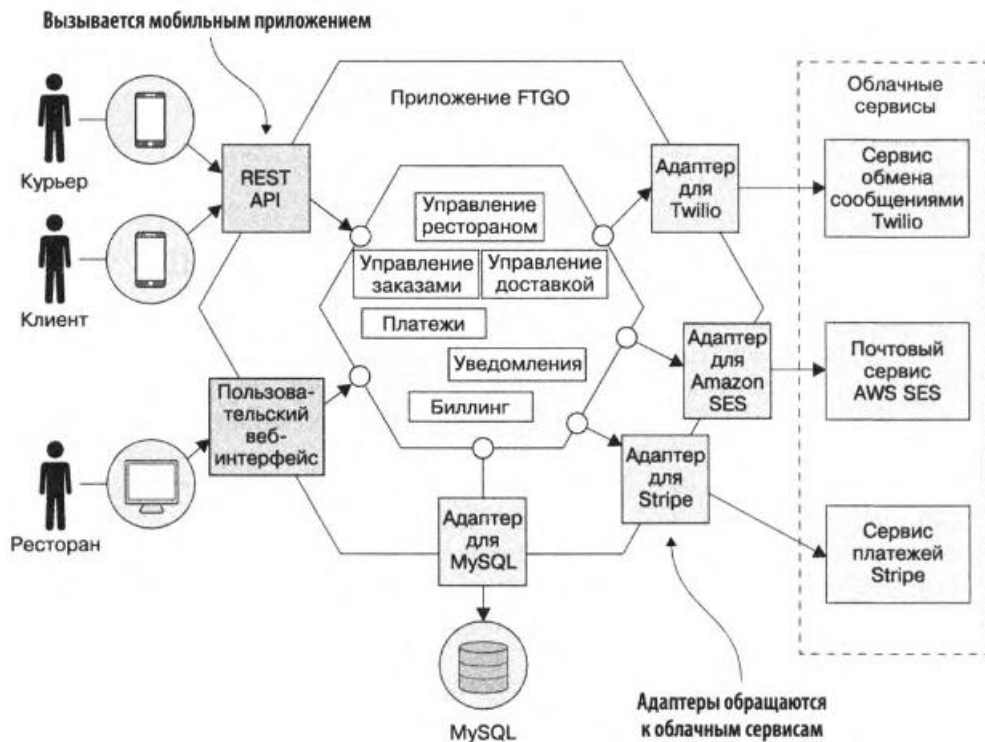


Рисунок 2.11 – Монолитное приложение

Микросервисная архитектура — это методология проектирования и построения приложений, при которой функционал разделяется на небольшие, автономные и взаимодействующие между собой сервисы. Каждый микросервис представляет собой отдельный компонент, обслуживающий конкретные бизнес-задачи [9, 19]. Вот основные характеристики микросервисной архитектуры:

1. Разделение на сервисы: функционал приложения разбивается на небольшие сервисы, каждый из которых отвечает за конкретный аспект приложения.
2. Независимость сервисов: каждый микросервис может быть разработан, развернут и масштабирован независимо от других. Это обеспечивает гибкость и ускоряет процесс разработки и внесения изменений.
3. Распределенная архитектура: сервисы взаимодействуют друг с другом посредством API, обеспечивая распределенную архитектуру. Это позволяет создавать гибкие и масштабируемые системы.

4. Самостоятельное развертывание: микросервисы могут быть развернуты отдельно, что позволяет внедрять изменения и обновления без остановки всего приложения.

5. Многоязычное программирование: различные микросервисы могут быть написаны на разных языках программирования и использовать различные технологические стеки в зависимости от их уникальных требований.

Преимущества микросервисной архитектуры:

- гибкость и масштабируемость: микросервисы могут быть масштабированы и обновлены независимо друг от друга, что обеспечивает большую гибкость в управлении системой;
- легкость разработки несколькими командами: разработчики могут работать над отдельными микросервисами параллельно, что ускоряет процесс разработки;
- улучшенная изоляция и отказоустойчивость: ошибка в одном сервисе не влияет на работу других, что обеспечивает лучшую изоляцию и отказоустойчивость.

Недостатки микросервисной архитектуры:

- сложность управления: управление большим количеством микросервисов требует более сложной инфраструктуры и средств управления;
- распределенная архитектура: взаимодействие между сервисами может повлечь за собой проблемы сетевой задержки и сложности в отладке;
- высокие требования к инфраструктуре: создание и поддержка инфраструктуры для микросервисов может потребовать дополнительных усилий.

Микросервисная архитектура часто выбирается для крупных и сложных проектов, где требуется высокая гибкость, масштабируемость и возможность разработки в распределенной команде. Сейчас существует большое количество инструментов, позволяющие облегчить построение

приложений с использованием микросервисной архитектуры. На рисунке 2.12 представлено приложение из рисунка 2.11 переделанное с использованием микросервисов.

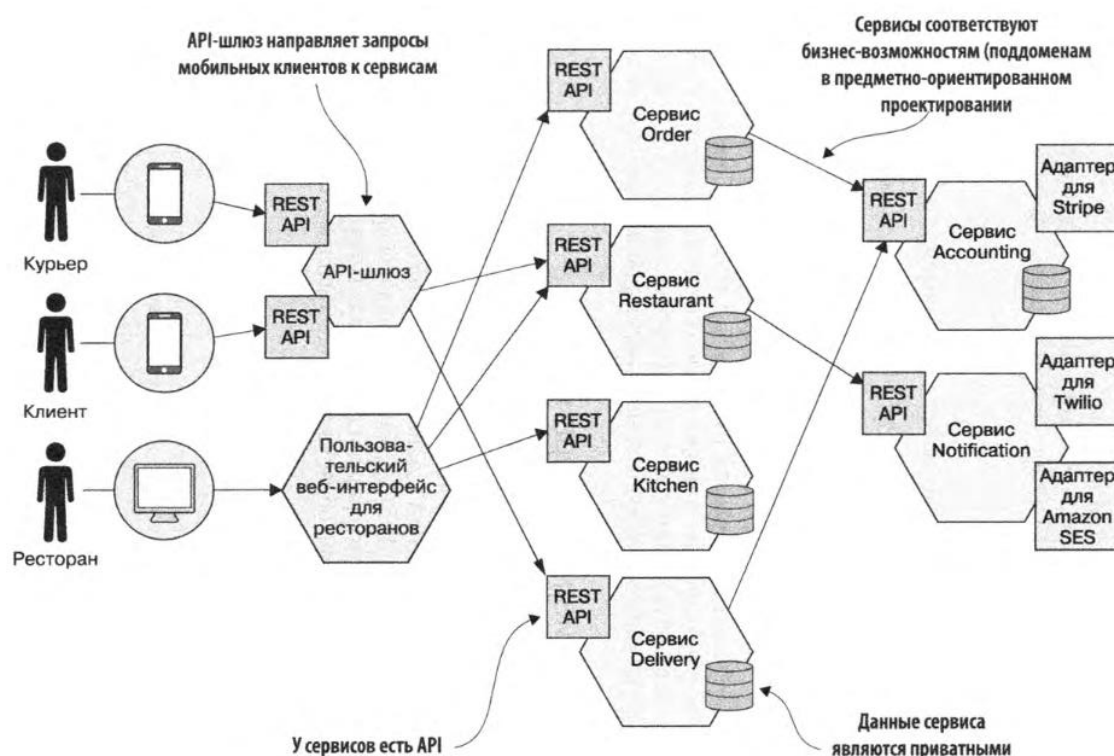


Рисунок 2.12 – Монолит, переделанный в микросервисы

В таблице 2 представлена сравнительная характеристика монолитной и микросервисной архитектуры.

Таблица 2 – Сравнительная характеристика монолитной и микросервисной архитектуры.

Характеристика	Монолитная архитектура	Микросервисная архитектура
Легкость масштабирования	-	+
Простота тестирования	+	-
Независимость логических модулей	-	+
Простота разработки	+	-
Простота инфраструктуры	+	-

2.4.3 Итоговая архитектура платформы

Исходя из рассмотренных преимуществ и недостатков архитектур, был сделан выбор в пользу микросервисной архитектуры. Это позволит легко

масштабировать платформу под большие нагрузки, а также весьма просто добавлять новый функционал не затрагивая старый. На рисунке 2.13 представлена архитектура платформы.

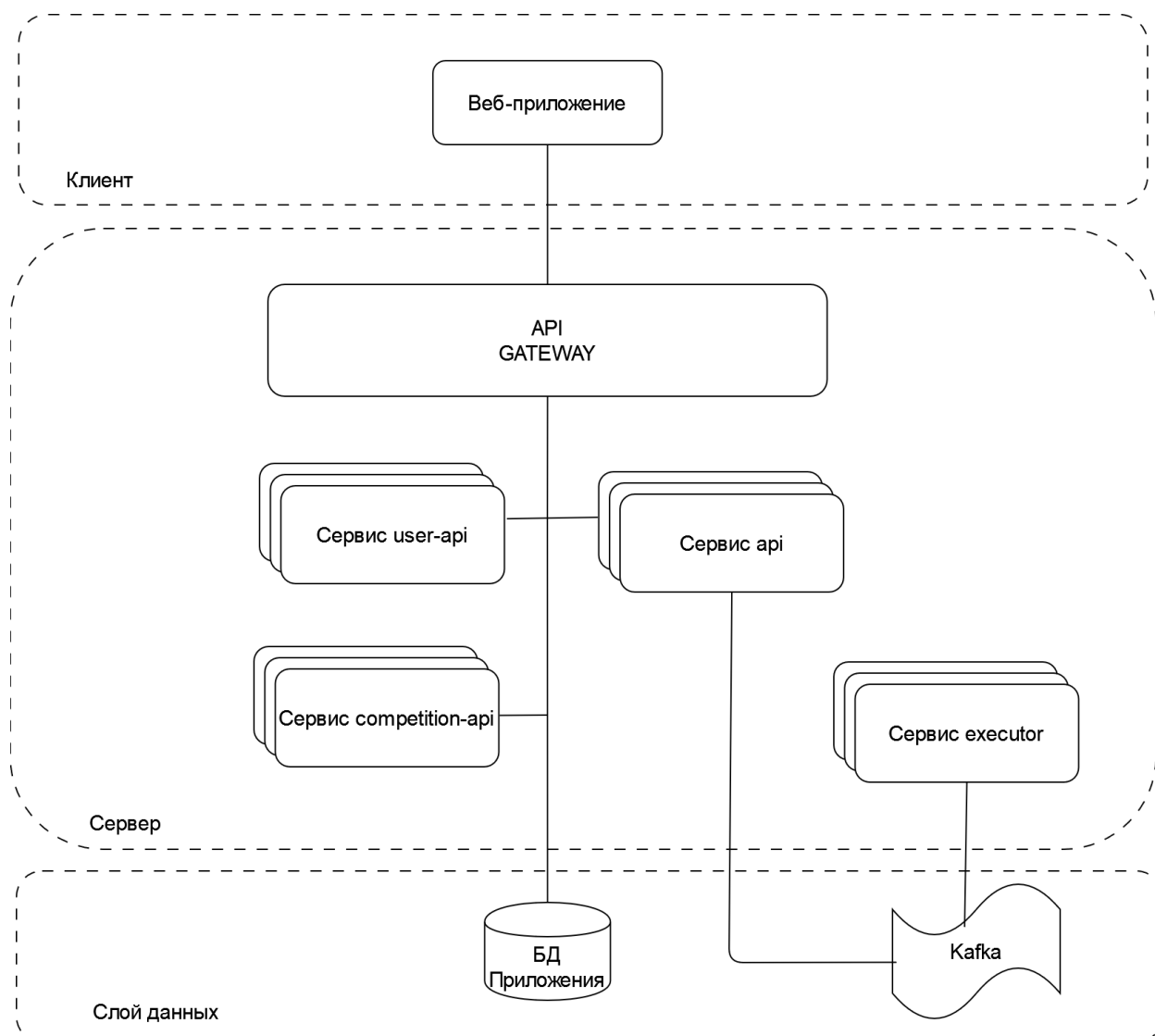


Рисунок 2.13 – Итоговая архитектура системы

Сервисы user, api, auth, competition-api, executor – это основные сервисы платформы, полученные методом разделения ответственности. В данном случае сервис представляет собой приложение на Kotlin. Все эти сервисы могут быть запущены в большом количестве экземпляров. Все сервисы, кроме executor взаимодействуют между собой по HTTP. Для взаимодействия с executor используется асинхронное взаимодействие при помощи Kafka. Это позволяет балансировать нагрузку, а также формировать очередь на выполнение кода.

Шлюз (API Gateway) – распределяет запросы между сервисами по url запроса и является единой точкой входа в серверную часть. В инфраструктуре шлюзом является Ingress Nginx контроллер Kubernetes, который в зависимости от префикса URL будет распределять запросы по разным VirtualService а ты в свою очередь распределяют нагрузку между подами.

3 РАЗРАБОТКА ПЛАТФОРМЫ

Серверная часть разрабатывается с применением архитектурного стиля REST (Representational State Transfer, передача состояния представления) — это архитектурный стиль взаимодействия компонентов распределённого приложения в сети [5].

Назначение REST состоит в том, чтобы придать проектируемой системе такие свойства, как:

- производительность;
- масштабируемость;
- гибкость к изменениям;
- отказоустойчивость;
- простота поддержки.

Одним из принципов REST, который помогает добиться выполнения этих свойств, является кеширование — сохранение ответа сервера, вместо повторного вычисления или получения данных и БД. Это значительно сокращает время ответа и нагрузку на железо.

Однако, в распределенных системах возникает проблема своевременной инвалидации кеша. В разрабатываемой платформе кеш используется, для получения задачи по id, решения по id и тестов для задачи. Для инвалидации используется оповещение о необходимости инвалидации при помощи события в Kafka. Так как используется кеш, хранящийся в памяти приложения, а экземпляров приложений может быть несколько, каждое приложение должно представлять собой отдельную группу-потребитель в соответствующем топике. Платформа использует библиотеку Caffeine, как реализацию in-memory кеша [23].

Сами сервисы будут построены по принципу contract-first. То есть, сначала описывается контракт, схемы данных при помощи спецификации open-api, а после чего из него генерируется код точек взаимодействия с

программой и классы ответов. Это упростит интеграцию платформы в сторонние системы, а также поможет формализовать этап разработки.

Так же для удобства разработки, тестирования и предотвращения ошибок в анализе все возможные операции над доменной областью описываются при помощи файлов формата Markdown еще до этапа разработки. После чего программа тестируется в соответствии с описанными алгоритмами.

3.1 Разработка сервиса выполнения кода и проверки решения

Сервис служит для запуска кода и проверки решения на правильность. Проверка осуществляется запуском подготовленного кода, отправкой данных через консоль, чтением результата и сравнением с ожидаемым результатом. Взаимодействие с сервисом происходит асинхронно через Kafka [22].

Топик для вызова операции выполнения кода – `codeest.runner.request`. Контракт топика представлен на рисунке 3.1.

RunCodeRequestEvent (codest.runner.request)

Описание

Название поля	Тип	Обязательность	Описание
code	String	+	Код, готовый к выполнению/компиляции
tests	ExecutionTestCase[]	-	Тесты
language	Language	+	Язык и версия, на котором запускается код

Language (Enum)

Значения

Название
JAVA
PYTHON

ExecutionTestCase (Enum)

Название поля	Тип	Обязательность	Описание
inputData	String[]	+	Входные данные тест-кейса
outputData	String	-	Ожидаемый результат тест-кейса

Рисунок 3.1– Контракт топика codest.runner.request

Топик для результата тестирования – codest.runner.response. Контракт топика представлен на рисунке 3.2.

RunCodeResponseEvent

Описание

Название поля	Тип	Обязательность	Описание
errorType	CodeRunnerErrorType	-	Информация об ошибке/ Непройдённый тесткейс
output	String[]	+	Вывод программы построчно в виде массива или сообщение о ошибке

CodeRunnerErrorType (Enum)

Значения

Название	Описание
COMPILE_ERROR	Ошибка компиляции
RUNTIME_ERROR	Ошибка выполнения
INTERNAL_ERROR	Ошибка сервиса
TIME_EXCEED_ERROR	Время выполнения превышено
TEST_ERROR	Программа не прошла тестирование

Рисунок 3.2 – Контракт топика `codeest.runner.response`

Алгоритм выполнения операции `RunCodeOperation` представлен на рисунках 3.3 – 3.4.

RunCodeOperation - запуск кода

Предназначена для компиляции, запуска кода и проверки решения.

Алгоритм

1. Прочитать из топика **codest.runner.request** и декодировать в соответствии с **контрактом**
2. Сохранить файл в временную папку
3. Если код необходимо скомпилировать:
 - i. Скомпилировать код при помощи **commandToCompile** из настроек для **language** из запроса. При ошибке компиляции:
 - а. Положить в топик **codest.runner.response** сообщение по **контракту** и **key = event.key**

Поле	Значение
errorType	COMPILE_ERROR
output	Сообщение из вывода компилятора

4. Выполнить код командой **commandToRun** из настроек для **language**
 - i. Каждый элемент из **input[]** из события отправить в стандартный поток ввода
 - ii. Стандартный поток вывода записать в **result[]** по строкам
 - iii. Поток ошибок записать в **errors**
5. Если **errors** не пустой:
 - i. Положить в топик **codest.runner.response** сообщение по **контракту** и **key = event.key**:

Поле	Значение
errorType	RUNTIME_ERROR
output	result + errors

Рисунок 3.3 – Алгоритм RunCodeOperation пункты 1-5

6. Иначе

- i. Сопоставить массив `result` и ожидаемые ответы из `request.tests`. Если для любого теста значения не совпадают:
 - а. Положить в топик `codest.runner.response` сообщение по контракту и `key = event.key`:

Поле	Значение
<code>errorType</code>	<code>TEST_ERROR</code>
<code>output</code>	<code>List.of(input, expected, actual)</code>

- ii. Иначе Положить в топик `codest.runner.response` сообщение по контракту и `key = event.key`:

Поле	Значение
<code>errorType</code>	<code>null</code>
<code>output</code>	<code>result</code>

7. Если 3 или 4 занимают более `maxTime` из настроек, то прервать выполнение и положить в топик * `codest.runner.response*` сообщение по контракту и `key = event.key`:

Поле	Значение
<code>errorType</code>	<code>TIME_EXCEED_ERROR</code>
<code>output</code>	"Время ожидания превышено"

В случае любой непредвиденной ошибки при выполнении одной задачи отменить ее выполнение и:

1. Залогировать ошибку `ParseRunnerRequestError`
2. Положить в топик `codest.runner.response` сообщение по контракту и `key = event.key`:

Поле	Значение
<code>errorType</code>	<code>INTERNAL_ERROR</code>
<code>output</code>	Сообщение из исключения

Рисунок 3.4 – Алгоритм `RunCodeOperation` пункты 6-8

Ключ события представляет собой `id` попытки. Это обеспечит попадание результата, на тот же экземпляр сервиса `api`, благодаря чему будет обеспечено 100% попадание в кеш решений. Оба топика сервиса содержат 10 разделов, что позволяет запустить до 10 экземпляров этого сервиса.

3.2 Разработка сервиса задач и решений

Сервис предоставляет API для создания задач, получения списка задач, получения конкретной задач, создания решений, получения списка решений пользователя, получение решения.

3.2.1 Получение задачи с решениями пользователя по ID задачи

Данный метод требует авторизации при помощи JWT токена. О реализации рассказано в главе 3.3.

Контракт представлен на рисунках 3.5 – 3.6.

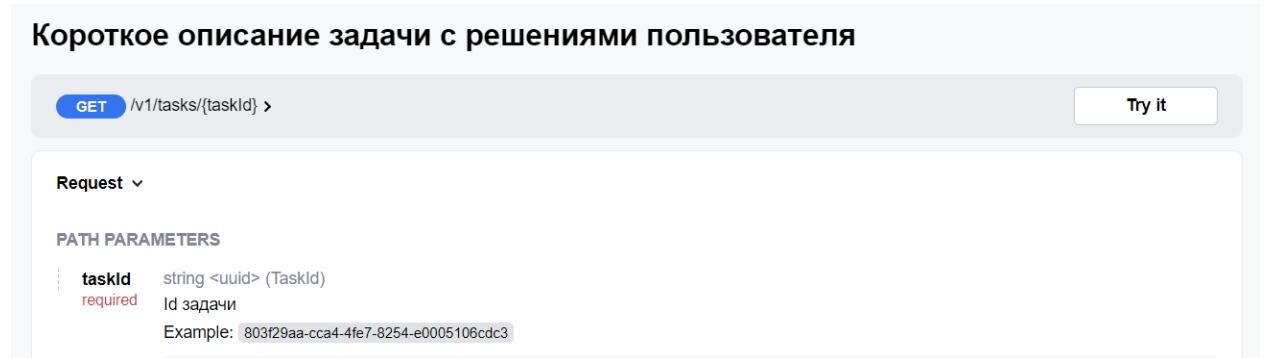


Рисунок 3.5 – Путь к вызову операции и входные параметры

id required	string <uuid> (TaskId) Уникальный идентификатор задачи
name required	string (TaskName) Имя задачи
level required	string (Level) Уровень сложности Enum: "easy" "medium" "hard"
description required	string Описание
isAuthor required	boolean Является ли пользователь автором задачи
languages required	Array of strings (TaskLanguages) Допустимые языки
isEnabled required	boolean (IsEnabled) Отображается ли задача в списке (есть доступ по id)
isPrivate required	boolean (IsPrivate) Отображается ли задача только в соревновании (есть доступ по id)
startCodes required	string (TaskStartCodes) Заготовки кода для пользователя. язык -> драйвер
solutions ▾ required	Array of objects (SolutionLiteResponse) Default: [] Решения для задачи этого пользователя (по умолчанию пустой)

Array [

id required	string <uuid> (SolutionId) Уникальный идентификатор решения
status required	string (SolutionStatus) Статус решения Enum: "pending" "accepted" "runtime_error" "compile_error" "test_error" "timeout_error" "internal_error"
createdAt required	string <date-time> (SolutionCreatedAt) Дата создания решения
language required	string (SolutionLanguage) Язык решения

]

Рисунок 3.6 – Схема результата

Алгоритм выполнения операции представлен на рисунке 3.7.

GetTaskLite - получение задачи для фронта

Параметры

Имя	Тип	Обязательность
taskId	path	+
userId	JWT	-

Алгоритм

1. Обогатится задачей из **tasks** (или в кеше)
 - i. Если не найдено, выбросить ошибку **TaskNotFound (422)**
2. Если **userId** передан обогатится в **attempts** по **taskId** и **userId**
3. Если **userId** из JWT передан и равен **userId** из задачи, то считать **attempts.isAuthor = true**
4. Вернуть ответ в соответствии мапингу из **openApi**

Рисунок 3.7 – Алгоритм

3.2.2 Получение списка задач доступных для решения

Контракт представлен на рисунках 3.8 – 3.9.

Список задач с пагинацией

GET /v1/tasks/list > Try it

Request ▾

QUERY PARAMETERS

offset

integer <int64> >= 0
Default: 0
Смещение страниц

limit

integer [1 .. 100]
Default: 10
Ограничение "сверху" на количество возвращаемых элементов данных
Example: limit=20

level

string (Level)
Уровень сложности
Enum: "easy" "medium" "hard"

search

string
Запрос по названию

Рисунок 3.8 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

currentPage required	integer <int64> Текущая страница
totalPages required	integer <int64> Всего страниц
items ▾ required	Array of objects (TaskLite)

Array [

id required	string <uuid> (TaskId) Уникальный идентификатор задачи
name required	string (TaskName) Имя задачи
level required	string (Level) Уровень сложности Enum: "easy" "medium" "hard"

]

Рисунок 3.9 – Схема результата

Алгоритм представлен на рисунке 10.

GetTaskList - получение списка задач

Параметры

Имя	Тип	Обязательность	По умолчанию
offset	query	-	0
limit	query	-	10
level	query	+	-
search	query	+	-

Алгоритм

1. Обогатится задачами из **tasks**.

Параметр	Значение
offset	offset
limit	limit
level	level , если передано
name	LIKE %query% если передано
is_enabled	true
is_private	false

2. Обогатится количеством страниц и базы с limit = **limit**

3. Вернуть ответ в соответствии мапингу из **openApi**

Рисунок 3.10 – Алгоритм

3.2.3 Создание задачи

Данный метод защищен авторизацией. Предоставляет возможность создания задачи. По-умолчанию задача является неактивной и недоступна при получении списка задач, но ее может просмотреть автор и также можно перейти к решению по ссылке. Это связано с тем, что задача может быть нестабильна и не готова полностью к общему доступу, но для того чтобы ее сделать доступной нужно выполнить успешные решения на всех языках.

Так же для всех языков формируется драйвер – код, который служит основой для тестирования программы пользователя.

Контракт представлен на рисунках 3.11 – 3.12.

name required	string (TaskName) Имя задачи
level required	string (Level) Уровень сложности Enum: "easy" "medium" "hard"
description required	string Описание
inputTypes required	Array of strings (TaskInputTypes) Типы входных параметров
outputType required	string (TaskOutputType) Тип выходного параметра
languages	Array of strings (TaskLanguages) Допустимые языки
isEnabled	boolean (IsEnabled) Отображается ли задача в списке (есть доступ по id)
isPrivate	boolean (IsPrivate) Отображается ли задача только в соревновании (есть доступ по id)
methodName required	string (MethodName) Название реализуемого метода(функции)
tests ▾ required	Array of objects (CreateTaskTestsRequest) Array [inputData required Array of strings (TestInputData) Входные данные теста outputData required string (TestOutputData) Выходные данные теста]
startCodes required	string (TaskStartCodes) Заготовки кода для пользователя. язык -> драйвер

Рисунок 3.11 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

id	string <uuid> (TaskId)
required	Уникальный идентификатор задачи

Рисунок 3.12 – Схема результата

Алгоритм представлен на рисунке 3.13.

CreateTaskOperation - создание задачи

Параметры

Название	Тип	Обязательность	Описание
TaskId	path	+	Id задачи
UserId	JWT	+	Id пользователя
CreateTaskRequest	body	+	Запрос

Алгоритм

1. Проверить тело запроса на валидность. При нарушении любого из пунктов выбросить **CreateRequestError(400)**
 - i. **inputTypes** не пустой
 - ii. Количество входных параметров в всех из **tests** равно количеству типов в **inputTypes**
 - iii. Количество тестов больше или равно 2
 - iv. Для всех языков присутствует стартовый код в **startCodes**
2. Составить **драйверы** для всех языков. В основу **драйвера** для языка подставить:
 - i. Имя метода
 - ii. Возвращаемый тип
 - iii. Сгенерировать и подставить вызовы функций для чтения входных параметров
 - iv. Вызов функции пользователя
 - v. Функции чтения
3. Вернуть **id** задачи

Рисунок 3.13 – Алгоритм

3.2.4 Добавление тестов к задаче

Метод позволяет добавить новые тесты к существующей задаче.

Добавить тесты может только тот, кто создал задачу.

Контракт представлен на рисунках 3.14 – 3.15.

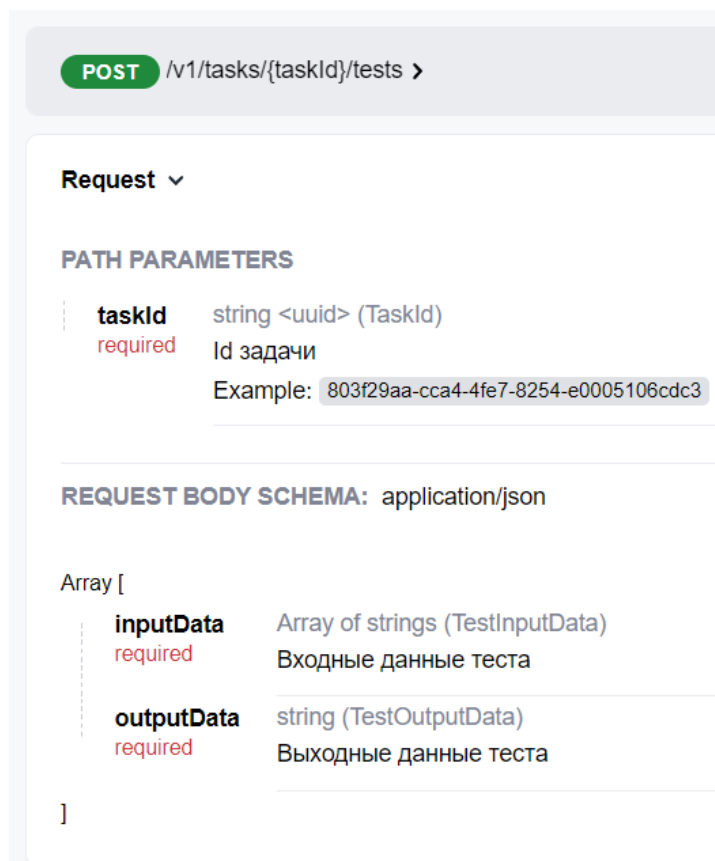


Рисунок 3.14 – Путь к вызову операции и входные параметры



Рисунок 3.15 – Схема результата

Алгоритм представлен на рисунке 3.16.

CreateTestsOperation - создание тестов

Параметры

Название	Тип	Обязательность	Описание
TaskId	path	+	Id задачи
UserId	JWT	+	Id пользователя
CreateTaskRequest	body	+	Запрос

Алгоритм

1. Обогатится задачей из **tasks** (или в кеше) по `id = test.taskId`
 - i. Если не найдено, выбросить ошибку **TaskNotFound (422)**
 - ii. Если `tasks.userId != userId` из **JWT**, выбросить ошибку **Forbidden (403)**
2. Обогатится тестами из **tests** по `taskId`
3. Убрать из тестов в запросе такие, которые уже есть или задублированы
 - iv. Если найдены противоречивые или количество элементов в `inputData !=` колву в текущих тестах - выбросить **TestsNotCorrect (422)**
4. Сохранить новые тесты в **tests**
5. Отправить событие об инвалидации кеша в топик `codest.cache.invalidate` в соответствии с **контрактом**

Рисунок 3.16 – Алгоритм

3.2.5 Удаление тестов задачи

Метод позволяет удалить тесты существующей задачи. Удалить тесты может только тот, кто создал задачу.

Контракт представлен на рисунке 3.17.

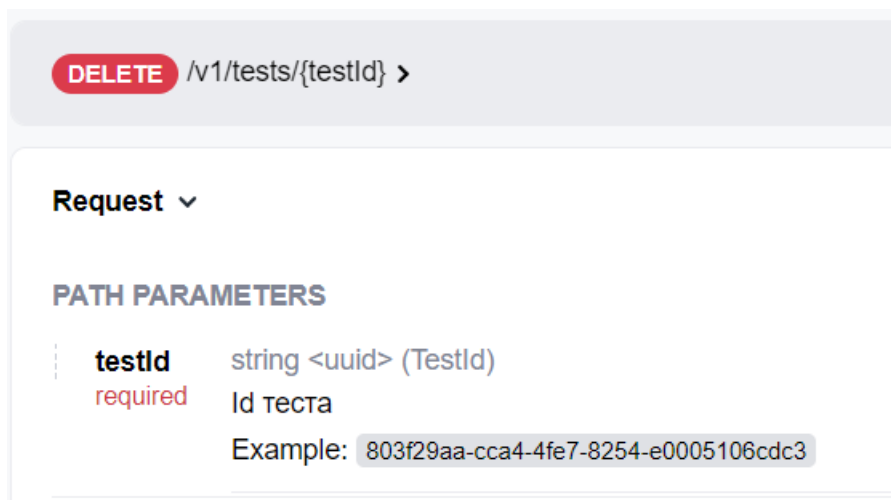


Рисунок 3.17 – Путь к вызову операции и входные параметры

Алгоритм представлен на рисунке 3.18

DeleteTests - удаление теста

Параметры

Название	Тип	Обязательность	Описание
UserId	JWT	+	Id пользователя
TestId	path	+	Id теста

Алгоритм

1. Обогатится тестом по id в **tests**
 - i. Если не найдено выбросить **TestNotFound (422)**
2. Обогатится задачей из **tasks** (или в кеше) по **id = test.taskId**
 - i. Если не найдено, выбросить ошибку **TaskNotFound (422)**
 - ii. Если **tasks.userId != userId** из **JWT** , выбросить ошибку **Forbidden (403)**
3. Удалить тест из **tests** по **id = testId**.
4. Отправить событие об инвалидации кеша в топик **codest.cache.invalidate** в соответствии с **контрактом**

Рисунок 3.18– Алгоритм

3.2.6 Получение собственных задач

Метод для получения задач, созданных пользователем, полученным из JWT.

Контракт представлен на рисунке 3.19.

Список задач пользователя

GET /v1/tasks/own >

Responses ▾

200 Ok

RESPONSE SCHEMA: application/json

Array [

id required	string <uuid> (TaskId) Уникальный идентификатор задачи
name required	string (TaskName) Имя задачи
level required	string (Level) Уровень сложности Enum: "easy" "medium" "hard"
description required	string Описание
isAuthor required	boolean (isAuthor) Является ли пользователь автором задачи
languages required	Array of strings (TaskLanguages) Допустимые языки
isEnabled required	boolean (isEnabled) Отображается ли задача в списке (есть доступ по id)
isPrivate required	boolean (isPrivate) Отображается ли задача только в соревновании (есть доступ по id)
startCodes required	string (TaskStartCodes) Заготовки кода для пользователя. язык -> драйвер

]

Рисунок 3.19 – Путь к вызову операции и схема результата
Алгоритм представлен на рисунке 3.20.

GetOwnTasks - получение списка своих задач

Параметры

Имя	Тип	Обязательность
userId	JWT	+

Алгоритм

- 1. Обогатиться задачами из **tasks** по **user_id = userId**
- 2. Вернуть ответ в соответствии мапингу из **openApi**

Рисунок 3.20 – Алгоритм

3.2.7 Получение решений пользователя

Получение всех решений задачи пользователя из JWT.

Контракт представлен на рисунках 3.21 – 3.22.

Возвращает решения задачи пользователя (lite)

GET

/v1/tasks/{taskId}/solutions >

Request ▾

PATH PARAMETERS

taskId

required

string <uuid> (taskId)

Id задачи

Example: 803f29aa-cca4-4fe7-8254-e0005106cdc3

Рисунок 3.21 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

Array [
id required	string <uuid> (SolutionId) Уникальный идентификатор решения
status required	string (SolutionStatus) Статус решения Enum: "pending" "accepted" "runtime_error" "compile_error" "test_error" "timeout_error" "internal_error"
createdAt required	string <date-time> (SolutionCreatedAt) Дата создания решения
language required	string (SolutionLanguage) Язык решения
]	

Рисунок 3.22 – Схема результата

Алгоритм представлен на рисунке 3.23.

GetTaskSolutions - получение решений по id задачи для пользователя

Параметры

Имя	Тип	Обязательность
taskId	path	+
userId	JWT	+

Алгоритм

1. Обогащаться попытками по taskId и userId
2. Вернуть ответ в соответствии мапингу из [openApi](#)

Рисунок 3.23 – Алгоритм

3.2.8 Получение решения по ID

Для получения решений по уникальному идентификатору решения, которое создается при отправке решения. Если статус решения не меняется в течении одной минуты считать, что произошла непредвиденная ошибка.

Контракт представлен на рисунках 3.24 – 3.25.

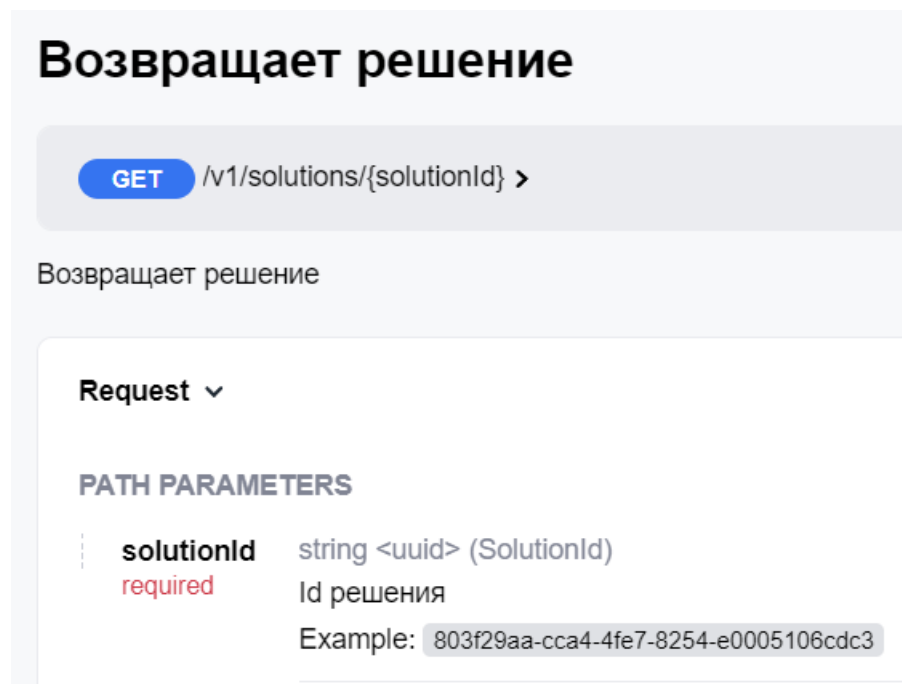


Рисунок 3.24 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

id required	string <uuid> (SolutionId) Уникальный идентификатор решения
status required	string (SolutionStatus) Статус решения Enum: "pending" "accepted" "runtime_error" "compile_error" "test_error" "timeout_error" "internal_error"
code required	string (SolutionCode) Код решения
language required	string (Language) Допустимый язык
createdAt required	string <date-time> (SolutionCreatedAt) Дата создания решения
error	Array of strings (SolutionError) Описание ошибки

Рисунок 3.25 – Схема результата

Алгоритм представлен на рисунке 3.26.

GetSolutionById - получение решения по id

Параметры

Имя	Тип	Обязательность
solutionId	path	+

Алгоритм

1. Обогатится попыткой из **attempts** (или в кеше)
 - i. Если не найдено, выбросить ошибку **AttemptNotFound (422)**
2. Если **status = 'pending'** и **createdAt - now() > 1 минута**, то
 - i. обновить запись в **attempts** по **id = solutionId** и **status = 'pending'**

Поле	Значение
status	internal_error
error	Ошибка. id = solutionId

- ii. Если было обновление отправить событие об инвалидации кеша в топик **codest.cache.invalidate** в соответствии с **контрактом**
3. Вернуть ответ в соответствии мапингу из **openApi**

Рисунок 3.26 – Алгоритм

3.2.9 Создание решения

Для создания решения. После создания решения оно также сохраняется в кеш.

Контракт представлен на рисунках 3.27 – 3.28.

Сохраняет решение

POST

/v1/solutions/task/{taskId} >

Сохраняет решение

Request ▾

PATH PARAMETERS

taskId

required

string <uuid> (TaskId)

Id задачи

Example: 803f29aa-cca4-4fe7-8254-e0005106cdc3

REQUEST BODY SCHEMA: application/json

Решение

code

required

string (SolutionCode)

Код решения

language

required

string (SolutionLanguage)

Язык решения

Рисунок 3.27 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

<div><div>id</div><div>required</div></div>	<div>string <uuid> (SolutionId)</div> <div>Уникальный идентификатор решения</div>
<div><div>status</div><div>required</div></div>	<div>string (SolutionStatus)</div> <div>Статус решения</div> <div>Enum: "pending" "accepted" "runtime_error" "compile_error" "test_error" "timeout_error" "internal_error"</div>
<div><div>code</div><div>required</div></div>	<div>string (SolutionCode)</div> <div>Код решения</div>
<div><div>language</div><div>required</div></div>	<div>string (Language)</div> <div>Допустимый язык</div>
<div><div>createdAt</div><div>required</div></div>	<div>string <date-time> (SolutionCreatedAt)</div> <div>Дата создания решения</div>
<div><div>error</div></div>	<div>Array of strings (SolutionError)</div> <div>Описание ошибки</div>

Рисунок 3.28 – Схема результата

Алгоритм представлен на рисунке 3.29.

53

CreateSolutionOperation - создание решения

Параметры

Название	Тип	Обязательность	Описание
TaskId	path	+	Id задачи
UserId	header	+	Id пользователя

Алгоритм

1. Обогатится задачей в **tasks** (или кеше)
 - i. Если задача не найдена выбросить **TaskNotFound** и завершить обработку
 - ii. **language** из тела не в **languages** из задачи выбросить **LanguageNotAcceptable** и завершить обработку
2. Сохранить в **attempts**. Маппинг:

Поле	Значение
id	uuid()
taskId	taskId из path
userId	userId из header
code	code из запроса
language	language из запроса

3. Отправить сообщение в **codest.runner.request** в соответствии с **контрактом** и ключем id из attempts Маппинг:

Поле	Значение
tests[]	task.tests
code	task.driver с подставленным code по ключу solution и testsCount = task.tests.size()
language	language из запроса

Рисунок 3.29 – Алгоритм

3.2.10 Обработка завершения решения

Для обработки выполненной проверки решения. Выполнение операции активизируется при чтении сообщения из Kafka.

Схема события представлена на рисунке 3.30.

RunCodeResponseEvent

Описание

Название поля	Тип	Обязательность	Описание
errorType	CodeRunnerErrorType	-	Информация об ошибке/ Не пройденный тесткейс
output	String[]	+	Вывод программы построчно в виде массива

CodeRunnerErrorType (Enum)

Значения

Название	Описание
COMPILE_ERROR	Ошибка компиляции
RUNTIME_ERROR	Ошибка выполнения
INTERNAL_ERROR	Ошибка сервиса
TIME_EXCEED_ERROR	Время выполнения превышено
TEST_ERROR	Программа не прошла тестирование

Рисунок 3.30 – Схема события

Алгоритм представлен на рисунке 3.31.

HandleRunCodeResponse - обработка завершения запуска кода

Алгоритм

1. Прослушать событие из топика **codest.runner.response** и декодировать в соответствии с **контрактом**
2. Обогатится попыткой в **attempts**. Если не найдено выбросить **AttemptNotFoundError** и перейти к обновлению попытки с **InternalError** и завершить обработку. Если **status != 'pending'** пропустить событие
3. Обогатится тестами по id задачи из **attempts**
4. Если **errorType** из события = null обновить попытку

Поле	Значение
status	accepted

5. Иначе выполнить обновление в соответствии с маппингом:

errorType	status	error
RUNTIME_ERROR	runtime_error	output из события
INTERNAL_ERROR	internal_error	output из события
COMPILE_ERROR	compile_error	output из события
TIME_EXCEED_ERROR	timeout_error	output из события
TEST_ERROR	test_error	output из события

6. Отправить событие об инвалидации кеша в топик **codest.cache.invalidate** в соответствии с **контрактом**
7. При возникновении неожиданной ошибки залогировать, обновить и отправить событие об инвалидации

Поле	Значение
status	internal_error
error	Непредвиденная ошибка по id = \$id

Рисунок 3.31 – Алгоритм

3.3 Разработка сервиса пользователей и аутентификации

Для аутентификации в системе будет использоваться система на основе токенов доступа. Аутентификация на основе токенов упрощает процесс аутентификации для уже известных пользователей. Для начала работы пользователь отправляет запрос к серверу, указав имя пользователя и пароль. Затем сервер подтверждает их на основании значений, зарегистрированных в

его базе данных идентификационной информации. Если идентификационные данные подтверждены, сервер возвращает токен аутентификации (который может храниться в базе данных).

Аутентификация на основе токенов обычно состоит из четырёх этапов:

1. Первоначальный запрос — пользователь запрашивает доступ к защищённому ресурсу. Изначально пользователь должен идентифицировать себя способом, не требующим токена, например, при помощи имени пользователя или пароля.

2. Верификация — аутентификация определяет, что идентификационные данные пользователя верны.

3. Токены — система выпускает токен и передаёт его пользователю. В случае аппаратного токена это подразумевает физическую передачу токенов пользователю. В случае программных токенов это происходит в фоновом режиме, пока фоновые процессы пользователя обмениваются данными с сервером.

4. Сохранение — токен удерживается пользователем, или физически, или в браузере/мобильном телефоне. Это позволяет ему выполнять аутентификацию без указания идентификационных данных.

Для того чтобы не запрашивать каждый раз подтверждения правильности токена у стороннего сервиса или базы данных наша система использует формат JWT (JSON Web Token).

JWT состоит из трёх частей:

1. Заголовок, содержащего тип токена и используемый алгоритм шифрования.

2. Полезной нагрузки, предоставляющей идентификационные данные аутентификации и другую информацию о пользователях или аккаунте. В нашем случае в качестве полезной нагрузки выступает userID.

3. Подписи, содержащей криптографический ключ, который можно использовать для подтверждения истинности информации в полезной нагрузке.

Подпись получается путем кодирования заголовка, полезной нагрузки в формате base64 и подписыванием их с использованием секретного ключа. Например, при помощи алгоритма HS256. Теперь для того чтобы проверить действительность этого токена необходимо лишь знать секретный ключ. Так же обычно токену устанавливается время жизни.

3.3.1 Регистрация пользователя

Метод для регистрации пользователя. Пароль пользователя хранится в хешированном при помощи BCrypt виде. BCrypt является одним из самых сложных для взлома хешей.

Контракт представлен на рисунках 3.32 – 3.33.

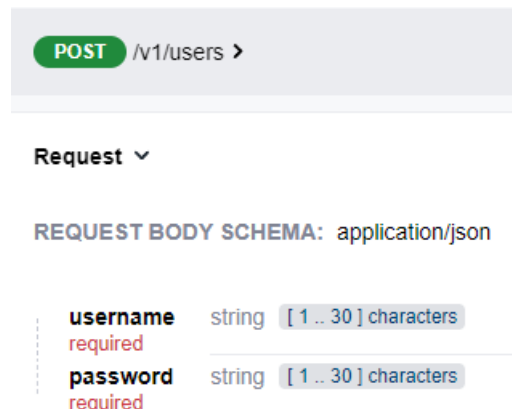


Рисунок 3.32 – Путь к вызову операции и входные параметры

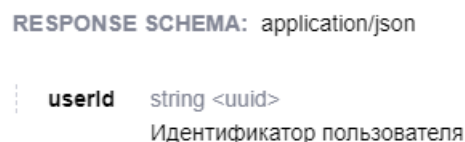


Рисунок 3.33– Схема результата

Алгоритм представлен на рисунке 3.34.

CreateUserOperation- создание решения

Параметры

Название	Тип	Обязательность	Описание
request	body	+	Данные пользователя

Алгоритм

1. Сохранить в **users**. Маппинг:

Поле	Значение
id	uuid()
username	request.username
password	request.password захешированное при помощи BCrypt

- i. Если конфликт при вставке - выбросить исключение **UsernameAlreadyExists**

2. Вернуть **id**

Рисунок 3.34 – Алгоритм

3.3.2 Логин пользователя

Для проверки пользовательских данных для входа и генерации токена JWT.

Контракт на рисунках 3.35 – 3.36.

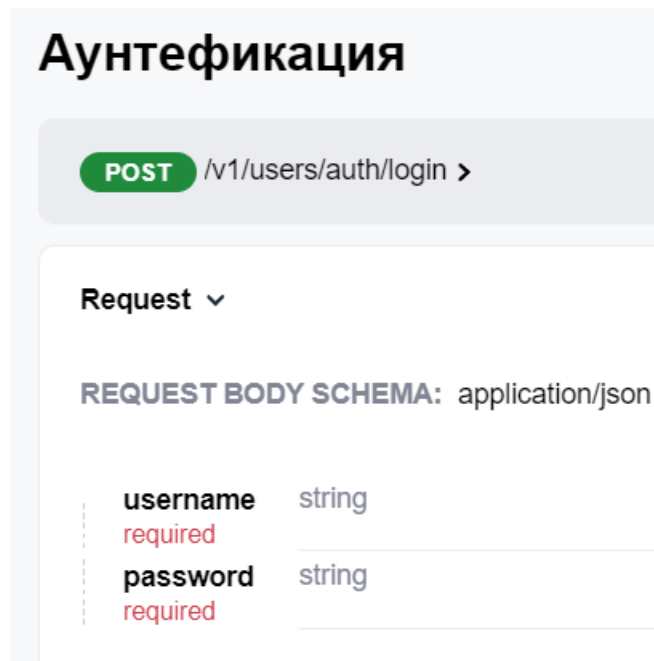


Рисунок 3.35 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

token	string
required	JWT токен

Рисунок 3.36 – Схема результата

Алгоритм представлен на рисунке 3.37.

LoginOperation- создание решения

Параметры

Название	Тип	Обязательность	Описание
request	body	+	Данные пользователя

Алгоритм

- Получить пользователя из **users** по `users.username = request.username`:
 - Если не найден выбросить **Unauthorized(401)**
- Захешировать **request.password** при помощи алгоритма **BCrypt**
 - Если **users.password = password** из п.2, то сгенерировать токен с **userId** и временем жизни **30 минут**
 - Иначе выбросить **Unauthorized(401)**

Рисунок 3.37 – Алгоритм

3.4 Разработка сервиса соревнований.

Сервис служит для создания соревнований, регистрации в соревнованиях, а также просмотра результатов.

В этом сервисе используется кеш соревнований по ID. Соревнования являются полностью неизменяемыми и поэтому не требуют инвалидации.

3.4.1 Получение информации о соревновании по ID

Для получения информации и участия в соревновании.

Контракт на рисунках 3.38 – 3.39.

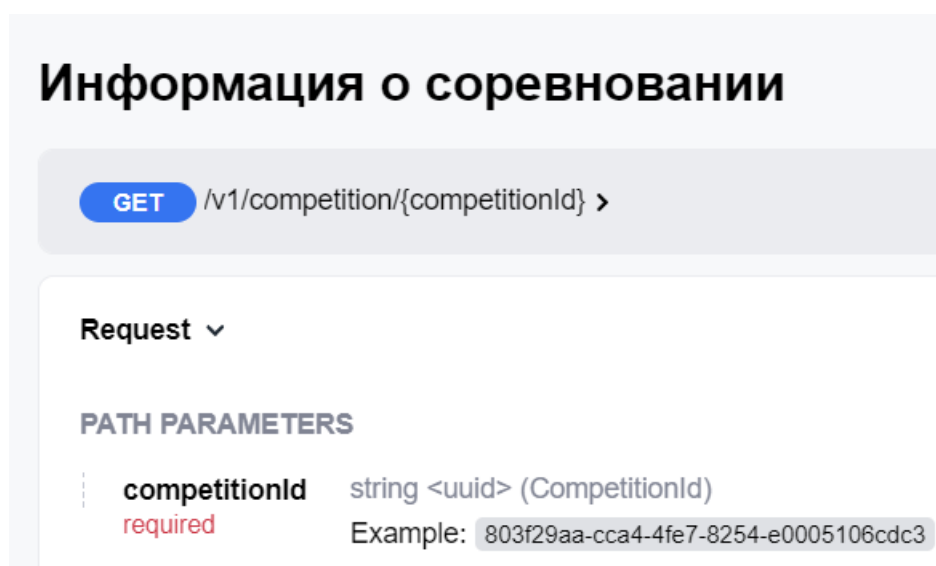


Рисунок 3.38 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

id required	string <uuid> (CompetitionId)
title required	string (CompetitionTitle) [1 .. 30] characters Название
description required	string (CompetitionDescription) Описание
startAt required	string <date-time> (CompetitionStartAt) Дата-Время начала
endAt required	string <date-time> (CompetitionEndAt) Дата-Время конца
isPrivate required	boolean (CompetitionIsPrivate) Доступно ли в списке соревнований
tasks ▾ required	Array of objects (CompetitionTasksLiteList) Список задач
Array [<div><div><div>id required</div><div>string <uuid> (TaskId)</div></div><div>name required</div><div>string (TaskName) Имя задачи</div><div>level required</div><div>string (Level) Уровень сложности Enum: "easy" "medium" "hard"</div></div>]	
isAuthor required	boolean Является ли автором пользователь из JWT
isParticipate required	boolean Участвует ли пользователь из JWT

Рисунок 3.39 – Схема результата

Алгоритм представлен на рисунке 3.40.

GetCompetitionByld - получение соревнования по id

Параметры

Название	Тип	Обязательность	Описание
CompetitionId	Path	+	Id соревнования
UserId	JWT	+	Id пользователя

Алгоритм

1. Получить соревнование из **competitions** (или из кеша) по **id = competitionId**
 - i. Если не найдено, выбросить **CompetitionNotFound(422)**
2. Вернуть Мappings:

Название	Значение
id	competitions.id
title	competitions.title
description	competitions.description
startAt	competitions.startAt
endAt	competitions.endAt
isPrivate	competitions.isPrivate
isAuthor	competitions.userId = userId из JWT
isParticipate	в competitions_users есть запись competitionId = competitions.id и userId = competitions.userId
tasks	Задачи из /v1/tasks{taskId}

Рисунок 3.40 – Алгоритм

3.4.2 Получение всех соревнований в которых участвовал пользователь

Для получения списка соревнований в которых участвует или участвовал пользователь. ID пользователя берется из JWT.

Контракт на рисунке 3.41.

Список соревнований в которых записан пользователь

GET /v1/competition/participate >

Responses ▾

200 OK

RESPONSE SCHEMA: application/json

Array [

id required	string <uuid> (CompetitionId)
title required	string [1 .. 30] characters Название
isParticipate required	boolean Участвует ли пользователь из JWT
startAt required	string <date-time> (CompetitionStartAt) Дата-Время начала
endAt required	string <date-time> (CompetitionEndAt) Дата-Время конца

]

Рисунок 3.41 – Путь к вызову операции и схема результата
Алгоритм представлен на рисунке 3.42.

GetParticipatingCompetitions - получение соревнования в которых участвовал пользователь

Параметры

Название	Тип	Обязательность	Описание
UserId	JWT	+	Id пользователя

Алгоритм

1. Получить id соревнований из **competitions_users** по **competitions_users.user_id** = userId из JWT
2. Вернуть. Маппинг:

Название	Значение
id	competitions.id
title	competitions.title
startAt	competitions.startAt
endAt	competitions.endAt
isPrivate	competitions.isPrivate
isParticipate	true

Рисунок 3.42 – Алгоритм

3.4.3 Участие в соревновании

Для принятия участия в соревновании.

Контракт на рисунке 3.43.

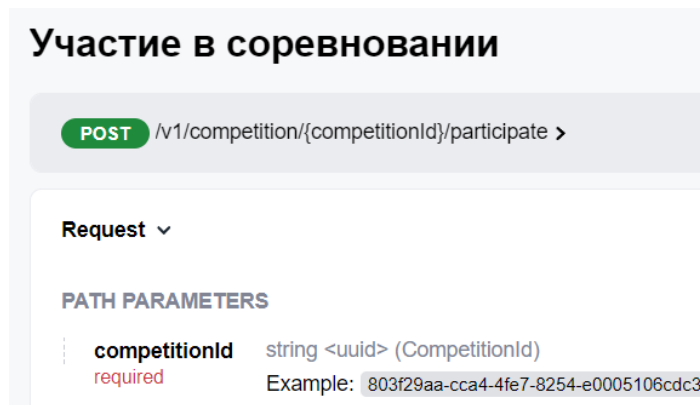


Рисунок 3.43 – Путь к вызову операции и входные параметры
Алгоритм представлен на рисунке 3.44.

ParticipateOperation - Принять участие

Параметры

Название	Тип	Обязательность	Описание
CompetitionId	Path	+	Id соревноваия
UserId	JWT	+	Id пользователя JWT

Алгоритм

- Получить соревнование из **competitions** по **id = competitionId**
 - Если не найдено, выбросить **CompetitionNotFound(422)**
 - Если соревнование началось выбросить **CompetitionAlreadyStarted(422)**
- Вставить запись в **competitions_users**

Рисунок 3.44 – Алгоритм

3.4.4 Получение всех соревнований, которые организовал пользователь

Для получения списка соревнований, которые создал пользователь. ID пользователя берется из JWT.

Контракт на рисунке 3.45.

Список соревнований созданных пользователем

GET /v1/competition/own >

Responses ▾

200 OK

RESPONSE SCHEMA: application/json

Array [

id required	string <uuid> (CompetitionId)
title required	string [1 .. 30] characters Название
isParticipate required	boolean Участвует ли пользователь из JWT
startAt required	string <date-time> (CompetitionStartAt) Дата-Время начала
endAt required	string <date-time> (CompetitionEndAt) Дата-Время конца

]

Рисунок 3.45 – Путь к вызову операции и входные параметры
Алгоритм представлен на рисунке 3.46.

GetOwnCompetitions - получение соревнования созданных пользователем

Параметры

Название	Тип	Обязательность	Описание
UserId	JWT	+	Id пользователя

Алгоритм

1. Получить соревнования из **competitions** по **user_id = userId**

Название	Значение
id	competitions.id
title	competitions.title
startAt	competitions.startAt
endAt	competitions.endAt
isPrivate	competitions.isPrivate
isParticipate	в competitions_users есть запись competitionId = competitions.id и userId = competitions.userId

Рисунок 3.46 – Алгоритм

3.4.5 Получения списка публичных соревнований

Для получения списка соревнований, в которых может участвовать пользователь.

Контракт на рисунках 3.47 – 3.48.

Список соревнований с пагинацией

GET /v1/competition/list >

Request ▾

QUERY PARAMETERS

offset	integer <int64> >= 0 Default: 0 Смещение страниц
limit	integer [1 .. 100] Default: 10 Ограничение "сверху" на количество возвращаемых элементов данных Example: limit=20

Рисунок 3.47 – Путь к вызову операции и входные параметры

Responses ▾

200 Ok

RESPONSE SCHEMA: application/json

currentPage required	integer <int64> Текущая страница										
totalPages required	integer <int64> Всего страниц										
items ▾ required	Array of objects (CompetitionLite)										
Array [<table><tr><td>id required</td><td>string <uuid> (CompetitionId)</td></tr><tr><td>title required</td><td>string [1 .. 30] characters Название</td></tr><tr><td>isParticipate required</td><td>boolean Участвует ли пользователь из JWT</td></tr><tr><td>startAt required</td><td>string <date-time> (CompetitionStartAt) Дата-Время начала</td></tr><tr><td>endAt required</td><td>string <date-time> (CompetitionEndAt) Дата-Время конца</td></tr></table>]		id required	string <uuid> (CompetitionId)	title required	string [1 .. 30] characters Название	isParticipate required	boolean Участвует ли пользователь из JWT	startAt required	string <date-time> (CompetitionStartAt) Дата-Время начала	endAt required	string <date-time> (CompetitionEndAt) Дата-Время конца
id required	string <uuid> (CompetitionId)										
title required	string [1 .. 30] characters Название										
isParticipate required	boolean Участвует ли пользователь из JWT										
startAt required	string <date-time> (CompetitionStartAt) Дата-Время начала										
endAt required	string <date-time> (CompetitionEndAt) Дата-Время конца										

Рисунок 3.48 – Схема результата

Алгоритм представлен на рисунке 3.49.

GetCompetitionsList - получение списка соревнований

Параметры

Имя	Тип	Обязательность	По умолчанию
offset	query	-	0
limit	query	-	10

Алгоритм

1. Обогатится в **competitions**.

Параметр	Значение
offset	offset
limit	limit
is_private	false

2. Обогатится количеством страниц и базы с limit = **limit**
3. Вернуть ответ в соответствии мапингу из **openApi**

Рисунок 3.49 – Алгоритм

3.4.6 Создание соревнования

Контракт на рисунках 3.50 – 3.51.

Создание соревнования

POST /v1/competition >

Request ▾

REQUEST BODY SCHEMA: application/json

title required	string (CompetitionTitle) [1 .. 30] characters Название
description required	string (CompetitionDescription) Описание
startAt required	string <date-time> (CompetitionStartAt) Дата-Время начала
endAt required	string <date-time> (CompetitionEndAt) Дата-Время конца
isPrivate required	boolean (CompetitionIsPrivate) Доступно ли в списке соревнований
tasks required	Array of strings <uuid> (CompetitionTasksIds) Список задач (id)

Рисунок 3.50 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

competitionId required	string <uuid> (CompetitionId)
----------------------------------	-------------------------------

Рисунок 3.51 – Схема результата

Алгоритм представлен на рисунке 3.52.

CreateCompetitionOperation - создание соревнования

Параметры

Название	Тип	Обязательность	Описание
request	body	+	тело запроса
UserId	JWT	+	Id пользователя

Алгоритм

1. Для каждой **taskId** из **request.tasks** выполнить запрос на **/v1/tasks{taskId}**
 - i. Если найдена задача с **isPrivate = true** и **isAuthor = false**, то выбросить **TaskIsPrivate(422)**
2. Если **startAt < now** или **endAt < startAt** выбросить исключение **PeriodIncorrect(422)**
3. Сохранить попытку в **competitions**
4. Сохранить задачи в **competitions_tasks**
5. Вернуть **id**

Рисунок 3.52 – Алгоритм

3.4.7 Получение результатов соревнования

Вычисляет результат соревнования на основе решений пользователей, созданных в рамках соревнования. Для того, чтобы не вычислять каждый раз можно сохранять результат после первого вычисления

Контракт на рисунках 3.53 – 3.54.

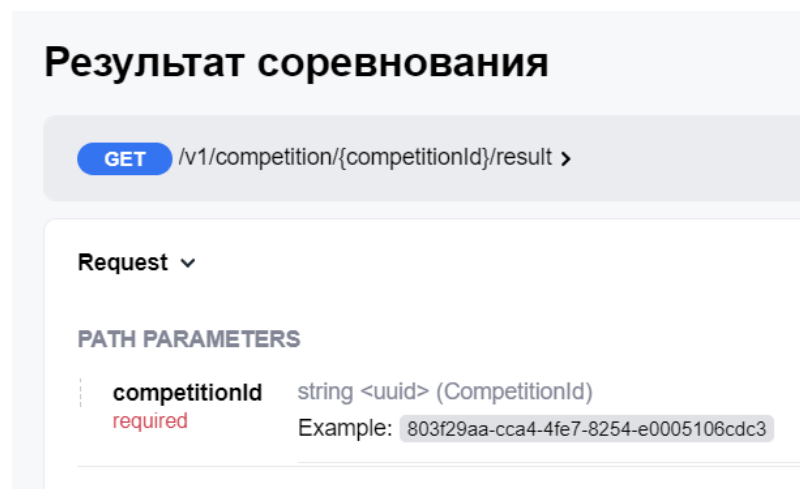


Рисунок 3.53 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

[0] ▾ object (UserCompetitionResponseLite)
Короткая результат соревнования

userId required	string <uuid> (UserId)
username required	string Имя пользователя
points required	integer Количество баллов

Рисунок 3.54 – Схема результата

Алгоритм представлен на рисунке 3.55.

GetCompetitionResultOperation - получение результатов соревнований

Параметры

Название	Тип	Обязательность	Описание
CompetitionId	Path	+	Id соревнования
UserId	JWT	+	Id пользователя
username	query	-	Имя пользователя

Алгоритм

1. Получить соревнование из **competitions** по **id = competitionId**
 - i. Если не найдено, выбросить **CompetitionNotFound(422)**
 - ii. Если соревнование не закончилось выбросить **CompetitionNotEnded(422)**
 - iii. Если пользователь не автор и не участник выбросить **Forbidden(403)**
2. Получить userIds
 - i. Если **query.username** передан, то обогатиться пользователями из **users** где **username like %query.username%**
 - ii. Иначе обогатиться из **competitions_users** по **userId = userId** из JWT
3. Вернуть список. Маппинг:

Название	Значение
userId	userId
username	users.username
points	Обогатиться последними успешными решениями для задач из competitions_tasks и сложить полученные points

Рисунок 3.55 – Алгоритм

3.4.8 Получения результата соревнования конкретного пользователя

Контракт на рисунках 3.56 – 3.57.

Результат соревнования для пользователя

GET /v1/competition/{competitionId}/result/{userId} >

Request ▾

PATH PARAMETERS

competitionId required	string <uuid> (CompetitionId) Example: 803f29aa-cca4-4fe7-8254-e0005106cdc3
userId required	string <uuid> (CompetitionId) Example: 803f29aa-cca4-4fe7-8254-e0005106cdc3

Рисунок 3.56 – Путь к вызову операции и входные параметры

RESPONSE SCHEMA: application/json

userId required	string <uuid> (UserId)
username required	string Имя пользователя
points required	integer Количество баллов
tasks ▾ required	object
[0] ▾	object (TaskWithAttemptIdResponse) Задача с попыткой
task ▾ required	object (TaskLite) Короткое описание задачи
id required	string <uuid> (TaskId)
name required	string (TaskName) Имя задачи
level required	string (Level) Уровень сложности Enum: "easy" "medium" "hard"
attemptId required	string <uuid> (TaskId)
isSolved required	boolean Решена ли задача

Рисунок 3.57 – Схема результата

Алгоритм представлен на рисунке 3.58.

GetCompetitionResultForUserOperation - получение результатов соревнований

Параметры

Название	Тип	Обязательность	Описание
CompetitionId	Path	+	Id соревнования
UserId	JWT	+	Id пользователя JWT
userId	query	+	Id пользователя

Алгоритм

- Получить соревнование из **competitions** по **id = competitionId**
 - Если не найдено, выбросить **CompetitionNotFound(422)**
 - Если соревнование не закончилось выбросить **CompetitionNotEnded(422)**
 - Если **userId из JWT** не автор и не участник выбросить **Forbidden(403)**
- Вернуть список. Маппинг:

Название	Значение
userId	query.userId
username	users.username
points	Обогащаться последними успешными решениями для задач из competitions_tasks и сложить полученные points
tasks[].task	Задачи из /v1/tasks{taskId} taskId из competition_tasks
tasks[].attemptId	Последняя успешная попытка для этой задачи и этого соревнования
tasks[].isSolved	tasks[].attemptId != null

Рисунок 3.59 – Алгоритм

3.5 Развертывание серверной части

Для работы в промышленном окружении будет использоваться Kubernetes, при помощи которого можно эффективно применять как горизонтальное, так и вертикальное масштабирование. Вертикальное масштабирование заключается в увеличении мощности серверов, а горизонтальное в увеличении количества экземпляров приложений. Kubernetes будет автоматически поддерживать заданное количество экземпляров приложений и распределять нагрузку между ними.

Минимальная единица развертывания в Kubernetes – pod (пода), которая состоит из контейнеров – экземпляров образа. Для упаковки приложений в контейнеры необходимо описать и сохранить образы – снимков системы.

Для декларативного управления количеством экземпляров сервисов используется ReplicaSet контроллер, который следит за текущим количеством живых экземпляров под и изменяет их при необходимости.

Для управление обновлением версий под используется контроллер Deployment, который определенным образом, в зависимости от выбранной стратегии, пересоздает поды при помощи управления ReplicaSet.

Для назначения подам hostname и балансировки нагрузки между подами используется VirtualService.

Для открытия доступа извне кластера Kubernetes и дополнительной балансировки нагрузки используется Ingress.

Таким образом развертывание платформы можно раздели на две части: упаковка сервисов в образы и обновление Deployment, VirtualService и IngressController

3.5.1 Создание кластера Kubernetes

Для упрощения создания и управлением кластера Kubernetes будет использоваться microk8s [29]. Для установки на Linux может использоваться snap:

```
sudo snap install microk8s --classic --channel=1.30
```

Для добавления Ingress необходимо установить дополнение Ingress:

```
microk8s enable ingress
```

Для запуска ноды и кластера используем:

```
microk8s start
```

Сейчас наш кластер состоит всего лишь из одного узла. Для добавления новых на master узла введем:

```
microk8s add-node
```

Утилита выведет нам команду, которую нужно ввести на другом узле. После чего добавится новый узел.

3.5.2 Сборка образов

Для начала требуется описать образ при помощи Dockerfile. Пример Dockerfile для codest-api:

```
FROM gradle:jdk19 AS build

COPY codest-api/src /app/codest-api/src
COPY codest-api/build.gradle.kts /app/codest-api/build.gradle.kts
COPY codest-shared/src /app/codest-shared/src
COPY codest-shared/build.gradle.kts /app/codest-shared/build.gradle.kts
COPY codest-logger/src /app/codest-logger/src
COPY codest-logger/build.gradle.kts /app/codest-logger/build.gradle.kts
COPY settings.gradle.kts /app/settings.gradle.kts
COPY gradle.properties /app/gradle.properties
WORKDIR /app
RUN gradle codest-api:distTar
COPY codest-api/build/distributions/*.tar /opt/app.tar
RUN tar -xf /opt/app.tar -C /opt

FROM openjdk:19-jdk
COPY --from=build /opt/app /opt/app
ENV CLASSPATH=/opt/app/lib/*
WORKDIR /opt/app/lib
CMD java $JAVA_OPTS r3nny.codest.api.MainKt
```

В данном файле на первом этапе собирается архив с всеми зависимостями приложения и распаковывается в папку. На втором этапе приложение подготавливается к запуску. \$JAVA_OPTS берется из переменных окружения.

Для того, чтобы у пользователя был доступ к клиентскому приложению необходимо настроить веб-сервер. Из-за простоты настройки был выбран Nginx [30]. Файл настройки:

```
worker_processes 4;

events { worker_connections 1024; }

http {
    server {
        listen 80;
        server_name codest.r3nny.ru;
        root /usr/share/nginx/html;
```

```

        include /etc/nginx/mime.types;
        location / {
            try_files $uri $uri/ /index.html;
        }
    }
}

```

Dockerfile для клиентского приложения:

```

FROM node:lts-alpine as build-stage
WORKDIR /app
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build

```

```

# этап production (production-stage)
FROM nginx:stable-alpine as production-stage
COPY nginx.conf /etc/nginx/nginx.conf

COPY --from=build-stage /app/dist /usr/share/nginx/html

CMD ["nginx", "-g", "daemon off;"]

```

Для автоматизации будет использоваться GitHub Actions – средство автоматизации процессов сборки, публикации, развертывания. Пример джобы для сборки и публикации образа:

```

build-codest-api:
  runs-on: ubuntu-latest
  permissions:
    packages: write
  steps:
    - name: Checkout
      uses: actions/checkout@v3
    - name: Login to GitHub Container Registry
      uses: docker/login-action@v1
      with:
        registry: ghcr.io
        username: ${github.actor}
        password: ${secrets.TOKEN}
    - name: Setup Buildx
      uses: docker/setup-buildx-action@v2
      with:
        driver-opts: |
          image=moby/buildkit:v0.11.2
    - name: Build Codest-api
      uses: docker/build-push-action@v4
      with:
        context: ./
        file: ./${env.CODEST_API_NAME}/Dockerfile
        push: true
        tags: ${env.CODEST_API_IMAGE}
        cache-from: type=gha,scope=${env.CODEST_API_NAME}
        cache-to: type=gha,mode=max,scope=${env.CODEST_API_NAME}

```

3.5.3 Развертывание БД и Kafka

Для развертывания БД или любого приложения с состоянием требуется использовать StatefulSet, который гарантирует развертывание поды на одном и том же узле. Это необходимо для хранения данных.

Развернем один экземпляр Postgres, один экземпляр Zookeeper и три брокера Kafka. Описание StatefulSet и Service приведено в приложении.

3.5.4 Развертывание сервисов

Рассмотрим подробнее описание сервисов на примере codest-api.

Начнем с описания ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: codest-api-config
  labels:
    app: codest-api
data:
  application.conf: {...}
```

ConfigMap – объект при помощи которого можно описать пары ключ-значение, назначить их приложению и менять настройки прямо во время работы приложения. Когда обнаружит изменения в файле и перестроит граф зависимостей. В ConfigMap не стоит указывать пароли и любую конфиденциальную информацию. В нашем файле они заменены на переменные окружения.

Для хранения секретов используется объект Secret. Пример манифеста:

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
```

Позже секреты можно будет прокинуть в сервис через переменные окружения.

Далее опишем Service:

```
apiVersion: v1
kind: Service
```



```

metadata:
  name: svc-codest-api
spec:
  type: ClusterIP
  ports:
  - port: 8080
    targetPort: 8080
  selector:
    app: codest-api

```

При помощи Service мы сможем достучаться к подам с селектором app = codest-api по домену codest-api.

Самый большой манифест – Deployment. Рассмотрим его подробнее:

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: codest-api
spec:
  replicas: 2 #[1]
  selector:
    matchLabels:
      app: codest-api
  revisionHistoryLimit: 3 #[2]
  progressDeadlineSeconds: 300
  minReadySeconds: 10
  strategy: #[3]
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 0
      maxSurge: 1
  template:
    metadata:
      name: codest-api
      labels:
        app: codest-api
    spec:
      containers:
      - name: codest-api #[4]
        image: ghcr.io/r3nnyweb/codest-api:latest
        ports:
        - containerPort: 8080
        livenessProbe: #[5]
          httpGet:
            path: /system/liveness
            port: 8085
            scheme: HTTP
            initialDelaySeconds: 15
            periodSeconds: 10
        readinessProbe: #[6]
          httpGet:
            path: /system/readiness
            port: 8085
            scheme: HTTP
            initialDelaySeconds: 15
            periodSeconds: 10

```

```

resources: #[7]
  limits:
    memory: 512Mi
    cpu: 500m
env: #[8]
  - name: POSTGRES_PASSWORD
    valueFrom:
      secretKeyRef:
        name: POSTGRES
        key: POSTGRES_PASSWORD
  - name: JWT_SECRET
    valueFrom:
      secretKeyRef:
        name: JWT
        key: JWT_SECRET
  - name: POD_NAME
    valueFrom:
      fieldRef:
        fieldPath: metadata.name
  - name: JAVA_OPTS
    value: >-
      -Dconfig.file=/opt/app/config/application.conf
      -Xmx256m
      -XX:+AlwaysActAsServerClassMachine
      -XX:+UseG1GC
volumeMounts: #[9]
  - name: config
    mountPath: "/opt/app/config/"
    readOnly: true
volumes: #[10]
  - name: config
    configMap:
      name: codest-api-config

```

1 – задание количества под;

2 – количество версий ReplicaSet, которые хранит Kubernetes между которыми можно откатиться в случае ошибки;

3 – описание стратегии обновления версии под. Выбранные параметры подразумевают, что поды заменяются по одной;

4 – контейнер с приложением. Контейнеров может быть несколько как короткоживущие (например, забирающие секреты из хранилища секретов) так и связанные с жизненным циклом всей поды (например, контейнер который обогащает доп информацией все входящие запросы);

5 – проба, обозначающая жизнеспособность поды и влияет на перезапуск;

6 – проба, обозначающая готовность приложения и влияет на трафик через Service, Ingress, Istio;

7 – ресурсы, выделяемые на контейнер;

8 – переменные окружения, которые будут доступны на уровне поды. Это могут быть секреты, информация о кластере/узле/поде, просто строковые значения и т.д;

9 – описание присоединенного к поду состояния;

10 – присоединение ConfigMap;

Последнее что нужно описать – ingress контроллер для обеспечения видимости извне кластера:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: r3nny-host
spec:
  ingressClassName: nginx
  rules:
    - host: api.codest.r3nny.ru
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: svc-codest-api
                port:
                  number: 8080
```

Так же сюда будут добавлены маппинги для user-api и клиентского приложения.

3.6 Разработка клиентской части

Клиентское приложение представляет собой браузерное приложение, написанное на Vue.js – фреймворке написанном на JavaScript.

Дерево окон клиентского приложения представлено на рисунке 3.60.



Рисунок 3.60 – Дерево окон клиентского приложения

Главная страница с поиском представлена на рисунке 3.61. С нее можно перейти на форму входа (кнопка авторизация), на страницу выбора задач (через поиск или кнопку задачи), а также на страницы взаимодействия с API.

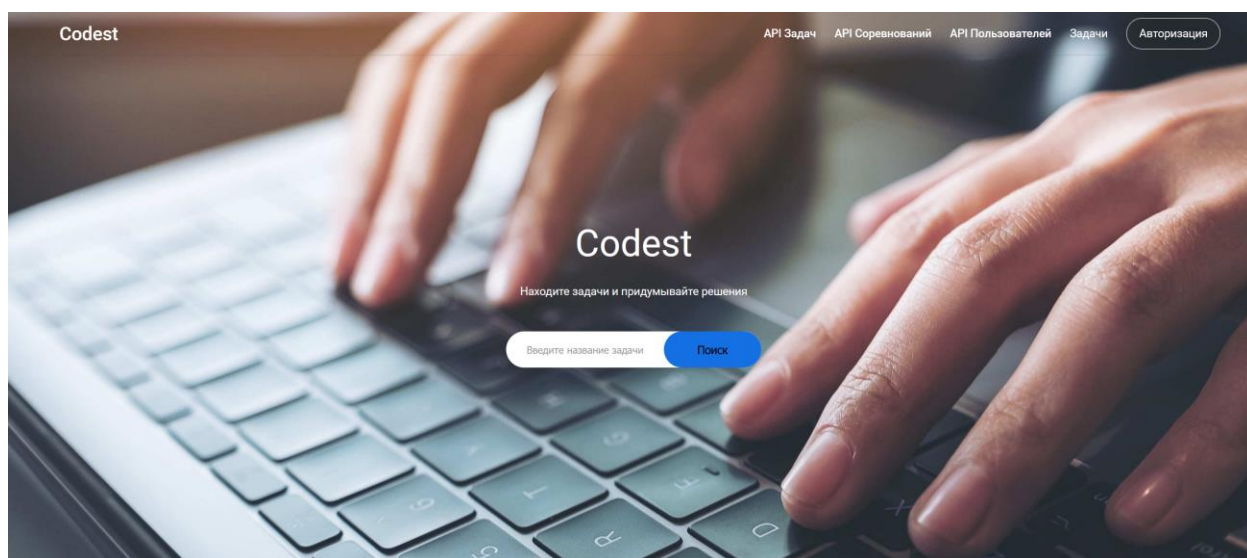
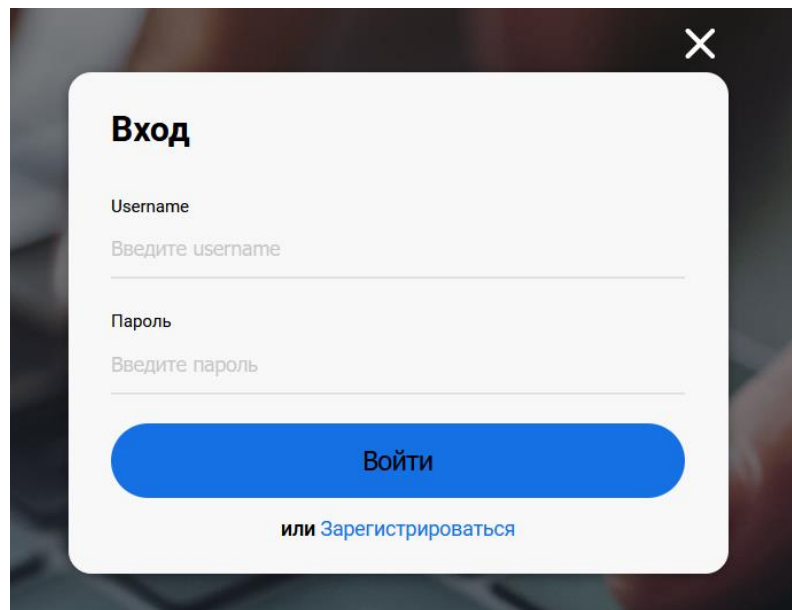


Рисунок 3.61 – Главная страница

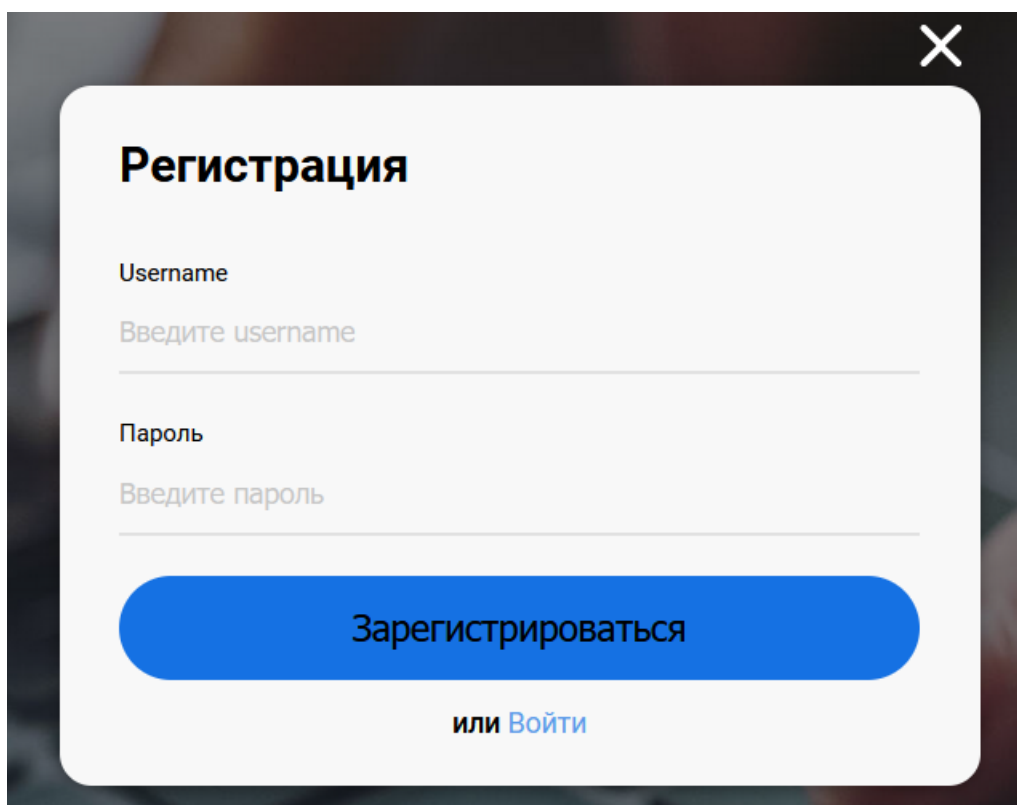
Форма входа представлена на рисунке 3.62. При входе, выполняется вызов операции Login [п.3.3.2]. Полученный токен сохраняется в localStorage и передается с каждым запросом. Так же сохраняется флаг авторизации в Vuex Store. С нее можно перейти на форму регистрации.



The image shows a login dialog box titled "Вход" (Login). It contains two input fields: "Username" with the placeholder text "Введите username" and "Пароль" (Password) with the placeholder text "Введите пароль". Below the fields is a large blue button labeled "Войти" (Login). At the bottom, there is a link that says "или Зарегистрироваться" (or Register).

Рисунок 3.62 – Форма входа

Форма регистрации представлена на рисунке 3.63. При отправке данных вызывается операция CreateUser [п. 3.3.1].



The image shows a registration dialog box titled "Регистрация" (Registration). It contains two input fields: "Username" with the placeholder text "Введите username" and "Пароль" (Password) with the placeholder text "Введите пароль". Below the fields is a large blue button labeled "Зарегистрироваться" (Register). At the bottom, there is a link that says "или Войти" (or Login).

Рисунок 3.63 – Форма регистрации

Страница выбора задач представлена на рисунке 3.64. При открытии страницы клиент вызывает операцию GetTaskList [п. 3.2.2].

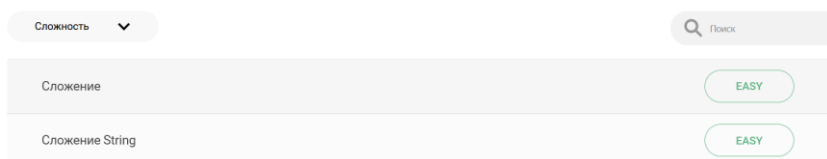


Рисунок 3.64 – Страница выбора задачи

Форма отправки решения представлена на рисунке 3.65. Для отображения задачи при открытии страницы вызывается операция `GetTaskLite` [п.3.2.1]. При отправке на проверку вызывается операция `CreateSolution` [п.3.2.9]. После отправки решения каждые 500мс вызывается операция `GetSolutionById` [п.3.2.8] с ID решения, полученным при создании решения. Вызов получения решения происходит пока статус решения 'PENDING'. Если пользователь авторизован, то ему доступна возможность посмотреть его решения, а также, если пользователь – автора задачи ему доступна возможность переключения доступности задачи (флаг `isEnabled`).

Сложение String

Задача доступна для добавления в соревнования и в общем списке [Переключить](#)

Решить Решения

EASY

lorem ipsum

JAVA

```
1 class Solution {  
2     public int sumStr(String a, String b) {  
3  
4     }  
5 }
```

Отправить на проверку

Рисунок 3.65 – Форма отправки решения

Страница просмотра решений пользователя представлена на рисунке 3.66. Решения пользователя получаются вместе с задачей. При нажатии на конкретное решение открывается окно просмотра решения.

Сложение String

Задача доступна для добавления в соревнования и в общем списке [Переключить](#)

Решить

Решения

java	2024-05-10T11:10:08.460	INTERNAL_ERROR
java	2024-05-10T11:09:32.405	COMPILE_ERROR
python	2024-05-08T20:23:44.509	ACCEPTED
python	2024-05-08T20:23:36.734	TEST_ERROR
java	2024-05-08T20:20:45.545	ACCEPTED
java	2024-05-08T20:20:20.435	TEST_ERROR
java	2024-05-08T20:20:13.366	COMPILE_ERROR

Рисунок 3.66 – Список решения пользователя

Просмотр решения пользователя показан на рисунке 3.67. Для отображения вызывается операция `GetSolutionById` [п.3.2.8].

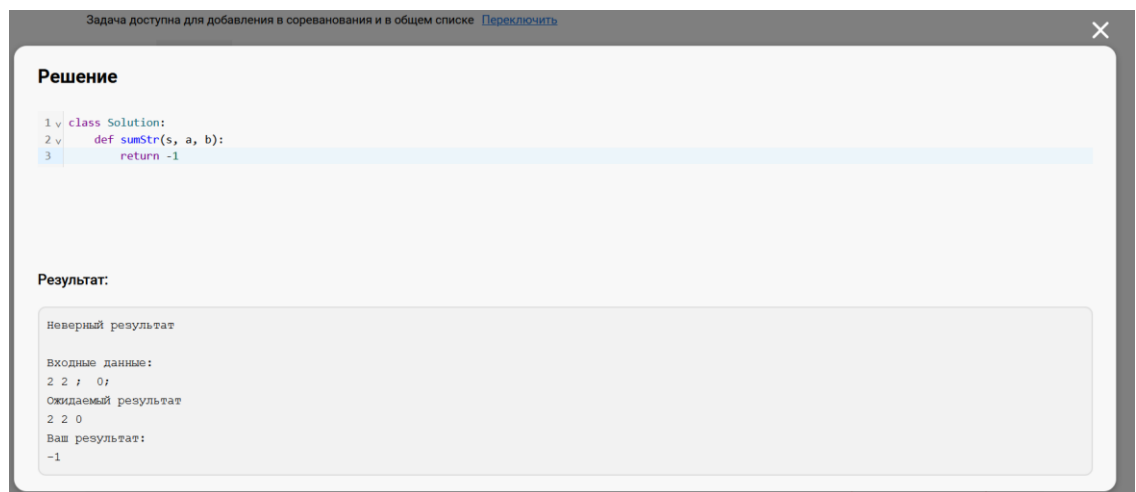


Рисунок 3.67 – Решение пользователя

Интерфейсы для взаимодействия с API представляют собой сгенерированные из OpenAPI страницы при помощи которых можно посмотреть и вызвать различные операции в сервисах. Это необходимо для операций, недоступных на клиенте: создание задачи, взаимодействие с соревнованиями. Пример такой страницы представлен на рисунке 3.68.

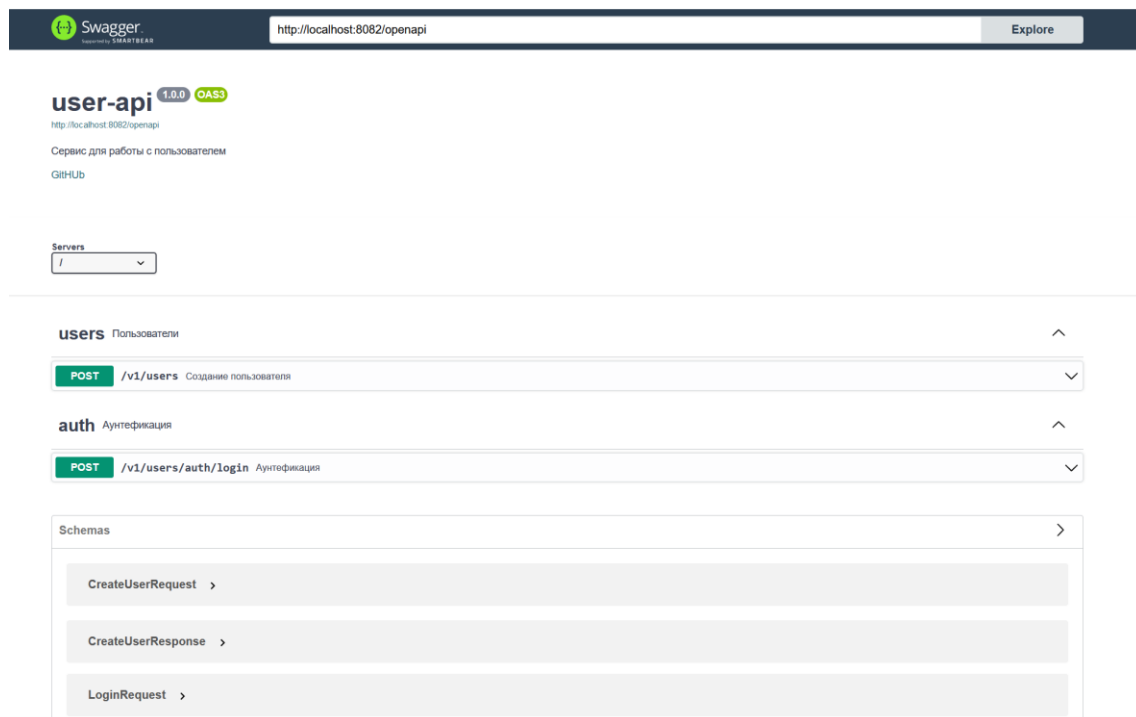


Рисунок 3.68 – Пример интерфейса для взаимодействия с API

4 ТЕСТИРОВАНИЕ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

4.1 Тестирование серверной части

При разработке серверной части активно применялся подход TDD (Test Driven Development), который подразумевает активное написания авто-тестов перед реализацией функциональности, которую эти тесты проверяют. Этот подход также называют разработкой через тестирование. Разработка через тестирование (англ. *test-driven development*, *TDD*) — техника разработки программного обеспечения, которая основывается на повторении очень коротких циклов разработки: сначала пишется тест, покрывающий желаемое изменение, затем пишется код, который позволит пройти тест, и под конец проводится рефакторинг нового кода к соответствующим стандартам [12].

Такой подход обеспечивает устойчивое развитие кодовой базы и очень высокую степень как автоматизации тест-кейсов, так и степень покрытия ими спецификации.

Как правило использование TDD подразумевает написание Unit тестов — тестов, которые проверяют правильность работы конкретной единицы поведения будь то класс, метод, или целый модуль. Если придерживаться принципа единственной ответственности, то выделить юниты для тестирования не составит труда, особенно при использовании TDD.

Важным параметром Unit тестов является скорость их выполнения. Если Unit тесты будут выполняться очень быстро это позволит разработчикам писать их больше и запускать чаще. В связи с этим в Unit тестах не учувствуют внепроцессные зависимости [13], такие как файловая система, базы данных, брокеры сообщений, Docker контейнеры. Взаимодействие юнитов между собой и с внепроцессными зависимостями проверяют интеграционные тесты.

Результат написания тестов представлен на рисунке 4.1

✔ Test Results	2 sec 669 ms	✔ Tests passed: 60 of 60 tests - 2 sec 669 ms
> ✔ RunCodeOperationTest	182 ms	
> ✔ BlockingCodeFileServiceTest	37 ms	
> ✔ ExecutionResultTesterTest	7 ms	
> ✔ ProcessRunnerTest	222 ms	
> ✔ CreateSolutionOperationTest	1 sec 23 ms	
> ✔ CreateTaskOperationTest	334 ms	
> ✔ CreateTestsOperationTest	153 ms	
> ✔ DeleteTestOperationTest	59 ms	
> ✔ EnableTaskOperationTest	308 ms	
> ✔ GetAttemptByIdOperationTest	78 ms	
> ✔ GetFullTaskOperationTest	11 ms	
> ✔ GetLiteTaskOperationTest	45 ms	
> ✔ HandleRunCodeResponseOperationTest	61 ms	
> ✔ DriverGeneratorServiceTest	125 ms	
> ✔ JavaDriverGeneratorTest	7 ms	
> ✔ LanguageDriverGeneratorBuilderTest	11 ms	
> ✔ PythonDriverGeneratorTest	1 ms	
> ✔ CreateTaskValidationKtTest	5 ms	

Рисунок 4.1 – Результат тестирования серверной части

Всего было написано 60 unit-тестов, которые покрывают операции и некоторые важные части, такие как запуск кода или формирование драйвера.

4.2 Ручное тестирование клиентской части

Поскольку серверная часть покрыта автотестами тестирование клиентской части можно проводить вручную средствами разработчика в браузере и проверять лишь то, что при определенных действиях, вызываются определенные операции на сервере, с передачей определенных параметров.

Тест-кейсы приведены в таблице 4.1.

Таблица 1 – Тест-кейсы клиента

Окно	Название	Действия	Ожидаемый результат
Главная страница	Переход к авторизации	1. Нажать кнопку “Выйти” 2. Нажать на кнопку “Авторизация”	1. Кнопка выйти смениться на кнопку “Авторизация” 2. Из Local Storage удалится JWT токен 3. Откроется модальное окно с входом.

Продолжение таблицы 1

	Переход к интерфейсу взаимодействия с API	1. В верхней части нажать на одну из кнопок с надписью “API”	1. Открывается Swagger выбранного сервиса
	Переход к списку задач через меню	1. В верхней части нажать на одну из кнопок с надписью “API”	1. Открывается страница с списком задач. 2. Делается запрос в getTasksList без параметров
	Переход к списку задач через поиск	1. В центре страницы ввести текст и нажать на поиск	1. Открывается страница с списком задач с введённым текстом. 2. Делается запрос в getTasksList без параметров
Список задач	Фильтрация по уровню сложности и названию	1. Выбрать уровень сложности и запрос по названию задачи	1. В списке останутся только задачи отфильтрованные по условиям 2. Выполниться запрос в getTasksList с соответствующими параметрами.
	Переход к отправке решения	1. Открыть выбранную задачу	1. Открывается страница с формой отправки решения 2. Выполняется запрос в getTask по ID из адреса страницы

Продолжение таблицы 1

Форма отправки решения	Отправка успешного решения и отображение результата	1. Ввести код на любом языке, решающий задачу 2. Отправить на проверку	1. Вызывается create Solution с выбранной задачей, языком, кодом. 2. Каждые 500мс вызывается getSolutionById, пока статус решения 'PENDING' 3. Показывается сообщение об успешном решении
	Отправка решения не решающего задачу и отображение результата	1. Ввести код, не решающий задачу 2. Отправить на проверку	1. Вызывается create Solution с выбранной задачей, языком, кодом. 2. Каждые 500мс вызывается getSolutionById, пока статус решения 'PENDING' 3. Показывается сообщение об неправильном решении и показывается, какой тест не прошел

Продолжение таблицы 1

	Отправка некомпилируемого решения	1. Ввести код, с ошибкой при компиляции 2. Отправить на проверку	1. Вызывается create Solution с выбранной задачей, языком, кодом. 2. Каждые 500мс вызывается getSolutionById, пока статус решения 'PENDING' 3. Показывается сообщение об ошибке и внутреннее сообщение
	Отправка решения неавторизованного пользователя	1. Убедиться, что пользователь не авторизован 2. Отправить любое решение	1. Перенаправление на форму входа 2. Решение не было отправлено
Страница решений пользователя	Просмотр списка решений пользователя	1. Открыть страницу	1. Запрос на сервер выполняться не будет 2. Отобразится список решений
	Просмотр решений	1. Открыть любое решение	1. Выполняется запрос getSolutionById 2. Открывается модульное окно с кодом и результатом решения
Форма регистрации	Ошибка во время регистрации	1. Ввести login уже существующего пользователя 2. Зарегистрироваться	1. Выполняется createUser 2. Выводиться ошибка из createUser
	Успешная регистрация	1. Ввести login не существующего пользователя 2. Зарегистрироваться	1. Выполняется createUser 2. Перенаправление на форму входа

Продолжение таблицы 1

Форма входа	Ошибка во время входа	1. Ввести login и password не существующего пользователя 2. Войти	1. Выполняется loginUser 2. Выводиться ошибка из loginUser
	Успешная регистрация	1. Ввести login и password существующего пользователя 2. Войти	1. Выполняется loginUser 2. Токен сохраняется в Local Storage 3. Перенаправление на главную

ОПИСАНИЕ ПРОГРАММЫ

1. Общие сведения. Codest – онлайн платформа для тренировки в решении алгоритмических задач и автоматизированном тестировании кандидатов. Программа написана на языках Kotlin и JavaScript. Для функционирования программы требуются: PostgreSQL, Apache Kafka, Gradle, JDK 17, Vite, Nginx, Docker, Kubernetes, Node.js 16. Представляет собой WEB приложение с браузерным клиентом.

2. Функциональное назначение. Программа предназначена для тренировки в решении задач и проведении соревнований. Пользователь может выбрать задачу по уровню сложности или названию, написать код на любом доступном языке, после чего получить результат решения, например, данные, при которых программа дает неправильный ответ. Также пользователь может создать задачу, создать соревнование, состоящее из задач, участвовать в соревновании, просматривать результаты соревнования.

3. Описание логической структуры. Программа состоит из:

- клиента (браузерного приложения);
- сервиса API (Задачи, решения);
- сервиса Competition API (Соревнования);
- сервиса User-API (Аутентификация, регистрация);
- сервиса Runner (Выполнение кода, тестирование);

Алгоритмы всех возможных операций описаны в главе 2 и в 3.

4. Используемые технические средства. Программа может быть запущена на кластере Kubernetes. Для взаимодействия клиента с программой используется браузер на любом устройстве с возможностью выхода в интернет.

5. Вызов и загрузка. Единственной точкой взаимодействия с программой является веб-браузер. Путь к сайту зависит от настроек Ingress контроллера и сервера. Например, <https://codest.r3nny.ru>. Помимо основного

клиента поддерживается взаимодействие через страницу, сгенерированную из контракта сервиса. Например, <https://codeest.r3nny.ru/api/v1/tasks/swagger-ui>

6. Входные данные. В программе может быть множество форматов и видов входных данных в зависимости от выполняемой операции. При взаимодействии напрямую с API не через клиент все данные передаются в формате JSON, а структура и типы данных описаны в документации OpenAPI и в главе 3. Примером входных данных может быть объект, описывающий решение пользователя и состоящий из кода (строка), языка (строка) и ID задачи (строка в формате UUID).

7. Выходные данные. Выходные данные также, как и входные описаны в документации OpenAPI и в главе 3. Выходные данные отображаются на клиенте в виде текста с различным форматированием. Примером выходных данных может быть объект, описывающий результат проверки решения пользователя и состоящий из кода (строка), языка (строка), ID задачи (строка в формате UUID), ID решения (строка в формате UUID), статус (строка, например, «ACCEPTED»), ошибка (массив строк), время создания (дата-время).

РУКОВОДСТВО ОПЕРАТОРА

1. Назначение программы.

Данная программа предназначена для тренировки в решении алгоритмических задач и автоматизированного тестирования кандидатов в том числе при помощи соревнований.

1.1 Функционал программы

Функционал программы:

- просмотр и фильтрация задач;
- выбор задачи для решения;
- выбор языка для решения;
- написание решения задачи на выбранном языке с автоматической проверкой;
- создание публичных или приватных задач;
- создание соревнований;
- участие в соревновании;
- просмотр результата соревнования;

2. Условие выполнения программы

Для минимального выполнения программы без возможности масштабирования при помощи Kubernetes необходим установленный набор разработчика Java 19 с компилятором; инструмент сборки Gradle 8.2; Docker; docker-compose; Node.js 16; компиляторы и интерпретаторы для всех языков программирования, доступных для решения. Минимальные требования к системе:

- операционная система: Linux, MacOS, Windows 10-11;
- оперативная память: 16 Гб;
- место на диске 4 Гб;
- процессор 6 ядер 3.0 МГц;

3. Выполнение программы.

Для запуска программы сначала необходимо запустить настроенные PostgreSQL и Kafka. Для этого в папке dev необходимо запустить:

```
docker-compose up -d
```

После запуска зависимостей необходимо запустить все сервисы. Для этого в корневом каталоге запустить команды:

```
gradlew :codest-api:run
```

```
gradlew :codest-user-api:run
```

```
gradlew :codest-competition-api:run
```

```
gradlew :codest-runner:run
```

Последний этап – запуск клиента в режиме разработчика. Для этого в каталоге codest-front запустить команды:

```
npm install
```

```
npm run dev
```

3.1 Выбор задачи для решения

Для выбора задачи необходимо в браузере открыть страницу <http://localhost:5173/tasks>. Можно отфильтровать по сложности или названию (рисунок 6.1)

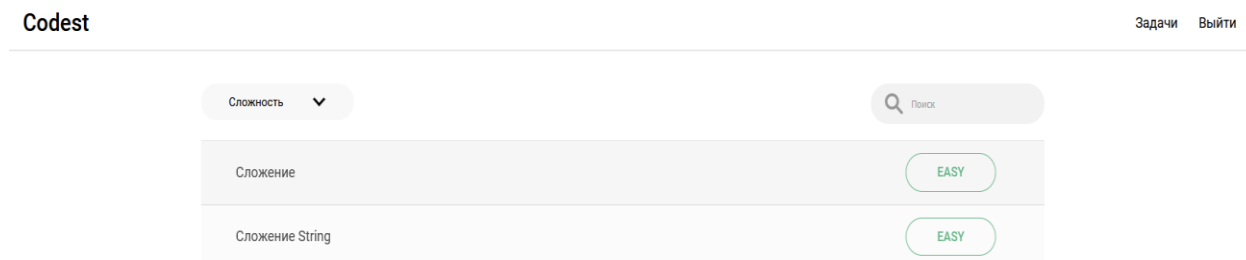


Рисунок 6.1 – Выбор задачи

3.2 Решение задачи

Для решения задачи, необходимо выбрать язык, реализовать метод, и нажать кнопку «Отправить на проверку». Пример решения с неправильным ответом на рисунке 6.2.

Сложение

Задача доступна для добавления в соревнования [Переключить](#)

Решить Решения

EASY

lorem ipsum

JAVA

```
1 v class Solution {  
2 v public int sum(int a, int b) {  
3     return a - b;  
4 }  
5 }
```

Отправить на проверку

Результат:

Неверный результат

Входные данные:

-1; 10;

Ожидаемый результат

9

Ваш результат:

-11

Рисунок 6.2 – Решение с неправильным ответом

3.3 Авторизация

Для авторизации необходимо открыть окно авторизации (рисунок 6.3) и ввести логин/пароль (рисунок 6.4).

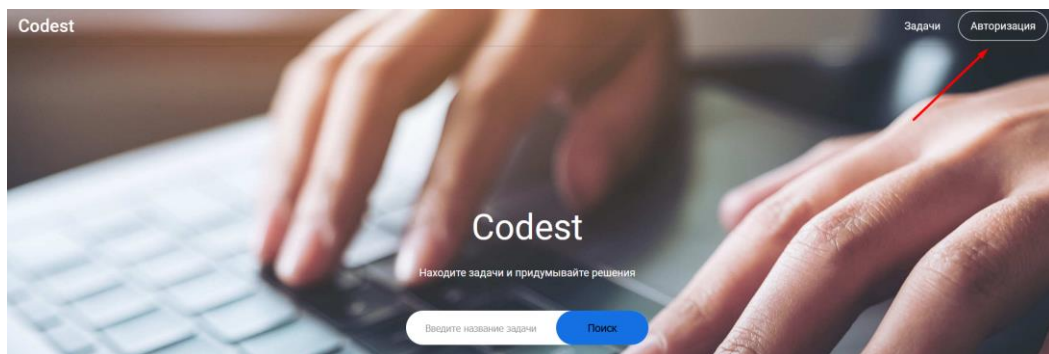


Рисунок 6.3 – Открытие окна авторизации

The screenshot shows a login window titled "Вход" (Login) with a close button (X) in the top right corner. It contains two input fields: "Username" with placeholder text "Введите username" and a red error message "Введите Username"; and "Пароль" (Password) with placeholder text "Введите пароль" and a red error message "Введите пароль". Below the fields is a grey "Войти" (Login) button. At the bottom, there is a link "или Зарегистрироваться" (or Register).

Рисунок 6.4 – Окно авторизации

3.4 Регистрация

Переход к регистрации осуществляется через окно авторизации.

The screenshot shows a registration window titled "Регистрация" (Registration) with a close button (X) in the top right corner. It contains two input fields: "Username" with placeholder text "О" and a red error message "Username минимум 2 символа"; and "Пароль" (Password) with a masked input (dots) and a red error message "Пароль минимум 8 символов". Below the fields is a grey "Зарегистрироваться" (Register) button. At the bottom, there is a link "или Войти" (or Login).

Рисунок 6.5 – Окно регистрации

3.5 Получение токена

Получение токена осуществляется непосредственно через обращение к сервису. Для этого необходимо перейти на страницу <http://localhost:8082/swagger-ui#/auth/login> и ввести логин/пароль (рисунок 6.6).

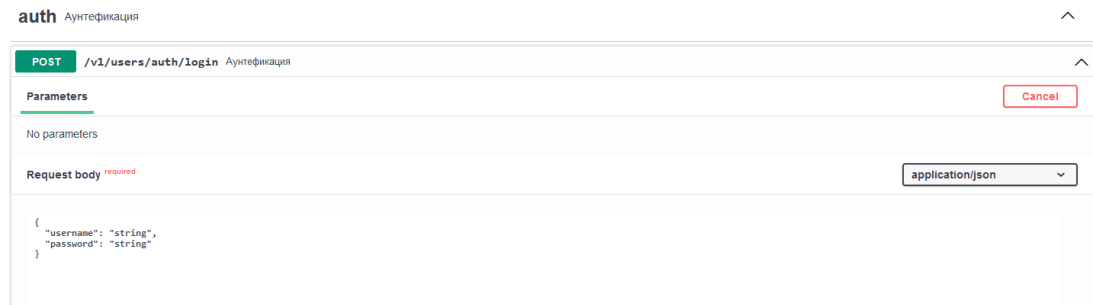


Рисунок 6.6 – Получение токена

3.6 Создание задачи

Для создания задачи необходимо перейти <http://localhost:8080/swagger-ui#/tasks/createTask> и ввести необходимые параметры (рисунок 6.7).

POST

/v1/tasks/ Создание задачи

Parameters

No parameters

Request body required

```
{
  "name": "1. Сложение",
  "level": "easy",
  "description": "lorem ipsum",
  "inputTypes": [
    "integer",
    "boolean",
    "int_array"
  ],
  "outputType": "integer",
  "languages": [
    "java"
  ],
  "isEnabled": true,
  "isPrivate": true,
  "methodName": "sum",
  "tests": [
    {
      "inputData": [
        "2",
        "0",
        "1"
      ],
      "outputData": "3"
    }
  ],
  "startCodes": {
    "java": "someCode"
  }
}
```

Execute

Рисунок 6.7 – Создание задачи

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

- 1) JUnit5 About: [Электронный ресурс]. URL: <https://junit.org/junit5/>.
- 2) Д. Макгрегор. От Java к Kotlin. – СПб.: Питер, 2023. – 448 с.
- 3) Ридчардсон К. Микросервисы. Паттерны разработки и рефакторинга. – СПб.: Питер, 2019. – 544 с.
- 4) Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. – СПб.: Питер, 2023. – 640 с.
- 5) Harihara S. Hands-On RESTful API Design Patterns and Best Practices – Birmingham: Packt, 2019. – 360с.
- 6) Документация Kotlin [Электронный ресурс]. URL: <https://kotlinlang.org/docs/home.html>.
- 7) Основное | Kora [Электронный ресурс]. URL: <https://kora-projects.github.io/kora-docs/ru/documentation/general/>.
- 8) Kotlin исходный код [Электронный ресурс]. URL: <https://github.com/JetBrains/kotlin>
- 9) Статья: “Микросервисы” [Электронный ресурс]. URL: <https://habr.com/ru/articles/249183/>.
- 10) PostgreSQL Docs [Электронный ресурс]. URL: <https://www.postgresql.org/docs/>.
- 11) Эккель Б. Философия Java, 4-е изд. – СПб.: Питер, 2021. – 1168с.
- 12) Бек К. Экстремальное программирование: разработка через тестирование. – СПб.: Питер, 2022. – 224с.
- 13) Хориков В. Принципы юнит-тестирования. – СПб.: Питер, 2022. – 320с.
- 14) Mellor A. Test-Driven Development with Java. – Birmingham: Packt, 2023. – 325с.
- 15) Freeman S., Pryce N. Growing Object-Oriented Software, Guided by Tests. – Boston: Addison-Wesley, 2010. – 385с.
- 16) Мартин Р. Чистый код: создание, анализ и рефакторинг. – СПб.: Питер, 2022. – 464с.

- 17) Куликов, С.С. Реляционные базы данных в примерах. – Минск: Четыре четверти, 2020. – 424 с.
- 18) Мартин Р. Чистая архитектура. – Спб.: Питер, 2022. – 360с.
- 19) Карнелл Д. Микросервисы Spring в действии. – СПб.: ДМК Пресс, 2022. – 490с.
- 20) Гош С. Docker без секретов. – СПб.: БВХ-Петербург, 2023. – 224с.
- 21) Kubernetes Docs [Электронный ресурс]. URL: <https://kubernetes.io/ru/docs/tutorials/kubernetes-basics/>.
- 22) Apache Kafka Docs [Электронный ресурс]. URL: <https://kafka.apache.org/documentation/>.
- 23) Introduction to Caffein Baeldung [Электронный ресурс]. URL: <https://www.baeldung.com/java-caching-caffeine>.
- 25) Gradle User Manual [Электронный ресурс]. URL: <https://docs.gradle.org/current/userguide/userguide.html>.
- 26) Soshin A. Kotlin Design Patterns and Best Practices. – Birmingham: Packt, 2024. – 475с.
- 27) Беллемар А. Создание событийно-управляемых микросервисов. – СПб.: БХВ-Петербург, 2022. – 320с.
- 28) Introduction to JSON Web Tokens [Электронный ресурс]. URL: <https://jwt.io/introduction/>.
- 29) Microk8s Docs [Электронный ресурс]. URL: <https://microk8s.io/docs>
- 30) Nginx Docs [Электронный ресурс]. URL: <https://nginx.org/ru/docs/>