СОДЕРЖАНИЕ

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 непременной 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 IT, 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 IT-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 компаниям 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:

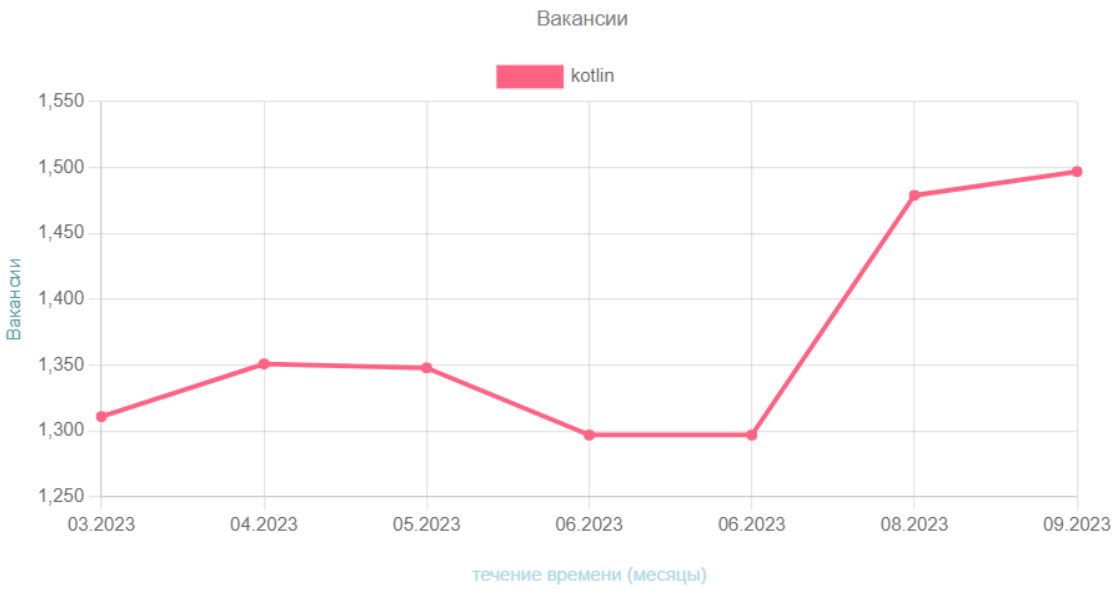
- Упрощенный процесс решения задач: участники контестов и просто желающие потренироваться в решении задач должны иметь возможность решать задачи, сводящиеся к реализации одной функции, без необходимости использования консольного ввода/вывода и приведения типов.
- Автоматическая проверка правильности 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 компаниям и другим 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. Объективными оценками, которые может предоставить платформа являются количество решенных задач и время, которое потратил пользователь на решение всего набора задач.
- Различные форматы контестов: платформа должна поддерживать как открытые, так и закрытые форматы проведения контестов.
- Возможность просмотра решений: пользователи должны иметь возможность просматривать решения других пользователей по решаемым задачам вне констеста.
- Организация удобного поиска задач: платформа должна облегчить пользователю процесс выбора задачи рекомендациями, подборками, 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. 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. 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.

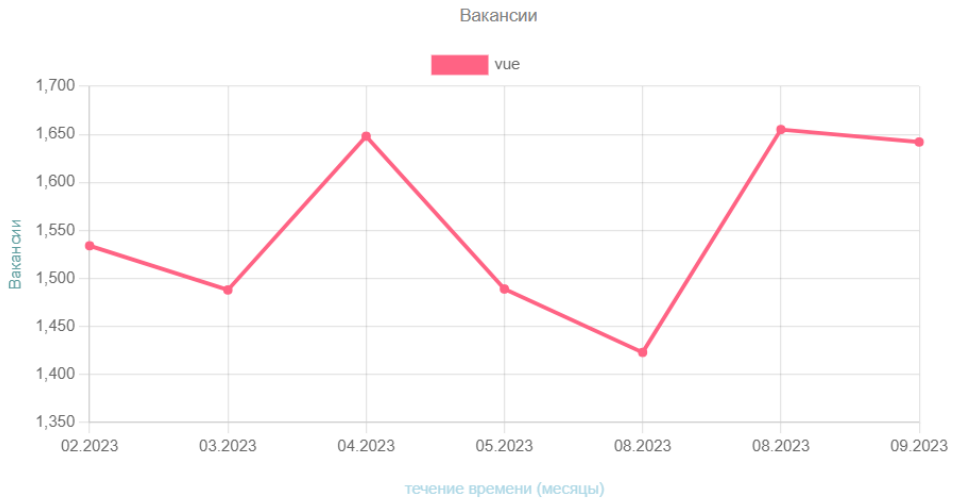


Рисунок 1.2 – Востребованность Vue на 09.2023

Выбор СУБД и технологий для размещения платформы для использования отложен до момента проектирования архитектуры.

KORA, PGSQL, KAFKA, KUBERNETES

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 серверное приложение приходит код пользователя, язык, на котором написан код и идентификатор задачи, по которому можно получить задачу из хранилища.
3. Приложение получает техническое описание задачи из базы данных. Техническое описание может состоять из тестовых данных, при помощи которых проверяется решение, название задачи, из которого формируется название реализуемого метода, входные и выходные типы, ограничение по времени выполнения. Такая усложненная структура задачи связана с тем, что мы берем на себя ответственность преобразования типов, получаемых из консоли, вызов метода, который реализовывает пользователь.
4. Из полученного технического описания задачи, кода, полученного от пользователя и специально заготовленного кода для каждого языка, формируется код, который будет выполняться на сервере. Специально заготовленный код, названный драйвером задачи, предназначен для того, чтобы подготовить программу к получению тестовых данных, считывать с консоли тестовые данные, преобразовывать их в необходимые типы, вызывать решение пользователя, выводить решение в консоль. Все это необходимо для того, чтобы минимизировать действия пользователя, не относящиеся к алгоритму.
5. В случае компилируемых языков, полученный код сохраняется в временный файл и компилируется компилятором языка.

6. Приложение создает процесс, в котором выполняется полученный код.

7. Серверное приложение, взаимодействуя с процессом через консоль, отправляет тестовые данные через консоль и считывает результаты.

8. Полученные результаты сравниваются с ожидаемыми и формируется результат проверки, который сохраняется в базу данных.

9. Результат возвращается для отображения пользователю.

В дальнейшем процесс может быть дополнен техническими деталями, связанными с архитектурой, такими как сохранение попытки перед запуском процесса и так далее.

2.2 Инфологическое проектирование Базы Данных

В предметной области можно выделить следующие сущности:

1) Пользователь
2) Задача
3) Решение
4) Тест
5) Соревнование
6) Компания

В предметной области можно выделить следующие связи между сущностями:

1) Пользователь создает Решение
2) Решение решает Задачу
3) Задача содержит Тест
4) Задача включена в Соревнование
5) Компания организует Соревнование
6) Пользователь участвует в Соревновании
7) Пользователь администрирует Компанию

Построение ER-Диаграмм

1. Связь Пользователь создает Решение

Для степени связи:

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

Для класса принадлежности сущности к связи:

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

Рисунок 2.1 – 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;

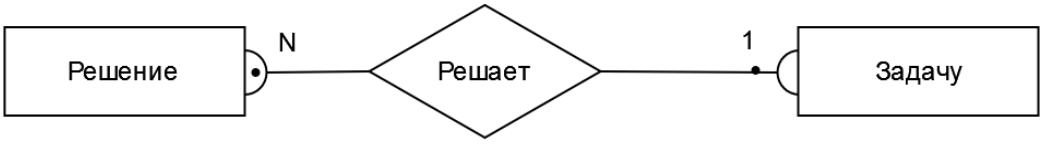


Рисунок 2.2 – ER-Диаграмма связи Решение решает Задачу

3. Связь Задача содержит Тест

Для степени связи:

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

Для класса принадлежности сущности к связи:

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

Рисунок 2.3 – ER-Диаграмма связи Задача содержит Тест

4. Связь Задача включена в Соревнование

Для степени связи:

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

Для класса принадлежности сущности к связи:

- Задача необязательно включена Соревнование;
- Соревнование необязательно содержит Задачи;

Рисунок 2.4 – ER-Диаграмма связи Задача включена в Соревнование

5. Связь Компания организует Соревнование

Для степени связи:

- Компания организует множество Соревнований;
- Соревнование организуется одной Компанией;

Для класса принадлежности сущности к связи:

- Компания необязательно организует Соревнование;
- Соревнование обязательно организуется Компанией;

Рисунок 2.5 – ER-Диаграмма связи Компания организует Соревнование

6. 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 2.6 – ER-Диаграмма связи Пользователь участвует в Соревновании

7. Связь Пользователь администрирует Компанию

Для степени связи:

- Пользователь администрирует множество Компаний;

- Компанию администрирует множество Пользователей;

  Для класса принадлежности сущности к связи:

- Пользователь необязательно администрирует Компанию;
- Компанию обязательно администрирует Пользователь;

Рисунок 2.7 – ER-Диаграмма связи Пользователь администрирует Компанию

Общая диаграмма представлена на рисунке 2.8.

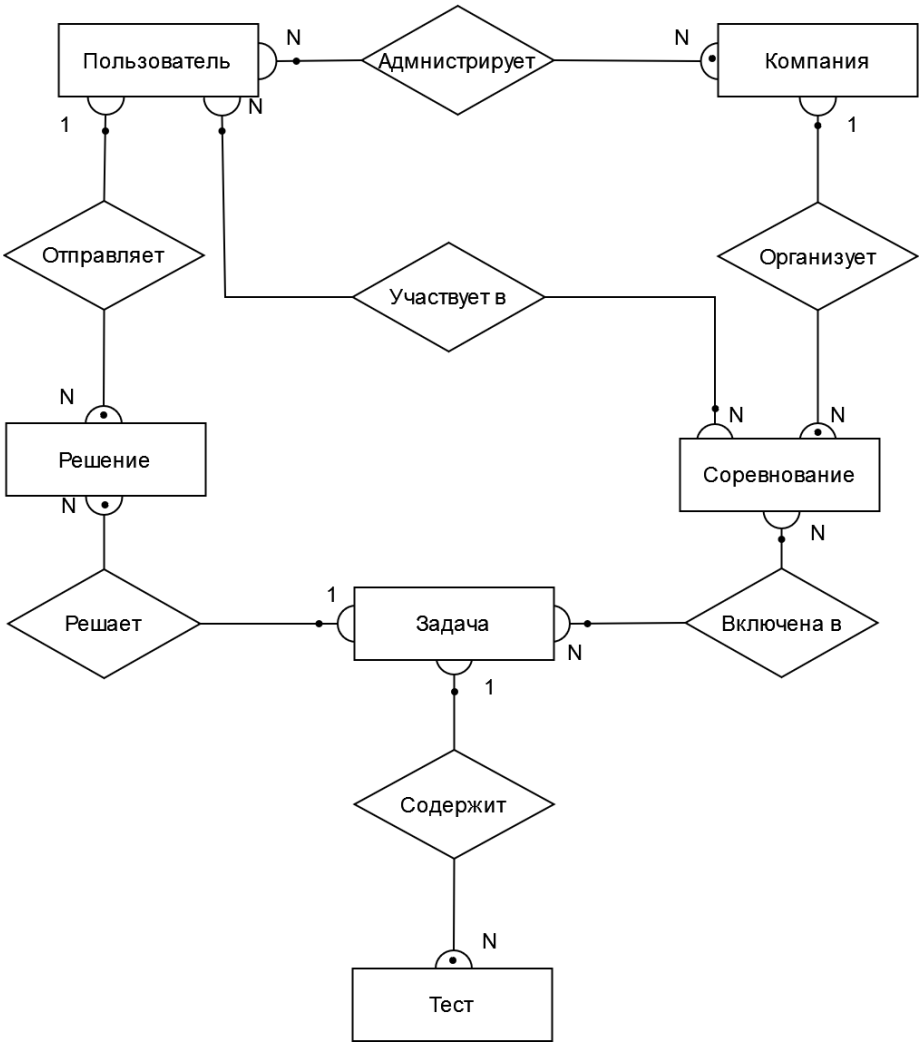


Рисунок 2.8 – Общая ER-Диаграмма

2.3 Даталогическое проектирование Базы Данных

2.3.1 Формирование предварительных отношений по ER-диаграмме

1) Пользователь создает Решение:

- Пользователь (user_id)
- Решение (attempt_id, user_id)

2) Решение решает Задачу:

- Задача (task_id)
- Решение (attempt_id, task_id)

3) Задача содержит Тест

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

4) Задача включена в Соревнование

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

5) Компания организует Соревнование

- Компания (company_id)
- Соревнование (competition_id, company_id)

6) Пользователь участвует Соревнование

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

7) Пользователь администрирует Компанию

- Пользователь (user_id)
- Компания (company_id)
- Пользователь_Компания (user_id, company_id)

2.3.2 Подготовка списка атрибутов. Распределение их по отношениям

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

2) Задача (task) - task_id, title, description, input_types, output_type, languages, is_enabled, method_name, is_private, level, tags, time_limit, created_at

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

4) Решение (attempt) – attempt_id, user_id, task_id, status, code, language, execution_time, error_message, actual_result, test_id, created_at

5) Компания (company) – company_id, title, description, created_at

6) Пользователь_Компания (user_id, company_id)

7) Соревнование (competition) – competition_id, title, description, start_at, created_at, is_private, company_id

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

9) Пользователь_Соревнование (user_competition) – user_id, competition_id, result_points

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 2.9.

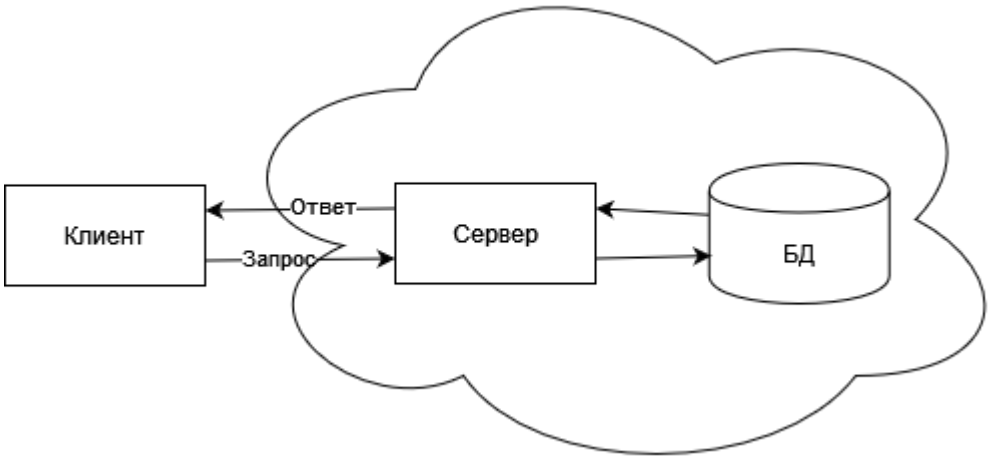


Рисунок 2.9 – Клиент-серверная архитектура

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 2.10.

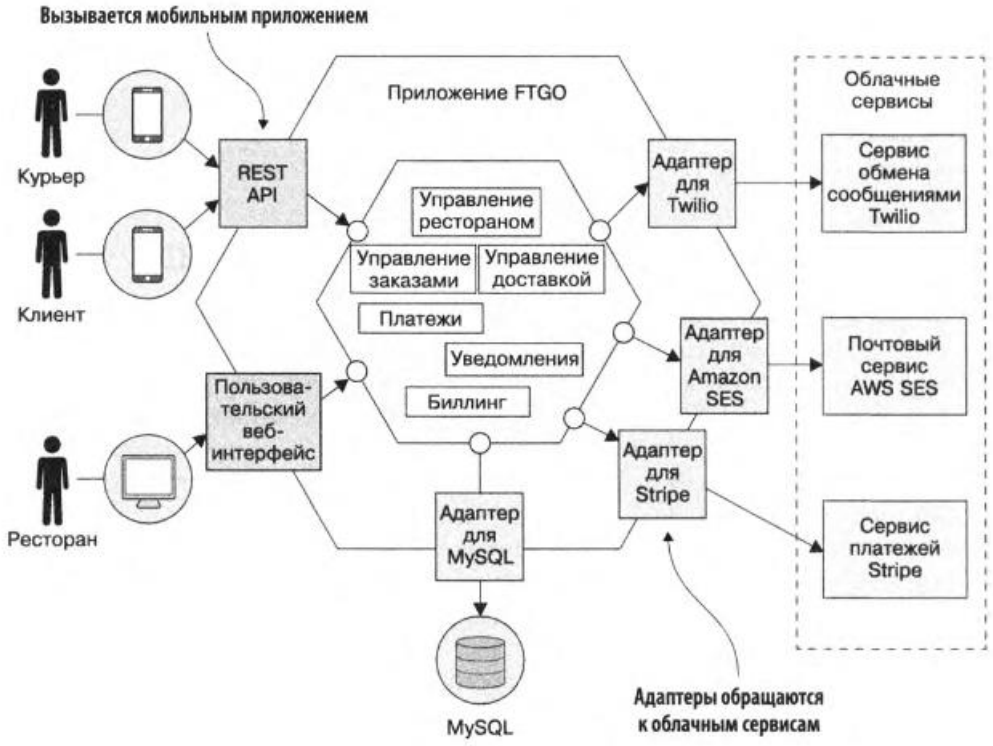


Рисунок 2.10 – Монолитное приложение

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. 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. Отдельные базы данных: каждый микросервис может использовать свою собственную базу данных, что обеспечивает изоляцию данных между сервисами.

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

6. 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 2.11 представлено приложение из рисунка 2.10 переделанное с использованием микросервисов.

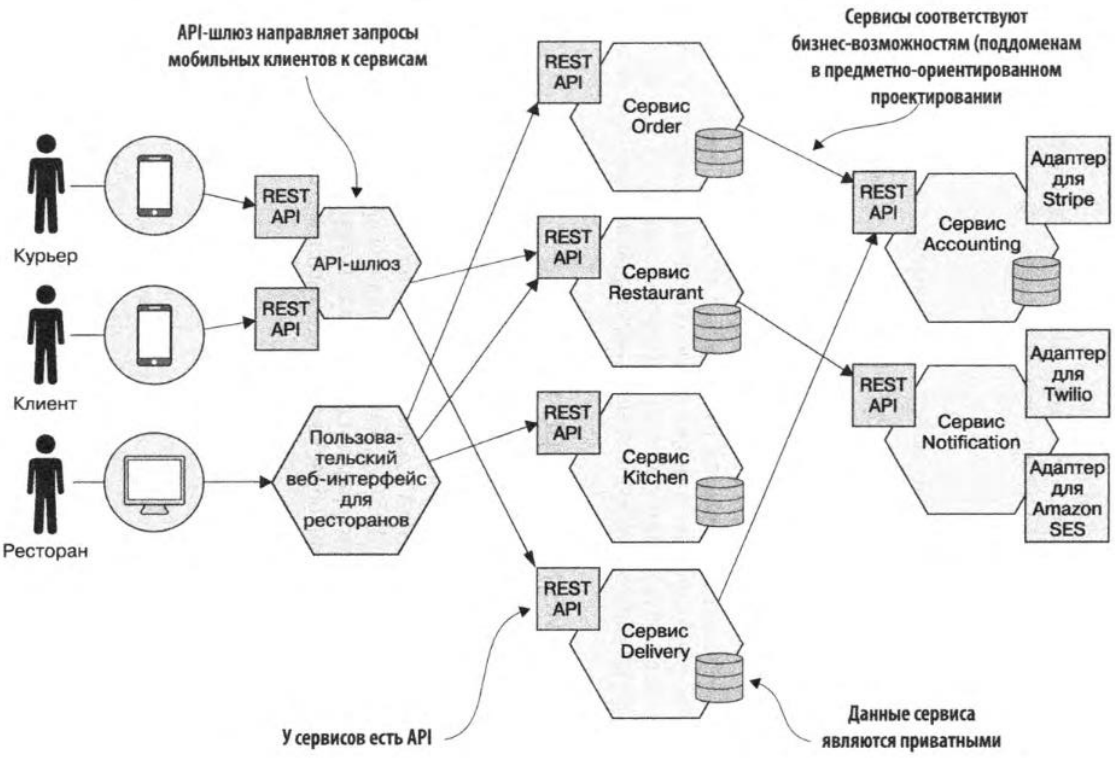


Рисунок 2.11 – Монолит, переделанный в микросервисы

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 2.12 представлена архитектура платформы.

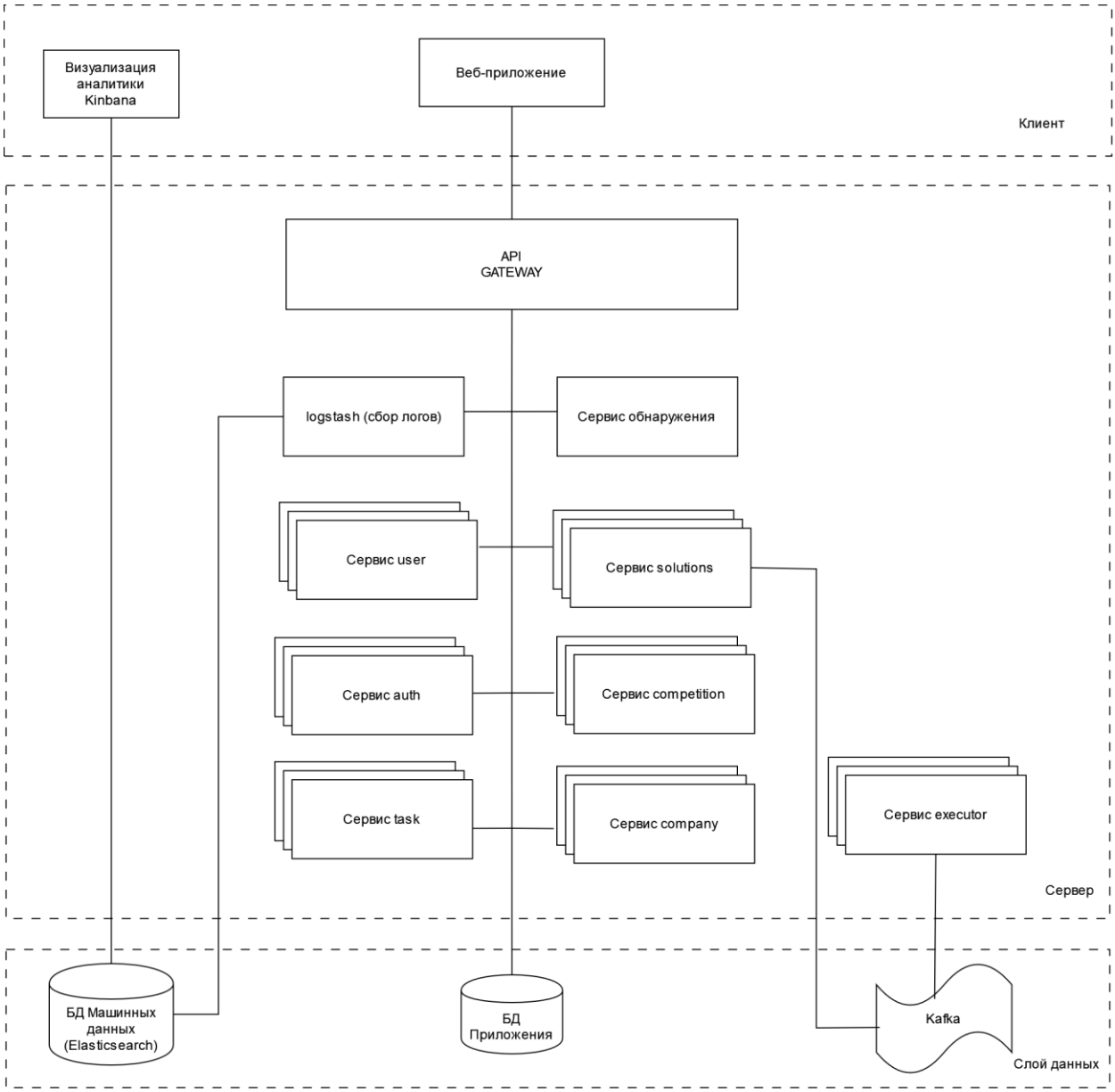


Рисунок 2.12 – Монолит, переделанный в микросервисы

Сервисы user, api, auth, company-api, runner – 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. Для балансировки нагрузки между экземплярами используется балансировка со стороны вызывающего. 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.

Сервис обнаружения позволяет регистрировать экземплярам сервисов свой адрес в едином месте, по которому доступен конкретный экземпляр. Так же полученная таблица доступа распространяется между сервисами. Благодаря сервису обнаружения становится возможной балансировка нагрузки.

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.

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.

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.

Топик для вызова операции выполнения кода – codest.runner.request. Контракт топика представлен на рисунке 3.1.1.

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 3.1.1 – Контракт топика codest.runner.request

Топик для результата тестирования – codest.runner.response. Контракт топика представлен на рисунке 3.1.2.

RunCodeResponseEvent

Описание

Название поля	Тип	Обязательность	Описание
errorType	CodeRunnerErrorType	-	Информация об ошибке/ Непройденный 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 3.1.2 – Контракт топика codest.runner.response.

Алгоритм выполнения операции RunCodeOperation представлен на рисунках 3.1.3 – 3.1.4

RunCodeOperation - запуск кода

Предназначена для компиляции, запуска кода и проверки решения.

Алгоритм

1. Прочитать из топика **codest.runner.request** и декодировать в соотвествии 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:
        a. 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** из запроса, полученый в 3.i или 1.i
    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	input + errors

Рисунок 3.1.3 – Алгоритм RunCodeOperation пункты 1-5.

6. **Иначе**
   i. Сопоставить массив output и ожидаемые ответы из request. Если значения не совпадают:
      a. Положить в топик **codest.runner.response** сообщение по контракту и **key = event.key**:

Поле	**Значение**
errorType	TEST_ERROR
output	List.of(expected, actual)

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

Поле	**Значение**
errorType	null
output	result

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**
errorType	TIME_EXCEED_ERROR
output	"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**
errorType	INTERNAL_ERROR
output	Сообщение из исключения

Рисунок 3.1.4 – 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

Контракт представлен на рисунках 3.2.1.1 – 3.2.1.2

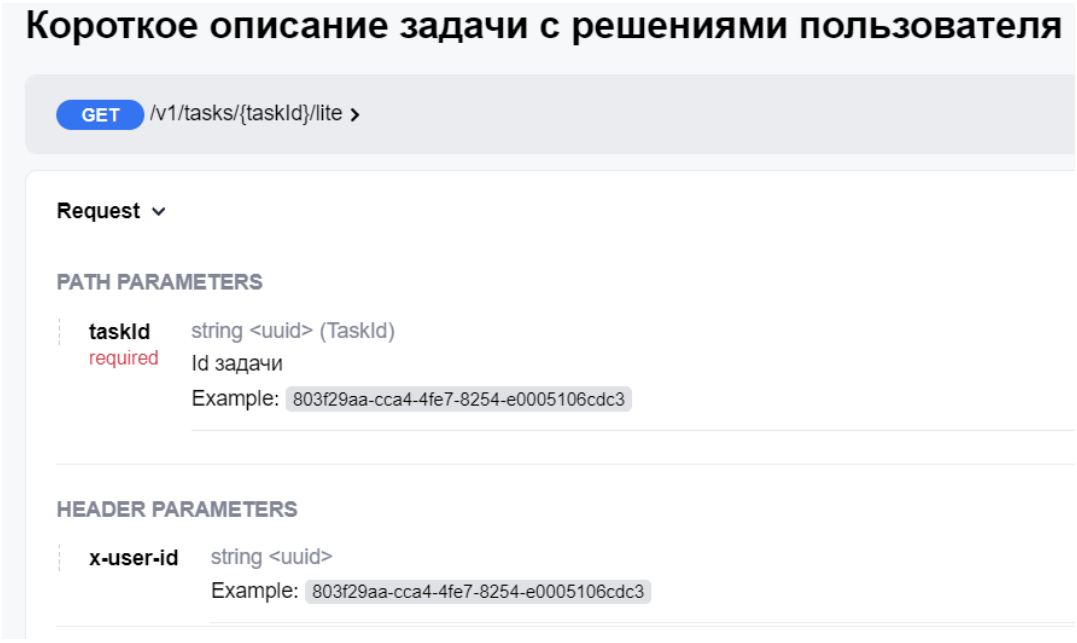


Рисунок 3.2.1.2 – Путь к вызову операции и входные параметры.

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

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 3.2.1.2 – Схема результата.

Алгоритм выполнения операции представлен на рисунке 3.2.1.3.

GetTaskLite - получение задачи для фронта

Параметры

Имя	Тип	Обязательность
taskId	path	+
userId	header	-

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. Вернуть ответ в соответсвии мапингу из openApi

Рисунок 3.2.1.3 – Схема результата.

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. Высконагруженные приложения. Программирование, масштабирование, поддержка. – СПб.: Питер, 2023. – 640 с.

5) Harihara S. Hands-On RESTful API Design Patterns and Best Practices – Birmingham: Packt, 2019. – 360c.

6) Документация Kotlin [Электронный ресурс]. URL: https://kotlinlang.org/docs/home.html.

7) Spring | Home [Электронный ресурс]. URL: https://spring.io/.

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) Spring Boot Docs [Электронный ресурс]. URL: https://docs.spring.io/spring-boot/docs/current/reference/html/.

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. – 325c.

15) Freeman S., Pryce N. Growing Object-Oriented Software, Guided by Tests. – Boston: Addison-Wesley, 2010. – 385c.

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.