**Содержание**

Введение 8

1 Аналитический обзор 10

1.1 Обзор предметной области 10

1.2 Аналитический обзор механизма автоматизированной проверки решений 11

1.3 Аналитический обзор механизма изолированного запуска программ 12

1.4 Обзор существующих решений в области обучающих веб-сервисов с возможностью автоматической проверки решений задач 13

1.5 Постановка задачи 17

1.6 Вывод по главе 18

2 Проектирование онлайн-платформы по решению алгоритмических задач 19

2.1 Проектирование структур данных 19

2.3 Проектирование базы данных 21

2.4 Проектирование микросервисной архитектуры приложения 23

2.5 Проектирования алгоритмов 27

2.6 Выводы по главе 30

3 Программное конструирование онлайн-платформы по решению алгоритмических задач 31

3.1 Выбор средств разработки 31

3.2 Описание модульной структуры программного средства 32

3.3 Описание структуры хранения контента онлайн-платформы 38

3.4 Описание клиентской части программного обеспечения 43

3.5 Выводы по главе 48

4 Демонстрация онлайн-платформы «SupremeCode» 49

4.1 Описание процесса демонстрации 49

4.2 Вывод по главе 55

5 Экономическое обоснование разработки онлайн-платформы «SupremeCode» 57

5.1 Основные аспекты реализации работы 57

5.2 Характеристика программного продукта с позиции маркетинга 57

5.3 Стратегический маркетинговый анализ целесообразности применения программного продукта 59

5.4 Определение продолжительности работ по разработке программного продукта 59

5.5 Определение затрат на разработку и внедрение программного продукта 62

5.7 Выводы по главе 66

6 Безопасность и экологичность выпускной квалификационной работы 67

6.1 Анализ вредных и опасных факторов, влияющих на здоровье программиста 67

6.2 Расчет системы искусственного освещения помещений 68

6.3 Вывод по главе 72

Заключение 74

Перечень использованных информационных ресурсов 75

Приложение А Техническое задание 79

Приложение Б Диаграмма последовательности обработки запроса на проверку решения 87

Приложение В Листинг программы 88

Введение

В современном мире информационные технологии играют ведущую роль в развитии различных отраслей экономики и науки. Следовательно, спрос на IT-специалистов, включая разработчиков программного обеспечения, постоянно растет. В связи с этим, возникает необходимость в подготовке новых специалистов в области программирования.

Онлайн-платформы с алгоритмическими задачами по программированию становятся неотъемлемой частью этого процесса подготовки. Эти платформы предоставляют студентам и начинающим программистам возможность развивать свои навыки, решая разнообразные задачи с различным уровнем сложности. Благодаря таким платформам студенты могут улучшить свои навыки алгоритмизации, научиться эффективно решать задачи и практиковаться в программировании на различных языках.

Более того, онлайн-платформы обеспечивают доступ к обучающим материалам, а также предоставляют возможность соревноваться с другими пользователями, что мотивирует студентов к развитию и позволяет им оценить свой прогресс в сравнении с другими. Такие платформы также часто предлагают системы автоматической проверки решений, что позволяет студентам получать обратную связь и исправлять ошибки, улучшая свои навыки программирования.

Целью данной работы является проектирование и разработка онлайн-платформы с алгоритмическими задачами по программированию.

В рамках данной работы будет нужно выполнить следующие задачи:

1. Анализ предметной области.

2. Обзор подобных информационных систем.

3. Формулирование функциональных требований к разрабатываемой информационной системе.

4. Проектирование программного средства.

5. Программное конструирование программного средства.

Первая глава содержит в себе анализ предметной области и постановку задачи.

Вторая глава содержит описание разработанных структур данных и связей между ними, алгоритмов и архитектуры приложения.

В третьей главе обосновывается выбор программных средств разработки и языка программирования. Представлены описания модулей созданного программного средства.

Четвёртая глава включает в себя результаты тестирования разработанного программного обеспечения.

В пятой главе подсчитаны расходы на разработку, а также подсчитана эффективность разрабатываемой системы. Определена маркетинговая ориентация программного продукта.

Шестая глава определяет безопасность и экологичность продукта выпускной квалификационной работы. Содержит в себе расчёт искусственного освещения помещений.

# 1 Аналитический обзор

В данном разделе приведены сведения о решаемой задаче. Описываются функции онлайн-платформы. Производится обзор существующих платформ для решения алгоритмических задач.

## 1.1 Обзор предметной области

Онлайн-платформы для решения алгоритмических задач стали важным инструментом для разработчиков программного обеспечения, желающих улучшить свои навыки кодирования и алгоритмического мышления. Эти платформы позволяют пользователям решать задачи различной сложности, охватывающие различные аспекты программирования, включая сортировку, поиск, динамическое программирование, структуры данных и математические алгоритмы. Платформы предоставляют доступ к обширной библиотеке задач, часто разделенных по категориям и уровню сложности. Пользователи могут решать задачи на различных языках программирования, что позволяет им тренироваться и развивать навыки на выбранном языке.

Решение задач сопровождается мгновенной оценкой решений: результаты могут включать правильность ответа, эффективность решения и рекомендации по улучшению кода. Платформы также предлагают объяснения решений, советы по оптимизации, и обсуждения, где пользователи могут обмениваться идеями и лучшими практиками. Это создает сообщество разработчиков, поддерживающее рост и обучение каждого из них. Некоторые платформы предлагают систему очков, рейтингов и достижений, стимулирующих пользователей к участию в решении все более сложных задач.

Таким образом, онлайн-платформы для решения алгоритмических задач представляют собой ценные ресурсы для тех, кто стремится улучшить свои навыки программирования, подготовиться к собеседованиям в крупных компаниях и оставаться в курсе новейших тенденций и методов в области разработки программного обеспечения.

## 1.2 Аналитический обзор механизма автоматизированной проверки решений

Автоматизированная проверка решений на онлайн-платформах для решения алгоритмических задач заключается в тестировании каждого отправленного решения по заранее установленному набору автоматизированных тестов. Эти тесты охватывают различные аспекты задачи, включая простые и сложные входные данные, чтобы убедиться, что решение работает корректно и эффективно во всех случаях. Авто-тесты также проверяют, чтобы решение не только выдавал правильный ответ, но и соблюдало ограничения по времени и памяти, заданные в условиях задачи.

Такой подход к проверке позволяет мгновенно и объективно оценить код пользователя. Авто-тестирование также выявляет граничные случаи, которые могут быть неочевидными для разработчика, и помогает избежать ошибок, которые могли бы возникнуть в условиях реальной среды. Если решение пользователя не проходит один или несколько тестов, платформа возвращает соответствующее сообщение о сбое и, в некоторых случаях, предоставляет подсказки или указания о том, что именно пошло не так.

Автоматизированная проверка с прогоном по набору авто-тестов также дает возможность платформам обеспечивать конкурентную и справедливую среду для всех пользователей, гарантируя, что все решения проверяются по одинаковым критериям и стандартам. Это способствует развитию навыков программирования, улучшению качества кода и повышению уровня компетенции пользователей.

## 1.3 Аналитический обзор механизма изолированного запуска программ

Тема безопасности на платформах, где запускается пользовательский код, является особенно актуальной. Запуск пользовательского кода создает потенциальные риски, включая возможность выполнения вредоносного кода и несанкционированного доступа к данным или ресурсам системы. Решением данной проблемы является применение контейнеров приложений. Docker [1] — это платформа для контейнеризации, которая позволяет создавать, развертывать и управлять приложениями в изолированных средах, называемых контейнерами. Контейнер — это легковесный, автономный пакет, содержащий все необходимое для запуска приложения, включая код, библиотеки, зависимости и конфигурации.

Контейнеры предоставляют возможность развертывать приложения в согласованной и предсказуемой среде, независимо от того, где они запускаются — на локальной машине, в облаке или на других серверах. Возможности контейнеров включают изоляцию процессов, файловой системы и сетевых интерфейсов, что позволяет им работать в среде, отделенной от хост-системы и других контейнеров.

Использование контейнеров для изоляции автоматизированной проверки на онлайн-платформах алгоритмических задач может существенно улучшить безопасность и стабильность платформы. Контейнеры позволяют запускать каждое решение пользователя в отдельной изолированной среде, тем самым ограничивая возможные взаимодействия между кодом пользователя и основной инфраструктурой платформы. Это предотвращает потенциальные атаки, такие как доступ к системным файлам хост-системы или запуск вредоносного кода.

Контейнеры также позволяют контролировать ресурсы, такие как процессорное время и память, которые доступны для каждого контейнера. Это обеспечивает справедливые условия для всех пользователей и предотвращает злоупотребление ресурсами. Изолированная среда контейнера также позволяет платформе применять ограничения безопасности, такие как запрет на определенные системные вызовы, что дополнительно снижает риск вредоносных действий.

В целом, использование контейнеров для изоляции автоматизированной проверки на онлайн-платформах алгоритмических задач повышает безопасность и надежность платформы, обеспечивая при этом справедливую и предсказуемую среду для всех пользователей. Это способствует защите данных и ресурсов платформы, а также улучшает опыт работы пользователей.

## 1.4 Обзор существующих решений в области обучающих веб-сервисов с возможностью автоматической проверки решений задач

На сегодняшний день существует множество онлайн-платформ для решения алгоритмических задач и проверки знаний в области программирования. Одной из самых известных и популярных платформ на международном уровне является «LeetCode» [2], а в России — «Информатикс» [3].

Эти две платформы значительно отличаются по своим подходам и целевым аудиториям. «Информатикс» сосредоточена на дистанционном обучении в области информатики и предлагает разнообразные курсы, задачи и тесты для школьников и студентов, а также для учителей. Эта платформа ориентирована на обучение и развитие студентов, позволяя преподавателям контролировать и отслеживать результаты учащихся.

В отличие от «Информатикс», «LeetCode» поддерживает большее количество языков программирования, что делает платформу более доступной для различных категорий программистов. В то же время сложность задач является более высокой, что увеличивает порог входа и уменьшает возможности решения слабо подготовленным студентам. Также отмечается формат задач: в частности, на платформе «Информатикс» менее обширное описание, нет статистики, задачи не делятся по тегам, напротив, в пользу данной платформы говорит наличие сопутствующих теоретических материалов в помощь студентам.

Важным преимуществом «LeetCode» является наличие встроенного редактора с подсветкой синтаксиса и возможностью тестирования программы на предоставленных данных, что облегчает процесс отладки и проверки кода. В отличие от этого, «Информатикс» предоставляет лишь кнопку для загрузки файла, что вынуждает пользователя писать и отлаживать код локально.

Также стоит упомянуть платформу «Codewars» [4], которая предлагает задачи для программистов всех уровней, позволяя участникам повышать свои навыки посредством обучения через решение задач, созданных сообществом. Подобно «LeetCode», «Codewars» поддерживает различные языки программирования и позволяет пользователям сравнивать свои решения с решениями других участников, что способствует улучшению своих навыков. Более того, «Codewars» предоставляет возможность создавать свои собственные задачи, что привлекает более опытных программистов и дает возможность сообществу активно участвовать в создании новых задач для платформы.

С точки зрения пользовательского интерфейса, «Информатикс» не дотягивает до современных стандартов, так как его дизайн устарел, а главная страница переполнена информацией, затрудняющей навигацию. Поиск задач также затруднен, поскольку они обозначены номерами, которые часто известны только преподавателям. Интерфейс страницы задачи представлен на рисунке 1.1.

В отличие от него, «LeetCode» предлагает современный дизайн и тщательно продуманный список задач с различными возможностями сортировки, тегирования и поиска по темам. Это упрощает пользователям поиск необходимых задач и улучшает общее взаимодействие с платформой. Интерфейс страницы задачи представлен на рисунке 1.2.



Рисунок 1.1 – Платформа «Информатикс»



Рисунок 1.2 – Платформа «LeetCode»

«Codewars» также отличается удобным пользовательским интерфейсом, ориентированным на задачи для программистов всех уровней. Платформа предоставляет возможность поиска задач по разным уровням сложности, языкам программирования и темам. Кроме того, интерфейс предоставляет возможность взаимодействия с сообществом путем обсуждения задач и решений, что повышает качество обучения и способствует обмену знаниями между участниками. Интерфейс страницы задачи представлен на рисунке 1.3.



Рисунок 1.3 – Платформа «CodeWars»

Платформа «LeetCode» имеет значительное преимущество перед другими платформами благодаря наличию онлайн-компилятора (playground), который позволяет выполнять пользовательские программы вне решения задач. Этот инструмент предоставляет разработчикам гибкость и свободу для экспериментов с кодом в интерактивной среде, что повышает качество обучения и развития навыков.

Онлайн-компилятор позволяет пользователям проверять, тестировать и оптимизировать свои алгоритмы и решения без давления или ограничений конкретной задачи. Это дает возможность разработчикам работать над собственными проектами или идеями, применять новые концепции и подходы к программированию, а также исследовать различные языки программирования.

Интерактивный характер онлайн-компилятора позволяет пользователям сразу видеть результаты выполнения своего кода, что ускоряет процесс обучения и отладки. Это также позволяет пользователям проверять эффективность и производительность своего кода, сравнивая различные подходы к решению задач.

Кроме того, наличие онлайн-компилятора помогает пользователям развивать навыки программирования в более широком контексте, чем просто выполнение конкретных задач. Это стимулирует творчество и инновации, позволяя пользователям экспериментировать и исследовать новые способы решения проблем.

Еще одной интересной возможностью является совместная работа над решением задачи в реальном времени. Это позволяет пользователям обмениваться идеями, исправлять ошибки друг друга и коллективно находить наилучшие решения.

При использовании функции совместного редактирования пользователи могут одновременно просматривать, редактировать и анализировать код друг друга. Это способствует обмену опытом и знаниями, а также повышает эффективность работы в команде.

Преимущества совместного редактирования кода включают возможность мгновенного обратного связи, ускорение процесса разработки и улучшение качества кода за счет совместного обсуждения и корректировки. Кроме того, это помогает развивать навыки командной работы и обучаться новым подходам к решению задач.

На данный момент ни одна из рассматриваемых платформ не имеет возможности совместного редактирования в реальном времени.

## 1.5 Постановка задачи

Проведенный анализ существующих образовательных веб-сервисов с автоматизированной проверкой задач выявил, что, несмотря на наличие основного функционала, они также имеют ряд недостатков. У «LeetCode» и «Codewars» отсутствует поддержка русского языка. На «Информатикс» выбор языков программирования ограничен и интерфейс может быть неудобным для пользователя. На «LeetCode» и «Информатикс» нет возможности добавить собственную задачу. На «Информатикс» и «Codewars» не возможности запустить программный код вне решения задачи.

Недостатки существующих платформ достаточно значительны и подчеркивают необходимость создания собственной онлайн-платформы «SupremeCode».

Цель разрабатываемой платформы – предоставить пользователям возможность учиться программированию и информационным технологиям с возможностью практического решения задач, которые будут автоматически проверяться.

Перечень задач для реализации:

* исследование платформ с алгоритмических задачами;
* проектирование архитектуры онлайн-платформы;
* разработать серверное приложение, обрабатывающее запросы пользователей и взаимодействующее с базой данных;
* реализовать интерфейс, который позволит пользователю взаимодействовать с системой и решать задачи.

## 1.6 Вывод по главе

В этой главе были изучены случаи, когда подобная онлайн-платформа окажется полезной. Проведено исследование аналогичных программных продуктов. Рассмотрена потребность в платформе «SupremeCode». Поставлена цель и сформулированы задачи для реализации онлайн-платформы.

# 2 Проектирование онлайн-платформы по решению алгоритмических задач

В данной главе описано проектирование программного средства, включающее в себя проектирование структур данных, базы данных, алгоритмов и конечных точек программного интерфейса приложения.

## 2.1 Проектирование структур данных

Данный раздел содержит в себе описание необходимых для реализации программного средства структур данных.

### 2.2.1 Описание структуры «Задача»

Задача является центральной сущностью в онлайн-платформах по решению алгоритмическим задач. Подробное описание данных о задаче представлено в таблице 2.1.

Таблица 2.1 – Описание структуры «Задача»

|  |  |  |
| --- | --- | --- |
| Название | Описание | Примечание |
| 1 | 2 | 3 |
| slug | Уникальный идентификатор задачи | Строка, однозначно определяет конкретную задачу на платформе |
| name | Название задачи | Обычно краткое и описательное |
| description | Описание задачи на языке разметки Markdown, включающее условия задачи, примеры ввода и вывода, а также дополнительные разъяснения и рекомендации | Включает ограничения и требования к эффективности |
| difficulty | Уровень сложности задачи | Уровень сложности один из нескольких: "Easy", "Normal" или "Hard" |
| languages | Список языков программирования, на которых пользователь может решать данную задачу | Например, "Cpp", "Java", "Javascript" и другие |
| tests | Код с автоматизированными тестами для проверки корректности решения | Для каждого языка указывается свой файл с тестами |
| solution | Шаблонный код для программирования решения | Не должен содержать решения задачи |

### 2.2.2 Описание структуры «Решение»

Структура «Решение» представляет собой пользовательский вариант решения задачи. Данная структура будет использована для долговременного хранения отправленных пользователем решений. Описание структуры данных решения задачи представлено в таблице 2.2.

Таблица 2.2 – Описание структуры «Решение»

|  |  |  |
| --- | --- | --- |
| Название | Описание | Примечание |
| 1 | 2 | 3 |
| problem\_slug | Уникальный идентификатор задачи | Строка, однозначно определяет конкретную задачу на платформе |
| language | Язык, который выбрал пользователь | Например, "Cpp", "Java", "Javascript" и другие |
| user\_id | Уникальный идентификатор пользователя | Число, однозначно определяет конкретного пользователя на платформе |
| code | Решение пользователя | Текст, программный код на выбранном языке |
| created\_at | Дата и время отправки решения | Заполняется автоматически |

### 2.2.3 Описание структуры «Результаты решения»

Решение, отправленное пользователем, и результат его проверки могут быть сильно разделены по времени создания. Имеет смысл рассмотреть их раздельно. Описание данной структуры представлено в таблице 2.3.

Таблица 2.3 – Описание структуры «Результаты решения»

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Описание | | Примечание |
| 1 | 2 | | 3 |
|  |  | |  |
| solution\_id | Уникальный идентификатор решения, связывает результат с конкретным решением | | Не может быть нулевым |
| tests | Общее количество тестов, по которым проверялось решение | | Не может быть нулевым |
| failures | Количество тестов, которые завершились неудачно | | Не может быть нулевым |
| errors | Количество ошибок, возникших во время выполнения решения | | Не может быть нулевым |
| created\_at | | Дата и время отправки решения | Заполняется автоматически |

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| status\_code | Код статуса выполнения решения | Может содержать коды, указывающие на успех, неудачу или ошибку |
| time | Время, затраченное на выполнение решения | Значение с плавающей точкой, указывает на время в секундах |
| logs | Лог-файл с деталями выполнения решения, включая возможные ошибки или предупреждения | Может содержать текстовое описание проблем |
| junit\_xml | XML-отчет, форматированный в стиле JUnit, содержит результаты тестирования | Требует дальнейшей обработки для вычисления значения solved |
| solved | Индикатор того, решена ли задача успешно | Логическое значение |

### 2.2.4 Описание структуры «Пользователь»

Структура пользователя довольно простая, ее описание представлено в таблице 2.4. Данная структура необходима для хранения информации о пользователе платформы и отслеживания прогресса решения задач.

Таблица 2.4 – Описание структуры «Пользователь»

|  |  |  |
| --- | --- | --- |
| Название | Описание | Примечание |
| 1 | 2 | 3 |
|  |  |  |
|  |  |  |
| id | Уникальный идентификатор пользователя | Число, однозначно определяет конкретного пользователя на платформе |
| username | Имя пользователя, используемое для входа и отображения в сервисе | Не может быть нулевым, уникальное значение |
| password | Хэшированный пароль пользователя | Не может быть нулевым |
| image | Ссылка на изображение профиля пользователя | Может быть нулевым, длина не должна превышать 255 символов |

## 2.3 Проектирование базы данных

Решено использовать реляционную модель данных для хранения информации в системе. В реляционной модели отдельные структуры данных описываются отношениями.

В этом разделе рассмотрены основные таблицы базы данных, такие как: пользователь, задача, решение и результат решения. А также таблицы с пользовательской статистикой и по платформе в целом.

Отношение «users», представляющее пользователей, связано с отношением «solution», представляющим решения пользователя. Связь имеет вид «один ко многим».

Отношение «solution», представляющее решение, связано с отношением «problem», представляющим задачу. Связь имеет вид «один ко многим».

Отношение «solution\_result», представляющее результаты проверки решений, связано с отношением «solution» связью вида «один к одному». Каждому решению соответствует единственный результат.

Набор отношений с именами, начинающимися с general\_statistics\_ содержит общую статистику по решению задач на платформе и включает в себя:

* general\_statistics\_topSolved — отношение, содержащее список наиболее часто решаемых задач.
* general\_statistics\_topAttempted — отношение, содержащее список задач, на которые пользователи пытались найти решение.
* general\_statistics\_topAttemptedNotSolved — отношение, содержащее список задач, на которые пользователи пытались найти решение, но не смогли.
* general\_statistics\_difficultyCounts — отношение, содержащее количество решенных задач по каждому уровню сложности.
* general\_statistics\_languageCounts — отношение, содержащее количество задач, решенных на каждом языке программирования.

Набор отношений с именами, начинающимися с user\_statistics\_ содержит статистику по пользователям и включает в себя:

* user\_statistics\_difficulty — отношение, содержащее количество задач по каждому уровню сложности для конкретного пользователя.
* user\_statistics\_language — отношение, содержащее количество задач, решенных на каждом языке программирования для конкретного пользователя.
* user\_statistics\_solved — отношение, содержащее количество решенных задач и попыток решения задач для конкретного пользователя.

Диаграмма, описывающая схему базы данных представлена на рисунке 2.1.

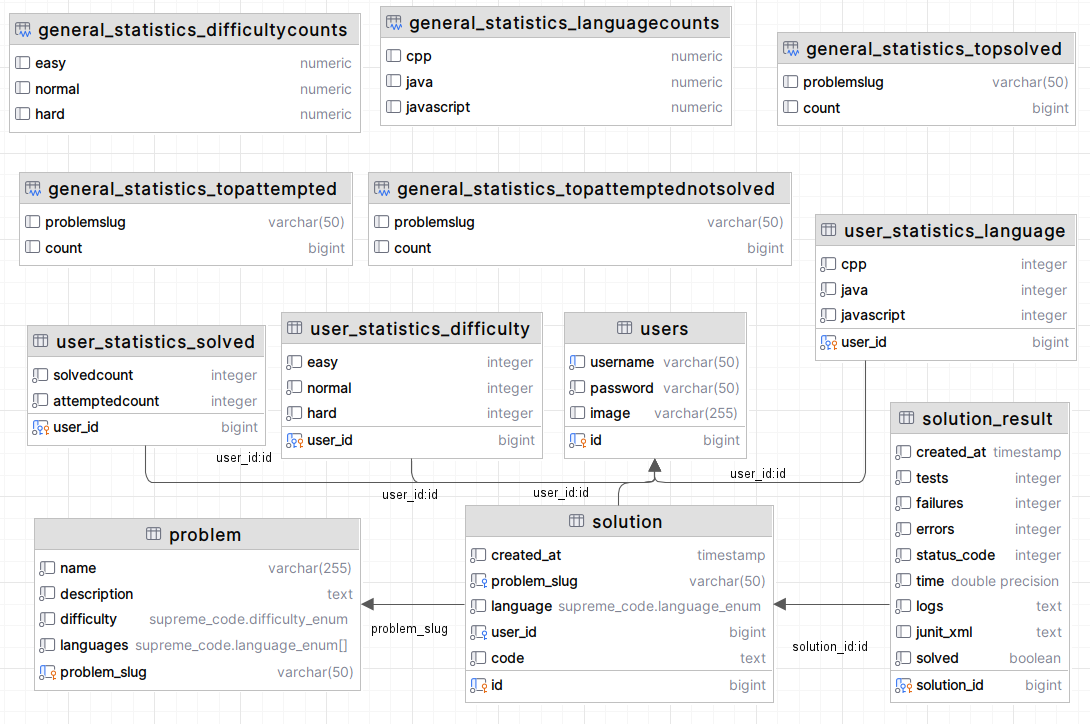


Рисунок 2.1 – Диаграмма структуры базы данных

## 2.4 Проектирование микросервисной архитектуры приложения

Для разрабатываемого средства была выбрана микросервисная архитектура, позволяющая проектировать небольшие, малосвязанные модули-микросервисы.

Решено разделить приложение на три связанных между собой микросервиса: веб-сервер, «task-runner», «test-runner». На рисунке 2.2 представлена диаграмма компонентов онлайн-платформы, включающая названные микросервисы, очередь сообщений, реляционную базу данных и клиентское приложение.

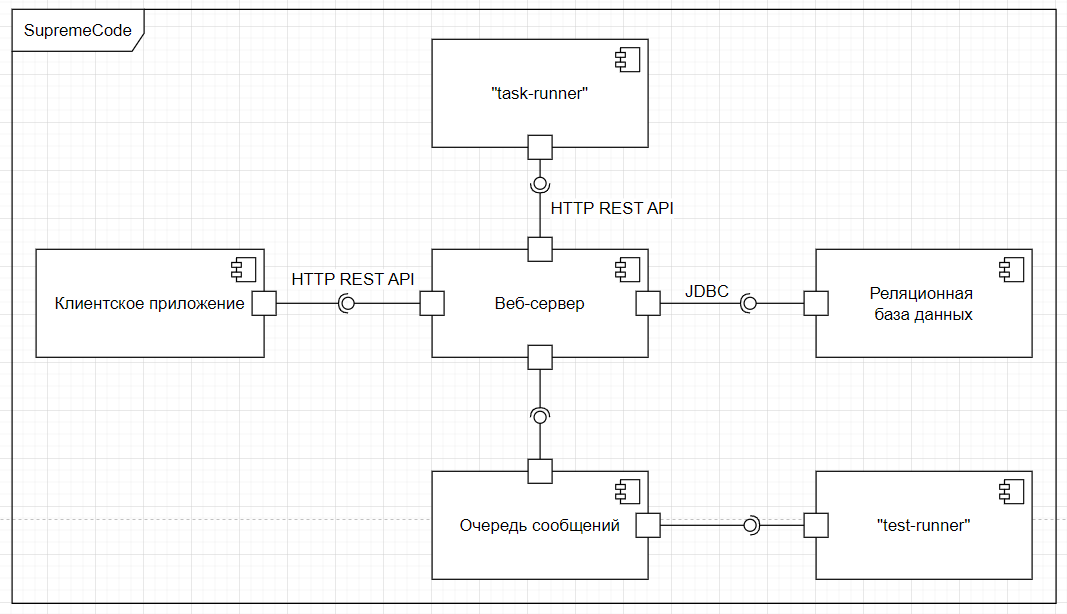


Рисунок 2.2 – Диаграмма компонентов онлайн-платформы

### 2.4.1 Проектирования центрального веб-сервера

Веб-сервер является центральным элементом системы и выступает в роли единой точки входа для всех внешних запросов. Он выполняет задачи авторизации и аутентификации пользователей, обеспечивая безопасный доступ к сервису. Веб-сервер обрабатывает запросы от браузера, предоставляя необходимые данные для отображения страниц, включая информацию о задачах, решениях, результатах и других аспектах онлайн-платформы.

Веб-сервер также отвечает за предоставление статистики по онлайн-платформе, что включает в себя метрики и аналитические данные о пользователях, их активности и результатах решения задач. Взаимодействуя с базой данных, веб-сервер обеспечивает доступ к данным, необходимым для работы платформы.

Кроме того, веб-сервер управляет взаимодействием с другими сервисами, такими как сервис тестирования и сервис запуска программ. Веб-сервер отправляет запросы с решениями сервису тестирования и получает обратно результаты. Веб-сервер также отправляет запросы сервису запуска программ, передавая задачи на выполнение и получая результаты их работы. Таким образом, веб-сервер играет ключевую роль в координации и интеграции различных компонентов системы.

### 2.4.2 Проектирования сервиса «test-runner»

Сервис «test-runner» отвечает за обработку запросов на проверку пользовательских решений. Он получает запросы из очереди сообщений, что обеспечивает асинхронное взаимодействие с веб-сервером. Благодаря использованию очереди, при отказе одного из сервисов решение пользователя не будет утеряно, так как запросы остаются в очереди для обработки позднее. Это снижает зависимость от времени выполнения задач, позволяя веб-серверу передавать запросы на проверку и сразу же продолжать работу, не ожидая завершения тестирования. Такая архитектура повышает производительность и устойчивость системы, а также обеспечивает более эффективное распределение нагрузки.

В качестве очереди сообщений выбрана Apache Kafka [5]. Очередь сообщений Kafka способна обрабатывать огромные объемы сообщений и поддерживать высокую пропускную способность. Kafka позволяет разделить темы на разные партиции, что обеспечивает параллельную обработку и распределение нагрузки. Для платформы, где важна масштабируемость и быстродействие, это значимое преимущество. Более того, Kafka имеет встроенные возможности для стриминговой обработки данных через Kafka Streams. Хотя в настоящее время не планируется использовать возможности стриминговой обработки данных, в будущем такая необходимость может возникнуть. Kafka может быть сложнее в развертывании и требует некоторого времени для изучения, но ее мощные возможности делают ее привлекательным выбором для крупных и высоконагруженных систем.

После получения запроса сервис «test-runner» запускает тестирование решения в изолированном окружении, таком как Docker контейнер, обеспечивая безопасность и изоляцию выполнения кода пользователя. После выполнения тестирования сервис возвращает полученный результат обратно в Kafka для дальнейшей обработки.

Сервис «test-runner» может быть независимо горизонтально масштабирован при увеличении нагрузки на платформу. Это позволяет динамически добавлять или уменьшать количество экземпляров сервиса в зависимости от потребностей, повышая общую производительность и устойчивость системы. Благодаря этому, система останется эффективной даже при больших объемах запросов или изменениях в нагрузке.

### 2.4.3 Проектирования сервиса «task-runner»

Сервис «task-runner» отвечает за запуск пользовательских программ и возвращение результатов их выполнения. Он взаимодействует с веб-сервером посредством REST API [6], обеспечивая эффективный и упорядоченный обмен информацией. После получения запроса от веб-сервера сервис запускает выполнение пользовательской программы в изолированном окружении, таком как Docker контейнер, чтобы обеспечить безопасность и изоляцию работы кода пользователя.

По мере выполнения программы сервис «task-runner» возвращает результат стандартного вывода программы обратно веб-серверу. Такая организация взаимодействия позволяет эффективно и последовательно обрабатывать запросы от веб-серверов, поддерживая стабильное и надежное выполнение задач.

Сервис «task-runner» также может быть независимо горизонтально масштабирован при необходимости, позволяя добавлять или удалять экземпляры сервиса в зависимости от объема запросов. Это повышает производительность и устойчивость системы, особенно при увеличении нагрузки или изменении в потребностях пользователей. Изоляция пользовательских программ в Docker контейнерах обеспечивает безопасность и стабильность выполнения, минимизируя риски, связанные с работой неподконтрольного кода.

## 2.5 Проектирования алгоритмов

Данный раздел содержит в себе описание необходимых для реализации программного средства алгоритмов.

### 2.5.1 Алгоритм обработки запроса на проверку решения

Обработка запроса на проверку пользовательского решения представляет собой многоэтапный процесс. Пользователь отправляет запрос через веб-интерфейс, который поступает на веб-сервер. Веб-сервер взаимодействует с базой данных и передает запрос в очередь сообщений через брокера.

Микросервис «test-runner» извлекает запрос из очереди, запускает проверку в изолированном контейнере и возвращает результаты через брокера сообщений. Эти результаты поступают на веб-сервер для обработки.

Веб-сервер обновляет базу данных и отправляет пользователю информацию о результатах.

Процесс проверки пользовательского решения представлен в виде диаграммы последовательности в приложении В.

### 2.5.2 Алгоритм работы с контейнером при запуске пользовательского кода

Для работы с контейнером с пользовательским кодом требуется свой алгоритм. Контейнер самостоятельно завершает свое выполнение при выполнении всех тестов. Завершить выполнение контейнера может выход времени выполнения за допустимые значения.

Применение ограничения времени выполнения пользовательского кода в контейнере является важным для обеспечения стабильности и безопасности системы. Без ограничений пользовательский код может выполнить долгий или бесконечный цикл, что приведет к избыточному использованию ресурсов, например, процессорного времени. Это может повлиять на производительность системы в целом, замедляя работу других сервисов и вызывая задержки в обработке запросов.

Словесное описание алгоритма запуска пользовательского кода в контейнере:

* контейнер создается, в него копируется решение, контейнер запускается;
* подключается таймер для ожидания превышения времени выполнения, если сработал таймер, то контейнер останавливается;
* подключается логгер, который собирает сообщения и ошибки;
* при завершении работы контейнера, он удаляется.

Диаграмма последовательности предоставляет наглядное представление процесса, позволяющее легче понять и проанализировать асинхронную логику, она представлена на рисунке 2.3.



Рисунок 2.3 – Диаграмма последовательностей для алгоритма работы с контейнером

Ключевые моменты на диаграмме:

* «Test-runner» – сервис по проверке пользовательских решений;
* «Docker container» – контейнер с пользовательским кодом решения, предоставленного для проверки;
* «Logger» – участник, обрабатывающий сообщения от контейнера;
* «Timer» – участник, ожидающий истечения времени;
* «Поток» – под «потоком» будем понимать объект, который принимает сообщения и ошибки. Его можно преждевременно завершить, например, при появлении ошибки или при выходе времени выполнения за допустимое значение.

## 2.6 Выводы по главе

В данной главе была спроектирована онлайн-платформа «SupremeCode».

Были спроектированы и описаны такие структуры данных, как Задача, Решение, Результаты решения, Пользователь. Было произведено проектирование базы данных. Описаны отношения и связи между ними.

Спроектированы микросервисы: по проверке решений, по запуску кода вне решения задач и веб-сервер. Были описаны используемые алгоритмы.

# 3 Программное конструирование онлайн-платформы по решению алгоритмических задач

В данном разделе приведено обоснование выбора средств разработки, используемых для создания онлайн-платформы. Описана система управления базой данных. Представлены описания интерфейсов и классов, их методов.

## 3.1 Выбор средств разработки

Для разработки программного средства было принято решение использовать язык программирования Java [7] на бэкенде с фреймворком Spring [8]. Spring — это фреймворк для разработки приложений на языке Java, который обеспечивает комплексный подход к построению современных веб-приложений. Его модульная архитектура и множество готовых инструментов позволяют ускорить разработку, обеспечить надежность и масштабируемость приложений.

На фронтенде используется язык программирования TypeScript [9] с библиотекой React [10]. TypeScript — это язык программирования, который представляет собой надстройку над JavaScript, добавляющую статическую типизацию и другие возможности, что делает код более надежным и понятным. React — это библиотека JavaScript для создания пользовательских интерфейсов, которая позволяет строить масштабируемые и переиспользуемые компоненты интерфейса.

В качестве системы управления базами данных выбрана PostgreSQL [11]. PostgreSQL — это мощная объектно-реляционная система управления базами данных, которая обеспечивает надежное хранение и эффективное управление данными. Она отличается высокой производительностью, расширяемостью и поддержкой широкого спектра функциональных возможностей.

Для обеспечения асинхронной коммуникации между компонентами системы используется брокер сообщений Apache Kafka. Kafka — это распределенная система потоковой обработки данных, которая обеспечивает надежную и масштабируемую передачу сообщений между различными компонентами приложения. Использование Kafka позволяет обеспечить надежную и эффективную передачу данных в реальном времени и упростить интеграцию между сервисами.

Для разработки программного средства также используется интегрированная среда разработки IntelliJ IDEA Community Edition [12]. IntelliJ IDEA — это интегрированная среда разработки (IDE) для языка программирования Java. Она предоставляет широкий набор инструментов для удобной и эффективной разработки, включая поддержку автоматической компиляции, интеллектуальные подсказки, отладку, рефакторинг и многое другое. Использование IntelliJ IDEA обеспечивает комфортное и продуктивное программирование.

Для управления версиями и совместной разработки кода применяется система контроля версий Git [13]. Git — это распределенная система управления версиями, которая позволяет отслеживать изменения в коде, вести историю разработки, совместно работать над проектами и управлять кодом в различных ветках. Использование Git обеспечивает прозрачность и контроль над изменениями в коде, а также упрощает процесс совместной разработки и интеграции изменений.

## 3.2 Описание модульной структуры программного средства

Программной код онлайн-платформы разделен на множество модулей. Рассмотрим подробно каждый из них.

### 3.2.1 Описание модуля «web»

Модуль web отвечает за взаимодействие с пользователем. Он состоит из нескольких пакетов: playground, problem, statistics, user.

Начнем с описания пакета problem. В нем содержатся классы по работе с задачами и их решениями. На рисунке 3.1 представлена диаграмма классов пакета problem.



Рисунок 3.1 – Диаграмма классов пакета problem

Класс ProblemController обрабатывает запросы на получение информации о задачах, принимает решения задач от пользователей. Он отправляет сообщение с решением в очередь Kafka через класс TestRunnerSenderService и подписывается к классу TestRunnerChannelService на результаты тестирования решения. TestRunnerChannelService слушает сообщения из очереди Kafka. При получении сообщения с результатами оно обрабатывается классом TestResultAnalyzerService, который определяет по выходному файлу тестирования правильно ли решена задача. Для чтения и записи данных в базу данных используются классы SolutionResultRepository и SolutionRepository.

Пакет user отвечает за авторизацию и аутентификацию пользователя, безопасность доступа к другим модулям, создание и удаление пользователей, получение информации о пользователе. Подробное изложение пакета user не представляет актуального интереса и будет опущено, потому что реализация является тривиальной. Соответствующие классы пакета user доступны в исходных файлах проекта.

Пакет statistics отвечает за предоставление статистики онлайн-платформы по решению задач пользователями. На рисунке 3.2 представлена диаграмма классов пакета statistics.



Рисунок 3.2 – Диаграмма классов пакета statistics

Опишем назначение методов класса StatisticsService:

* topSolved возвращает список наиболее часто решаемых задач;
* topAttempted возвращает список задач, на которые пользователи пытались найти решение;
* topAttemptedNotSolved возвращает список задач, на которые пользователи пытались найти решение, но не смогли его найти;
* difficultyCounts возвращает количество задач по каждому уровню сложности;
* languageCounts возвращает количество задач, решенных на каждом языке программирования;
* difficultyCounts возвращает количество задач по каждому уровню сложности для конкретного пользователя;
* languageCounts возвращает количество задач, решенных на каждом языке программирования для конкретного пользователя;
* solvedAndAttempted возвращает количество решенных и попыток решения задач для конкретного пользователя.

StatisticsService взаимодействует с StatisticsRepository для получения статистических данных о решенных задачах пользователей. За каждым из этих методов стоит сложный SQL запрос к базе данных PostgreSQL.

Сервисный слой (StatisticsService) абстрагирует сложность доступа к данным и предоставляет удобный интерфейс для работы с ними. Кроме того, сервисный слой объединяет несколько источников данных, в том числе и базу данных, для предоставления полезной информации.

Контроллер StatisticsController, который предоставляет информацию о userId, взаимодействует с сервисным слоем для получения статистических данных о задачах, решенных пользователем с указанным идентификатором пользователя. Это позволяет предоставлять персонализированную статистику для каждого пользователя.

### 3.2.2 Описание модуля «test-runner»

Модуль test-runner отвечает за выполнение тестирования пользовательских решений алгоритмических задач. Он получает код решения от модуля web, создает изолированный Docker контейнер, в котором запускает тестирование пользовательского кода по набору тестов. На рисунке 3.3 представлена диаграмма классов.



Рисунок 3.3 – Диаграмма классов модуля test-runner

В данной иерархии классов есть три различных тестера: CppTester, JavaTester и JavascriptTester. Каждый из этих тестеров отвечает за тестирование кода, написанного на соответствующем языке программирования. У каждого тестера есть методы для создания контейнера, выполнения тестов, а также для обработки результатов тестирования.

Tester является базовым классом для всех тестеров и содержит общие методы и свойства, которые могут быть использованы всеми тестерами. Он также определяет абстрактные методы, которые должны быть реализованы в каждом конкретном тестере.

Listener – это класс, отвечающий за прослушивание сообщений от брокера Kafka и передачу их соответствующему тестеру для выполнения. Каждый тестер имеет связь с Listener, чтобы получать сообщения о тестировании, которые нужно выполнить.

Таким образом, иерархия классов позволяет абстрагировать логику тестирования кода на различных языках программирования и обеспечивает гибкость и расширяемость системы.

### 3.2.3 Описание модуля «task-runner»

Модуль task-runner отвечает за выполнение кода пользовательских программ. Он получает программу от модуля web, создает изолированный Docker контейнер, в котором запускает пользовательский код. На рисунке 3.4 представлена диаграмма классов.



Рисунок 3.4 – Диаграмма классов модуля task-runner

Эта иерархия классов представляет собой архитектуру системы для запуска кода на различных языках программирования и обработки событий, связанных с выполнением этого кода.

В системе есть три наследника типа Runner - CppRunner, JavaRunner и JavascriptRunner, каждый из которых предназначен для запуска кода на соответствующем языке программирования. У каждого наследника Runner есть методы для создания контейнера, в котором будет выполняться код, и получения имени файла с кодом.

Также в системе есть классы ErrorEvent, InfoEvent и LogEvent, которые представляют различные события, связанные с выполнением кода. Эти события используются для передачи информации о процессе выполнения кода и любых возникающих ошибках или исключениях.

RunnerController является контроллером, который принимает запросы на запуск кода от клиентов и передает их в RunnerService для выполнения. RunnerService в свою очередь использует соответствующий Runner в зависимости от языка программирования и возвращает поток с событиями, связанными с выполнением кода.

## 3.3 Описание структуры хранения контента онлайн-платформы

Материалы, с которыми пользователи могут взаимодействовать, называются контентом. Контент является статичной частью онлайн-платформы, поскольку его содержание и форма редко изменяются. В настоящее время контент представлен задачами, но в будущем могут появиться и другие сущности.

Хранение контента в виде файлов обеспечивает гибкость и простоту управления. Файловая структура облегчает добавление, изменение и удаление контента с помощью инструмента управления версиями Git. Файловый подход позволяет сохранить контент в читаемом формате, облегчая совместную работу над ним и упрощая резервное копирование и восстановление данных.

Применение Git для контроля версий контента приносит надежность в процесс сопровождения контента онлайн-платформы. Благодаря своей специализации на контроле версий, Git может превосходить современные CMS [14] по уровню удобства и гибкости работы с контентом, особенно для опытных пользователей. Git позволяет создавать ветки для разработки новых функций или исправлений, а также переходить между версиями контента в случае необходимости. Это обеспечивает спокойную работу с контентом без страха потери данных, поскольку каждое изменение фиксируется и хранится в истории изменений. Кроме того, сервисы хостинга Git-репозиториев позволяют управлять доступом к отдельным задачам или к репозиторию в целом, регулируя права доступа для различных пользователей или групп. С помощью Git подмодулей осуществимо добавление пользовательских задач.

Контентный репозиторий представляет собой древовидную структуру.

На рисунке 3.5 представлена схема иерархической структуры контентного репозитория, состоящей из файлов и папок. Папки обозначены прямоугольниками со скругленными углами. Структура начинается с корневого каталога контентного репозитория с подкаталогами: content, schemas и template.



Рисунок 3.5 – Иерархическая структура контентного репозитория

Рассмотрим назначение каждой папки корневого каталога:

* папка template содержит шаблон для создания задачи;
* папка content содержит файл manifest.yaml, описывающий список содержимого в контентном репозитории и подкаталоги под каждую категорию контента;
* папка schemas содержит схемы валидации данных, содержимых в репозитории.

Далее подробно рассмотрим содержимое каждого из подкаталогов.

### 3.3.1 Описание структуры хранения задачи

Подкаталог content содержит информацию о задачах. Внутри него находится папка problems, в которой каждая задача представлена отдельной папкой. Пример хранения задачи представлен на рисунке 3.6. Задачи хранятся в виде папки с уникальным названием. Уникальность обеспечивается хранением папок в общей директории.



Рисунок 3.6 – Пример хранимой задачи

Каждая задача включает в себя файл manifest.yaml, описывающий ее параметры, и отдельные подкаталоги для каждого реализованного языка программирования: /Cpp, /Java, /Javascript.

Подкаталог template содержит шаблон для создания задачи. Данный шаблон можно скопировать в папку problems и отредактировать под конкретную задачу. Шаблон ускоряет создание задач, поскольку структура у них одинаковая.

Хранение контента в виде файлов и папок обеспечивает структурированное хранение задач, что упрощает управление ими и обеспечивает четкое разделение по языкам программирования. Использование Git позволяет отслеживать изменения в задачах и контролировать версии. Применение YAML [15] формата для хранения данных обеспечивает читаемость и легкость в редактировании задач. Разделение задач по папкам по языкам программирования позволяет легко находить и обновлять решения для каждого языка.

### 3.3.2 Описание схемы валидации данных

Подкаталог schemas содержит схемы валидации данных. Для валидации файлов YAML формата используется JSON Schema [16].

Применение валидации через JSON Schema файлы YAML обеспечивает надежность и консистентность данных, сохраненных в YAML формате. JSON Schema позволяет определить требования к структуре и содержанию данных, что облегчает обнаружение и предотвращение ошибок еще на этапе ввода информации. Плюсы такого подхода включают уменьшение вероятности ошибок и обеспечение согласованности формата данных.

Кроме того, использование JSON Schema в файлах YAML поддерживается современными средами разработки, что существенно улучшает удобство разработки новых задач. Например, поддержка IntelliJ IDEA обеспечивает автозаполнение (autocomplete) и другие инструменты для удобной работы с валидируемыми файлами, что упрощает процесс разработки и сокращает время на поиск и исправление ошибок в данных.

Схема валидации задачи является json файлом, в котором определены поля задачи. На рисунке 3.7 представлен отрывок JSON Schema задачи. Отрывок из JSON Schema определяет структуру поля "languages". Это поле представляет собой массив строк, где каждая строка должна быть уникальной (параметр "uniqueItems": true). Элементы массива могут быть только строкового типа (параметр "type": "string") и должны соответствовать одному из значений: "Cpp", "Java" или "Javascript" (параметр "anyOf", содержащий константы "const" с указанными значениями). Таким образом, данный отрывок JSON Schema обеспечивает валидацию массива языков программирования, допуская только указанные варианты и обеспечивая уникальность элементов в массиве.



Рисунок 3.7 – отрывок JSON Schema задачи

## 3.4 Описание клиентской части программного обеспечения

Клиентская часть представляет собой интерактивный веб-интерфейс на основе библиотеки React. Библиотека React обеспечивает высокую производительность разработки и модульность, что позволяет легко разрабатывать и поддерживать интерфейс. Легкость поддержки интерфейса обеспечивается декларативным описанием композиции UI компонентов, которые можно переиспользовать.

### 3.4.1 Описание страниц клиентской части

Для лучшего понимания структуры клиентской части на рисунке 3.8 представлена карта сайта [17]. Карта сайта представляет собой схему, отображающую структуру веб-приложения, включая основные страницы и их взаимосвязи. На карте сайта розовым цветом отмечены общедоступные страницы, а голубым доступные только после авторизации.

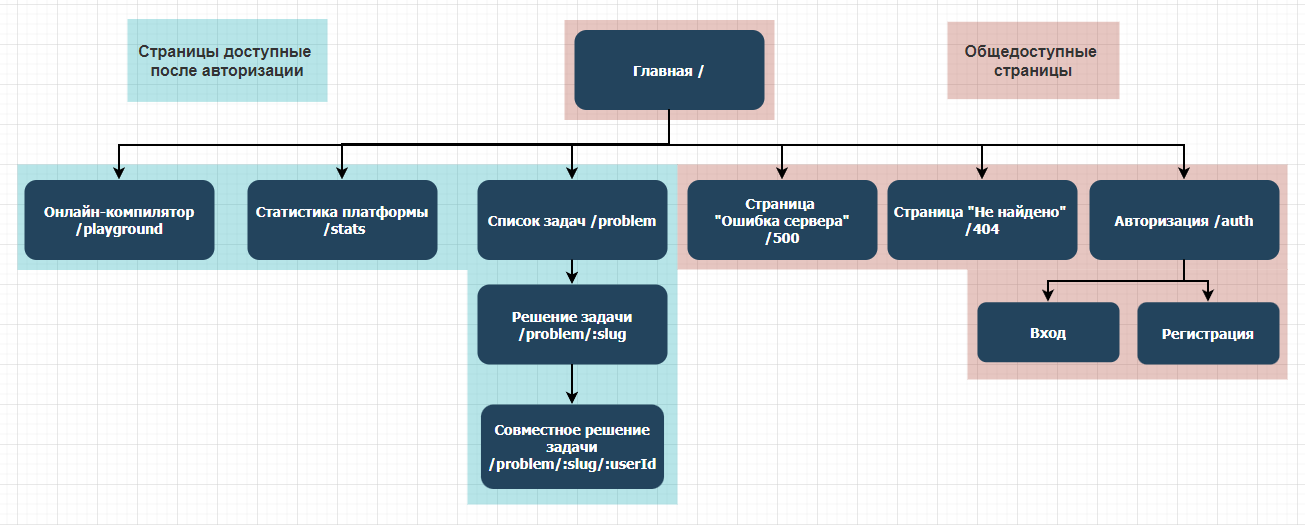


Рисунок 3.8 – Карта сайта SupremeCode

### 3.4.1 Описание главной страницы

Главная страница является входной точкой приложения и содержит обзорные данные о платформе. Здесь используются компоненты из библиотеки Mantine [19] для создания основных элементов интерфейса, таких как кнопки, карточки и панели навигации. В местах, где требуется кастомизация UI, применяется библиотека Tailwind CSS [20], что позволяет быстро и эффективно применять стили, поддерживая единый дизайн.

### 3.4.2 Описание страницы авторизации

Страница авторизации позволяет пользователям входить в систему и регистрироваться, обеспечивая доступ к функционалу платформы.

Для создания форм входа и регистрации используются готовые компоненты из библиотеки Mantine. Библиотека Mantine предоставляет удобный функционал для управления состоянием формы и обработки ошибок, что упрощает процесс валидации и отображения сообщений об ошибках пользователю.

Аутентификация пользователей осуществляется с помощью JSON Web Token (JWT) [21]. В целях безопасности токен не хранится в localStorage [22], а сохраняется в http-only cookies [23], что предотвращает доступ к токену из клиентского JavaScript и защищает его от атак XSS [24].

Для отправки запросов на сервер используется библиотека Axios [25]. Библиотека Axios обрабатывает запросы для аутентификации и регистрации пользователей, обеспечивая простоту и гибкость в настройке HTTP-запросов.

Глобальное состояние с информацией о пользователе поддерживается с помощью библиотеки Zustand [26]. Хранилище данных из библиотеки Zustand сохраняет данные о пользователе в localStorage, что позволяет сохранять состояние сеанса между перезагрузками страницы. Для обеспечения синхронизации состояния между разными вкладками браузера используется слушатель событий storage [27]. Этот слушатель обновляет состояние хранилища, если пользователь авторизовался на другой вкладке платформы, что обеспечивает согласованное состояние сеанса во всех открытых вкладках.

### 3.4.3 Описание страницы со списком задач

Страница задач предоставляет пользователям возможность просматривать доступные задачи. Загрузка данных для страницы осуществляется с помощью механизма loader из react-router [28], который запускает процесс загрузки данных параллельно с подгрузкой самой страницы. Это позволяет оптимизировать время загрузки и улучшить пользовательский опыт.

Для выполнения HTTP-запросов используется библиотека Axios, которая отправляет запросы на сервер для получения списка задач. Полученные данные кешируются с помощью библиотеки TanStack Query [29]. Результаты запросов сохраняются в кеше на 2 минуты, что позволяет минимизировать количество запросов к серверу и ускорить повторную загрузку данных.

При загрузке страницы предусмотрены различные сценарии отображения данных в зависимости от времени загрузки. Если данные загружаются не более полсекунды, отображается пустой список задач, что обеспечивает быстрое отображение интерфейса. Если загрузка данных занимает более полсекунды, пользователю отображается скелетон списка задач. Этот скелетон отображается не менее полсекунды, чтобы избежать мельканий и создать визуально приятный эффект загрузки.

### 3.4.4 Описание страницы решения задачи

Для отображения описания задачи в формате Markdown [30] и написания кода применяется библиотека Monaco Editor [31]. Библиотека Monaco Editor является частью популярного редактора кода Visual Studio Code [32], что обеспечивает пользователю знакомый и удобный интерфейс для работы с кодом.

Результаты решения задачи отображаются сразу после прохождения тестирования. Для этого используется механизм «длинные опросы» (long polling) [33], который позволяет получать обновления с сервера в реальном времени без необходимости постоянного опроса сервера. Это обеспечивает мгновенную обратную связь пользователю о результате выполнения кода.

Для реализации функции совместного редактирования кода используется библиотека Yjs [34], позволяющая создавать и управлять совместно используемыми структурами данных, такими как CRDT (Conflict-free Replicated Data Type) [35]. Протокол обмена данными реализуется с помощью библиотеки y-websocket [36], поддерживающей синхронизацию данных в реальном времени между несколькими пользователями. Для работы с y-websocket необходим сигнальный сервер, который также предоставляется библиотекой.

### 3.4.5 Описание страницы онлайн-компилятора

Страница с компилятором предоставляет возможность пользователям выполнять свой код в режиме реального времени. Для написания и редактирования кода используется библиотека Monaco Editor, как и на странице решения задачи.

Результаты компиляции и выполнения кода отображаются непосредственно после их получения с сервера. Для реализации этой функциональности применяется технология Server Sent Events (SSE) [37] с использованием библиотеки sse.js. Технология SSE позволяет серверу отправлять обновления клиенту в реальном времени через одностороннее соединение.

Страница с компилятором обеспечивает пользователям удобный и эффективный способ проверки кода и экспериментирования над ним, что содействует обучению и развитию навыков программирования.

### 3.4.6 Описание страницы статистики платформы

Страница статистики отображает данные по результатам решения задач. Для отображения графиков и диаграмм используется библиотека React Charts [38], входящая в состав Mantine. Это позволяет визуализировать данные и представлять результаты в удобной и понятной форме.

На странице статистики используются три вида графиков: столбчатая диаграмма BarChart, круговая диаграмма PieChart, радарная диаграмма RadarChart.

Круговая диаграмма PieChart используется для отображения статистики "Количество решенных и начатых задач", предоставляя долю решенных и начатых задач для быстрого визуального анализа.

Столбчатая диаграма BarChart применяется для отображения "Топ задач по количеству решивших", выделяя задачи, которые были успешно решены наибольшим количеством пользователей.

Радарная диаграмма RadarChart используется для отрисовки статистики "Количество решенных задач по языкам" и "Количество решенных задач по сложности". Для первой статистики радарная диаграмма помогает сравнить количество решенных задач на разных языках программирования, а для второй - оценить количество решенных задач в зависимости от их сложности.

Использование различных видов графиков на странице статистики обеспечивает пользователям удобный и наглядный способ визуализации аналитических данных, что способствует лучшему пониманию и анализу информации.

## 3.5 Выводы по главе

В данной главе были обоснованы выборы программных средств для разрабатываемой системы, включая выбор языка программирования Java.

Преимущества использования данных инструментов были рассмотрены подробно, а также проанализированы преимущества и недостатки программных комплексов разработки, необходимых для выполнения поставленных в работе задач. Кроме того, были рассмотрены модули и классы разрабатываемого программного средства, и для каждого из них представлены диаграммы с краткими описаниями классов и связей между ними.

Клиентская часть онлайн-платформы с алгоритмическими задачами была разработана с использованием современных технологий и подходов. Компонентная архитектура, централизованное управление состоянием и использование готовых библиотек для построения интерфейса обеспечивают высокую производительность разработки и удобство использования приложения.

# 4 Демонстрация онлайн-платформы «SupremeCode»

В данном разделе продемонстрирована работа онлайн-платформы «SupremeCode».

## 4.1 Описание процесса демонстрации

После открытия сайта, пользователь видит стартовую страницу, содержащую мотивирующий контент для обучения. На рисунке 4.1 представлен интерфейс этой страницы веб-сервиса.



Рисунок 4.1 – Начальная страница

При нажатии на кнопку «Начать работу» неавторизованный пользователь будет перенаправлен на форму регистрации, она показана на рисунке 4.2.



Рисунок 4.2 – Страница авторизации

При успешной авторизации попадаем на страницу со списком задач. Интерфейс этой страницы представлен на рисунке 4.3.



Рисунок 4.3 – Страница списка задач

Кликнув на любую задачу, откроется страница решения задачи. На странице есть подробное описание задачи, примеры входных и выходных данных, ограничения по входным данным. Можно выбрать один из трех языков программирования.



Рисунок 4.4 – Страница решения задачи

Когда напишете код, решающий задачу и будете готовы к проверке решения, нажмите кнопку «Запустить». Пока решение еще не протестировано, отображается спиннер загрузки. Перезагружать страницу самостоятельно не требуется. Результаты автоматически отобразятся на страницу без полной перезагрузки. Дождавшись завершения тестирования, можно увидеть на рисунке 4.5, что решение было одобрено. В левой панели отображается таблица с результатами решений. Также отображается дополнительная информация со статусом завершения программы, количеством выполненных тестов, ошибок, сбоев и временем выполнения. Как итог, задача решена. Справа от названия задачи отображается зеленая галочка, обозначающая успешное решение задачи.



Рисунок 4.5 – Страница решения задачи с решенной задачей

Иногда, самостоятельно решить задачу бывает сложно. В такие моменты можно пригласить более опытного друга для совместного решения проблемы. Для этого нажмите на кнопку «Оффлайн», чтобы перейти в онлайн режим, скопируйте ссылку для совместного редактирования и отправьте ее другу любым удобным способом. Меню копирования ссылки представлено на рисунке 4.6.



Рисунок 4.6 – Меню перехода в совместный режим

Друг, вставит эту ссылку в адресную строку браузера и подключится к совместному редактированию кода задачи. На рисунке 4.7 представлено два окна, слева владелец, справа гость. Можно заметить, что у каждого из пользователей появился второй курсор оранжевого цвета – это курсор второго участника совместного редактирования. Стоит отметить, что гостевой пользователь ограничен по возможности запуска проверки программ.



Рисунок 4.7 – Страницы двух пользователей при совместном редактировании

Если вам нужна площадка для программирования вне решения задачи, например, для проверки как работают стандартные функции вашего языка, то можете перейти на страницу «компилятора», она представлена на рисунке 4.8. На ней все также есть кнопка для запуска программы. Можно заметить, что текстовые сообщения о результатах выполнения программы появляются в правой панели по мере выполнения программы, а не после завершения. Это сложно передать скриншотом в отчете, но в реальности оно так и работает.



Рисунок 4.8 – Страница онлайн-компилятора

Если перейти на вкладку «Статистика», то можно обнаружить пользовательскую статистику по работе на онлайн-платформе. На рисунке 4.9 представлены первые два графика из шести существующих на странице статистики.



Рисунок 4.9 – Пользовательская статистика по решению задач

Проверим добавление новых задач. Пусть тестовая задача будет называться «test-problem». В контентном репозитории «supreme-code-content» есть шаблон для создания задачи. Скопируйте его в удобное для вас место. Откройте manifest.yaml в редакторе код IntelliJ IDEA. Выделим в атрибуте описания шаблон Markdown фрагмента и откроем его в отдельном окне с предпросмотром. Правая нижняя панель предпросмотра реагирует на редактирование Markdown разметки своевременным отображением результата. Попробуем отредактировать значение атрибута difficulty. Среда разработки подсказывает возможные значения: Easy, Hard, Normal. Все перечисленное представлено на рисунке 4.10.



Рисунок 4.10 – Окно редактирования задачи

Далее скопируйте из шаблона задачи папку /Java. Напишите шаблон решения в файл Solution.java и тесты для решения в JunitTest.java. Если выполнить все необходимое шаги, то задача появится на странице со списком задач, имея язык Java, как возможный для решения.

## 4.2 Вывод по главе

В этой главе было произведена демонстрация функционала онлайн-платформы «SupremeCode».

Был продемонстрирован процесс работы с онлайн-платформы: от мотивирующего контента на главной странице до регистрации, а затем к списку задач. После выбора задачи мы получаем детальное описание с примерами данных и ограничениями. Затем мы можем выбрать язык программирования и запустить решение задачи. Результаты тестирования отображаются динамически. Мы также можем пригласить друга для совместного редактирования кода задачи. Для программирования без задач предусмотрена страница онлайн-компилятора. Добавление новых задач проще благодаря шаблонам в контентном репозитории.

В результате проведенной демонстрации можно сделать вывод, что основные функции реализованы в том виде, в котором, предполагали поставленные требования.

# 5 Экономическое обоснование разработки онлайн-платформы «SupremeCode»

## 5.1 Основные аспекты реализации работы

Цель данной работы – создать онлайн-платформу «SupremeCode» для упрощения процесса обучения программированию за счет использования образовательной онлайн-платформы. Платформа позволяет ознакомиться с задачами, запрограммировать решение, отправить на автоматическую проверку решения. Программный продукт ускоряет обучение за счет сокращения времени на проверку решений. Актуальность обусловлена высоким спросом на интернет-образование в сфере IT. Уникальность платформы заключается в возможности решения задач на разных языках программирования, возможности запускать программы вне решения задач, совместного программирования решения, а также возможности предлагать задачи.

## 5.2 Характеристика программного продукта с позиции маркетинга

Программный продукт, созданный в ходе выполнения данной работы, является онлайн-платформой доступной из любого современного веб-браузера.

Перечислим основные требования, предъявляемые к программному продукту в целом:

* возможность получить информацию о задачах;
* возможность предоставить решение для задачи;
* сохранение предоставленных решений в базу данных;
* предоставление платформой статистики по решению задач;
* возможность добавить новую задачу на платформу.

Перечислим основные требования, предъявляемые к интерфейсу пользователя:

* наличие интуитивно понятной навигации по платформе;
* окно для написания программного кода;
* отображение графиков по статистике на платформе.

Результаты рассмотрения характеристик ПП с позиции маркетинга представлены на рисунке 5.1.



Рисунок 5.1 – Маркетинговая ориентация программного продукта

## 5.3 Стратегический маркетинговый анализ целесообразности применения программного продукта

Для определения целесообразности применения разработанного программного средства с позиции стратегического маркетинга был проведен SWOT-анализ. SWOT-метод анализа в стратегическом планировании, заключающийся в разделении факторов и явлений на четыре категории: strengths (сильные стороны), weaknesses (слабые стороны), opportunities (возможности) и threats (угрозы). Для определения сильных и слабых сторон организации в аспекте внедрения разработанного ПО в таблице 5.1 были определены стратегические определяющие для SWOT-анализа.

Таблица 5.1 – SWOT-анализ возможности внедрения ПП

|  |  |
| --- | --- |
| Возможности | Угрозы |
| 1. Изолированность запускаемых программ; 2. Добавление системы комментирования задач. | 1. Необходимость в дорогом серверном оборудовании, поддерживающем виртуализацию; 2. Замедление платформы при резком увеличении количества пользователей. |
| Сильные стороны | Слабые стороны |
| 1. Автоматическая проверка решенных задач; 2. Высокая отказоустойчивость; 3. Удобный интерфейс; 4. Возможность предлагать свои задачи. | 1. Отсутствие опыта реализации глобальных проектов; 2. Отсутствие яркого имиджа. |

В результате проведения анализа была доказана целесообразность применения данного программного продукта.

## 5.4 Определение продолжительности работ по разработке программного продукта

Для определения продолжительности работ по разработке программного продукта был составлен перечь работ и была определена их продолжительность. В разработке программного продукта принимали участие преподаватели-консультанты, руководитель и программист.

Определение этапов разработки и их продолжительности представлены в таблице 5.2.

Таблица 5.2 – Определение этапов разработки и их продолжительности

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Этапы | Виды работ | Длительность работ, час | | | Исполнители, чел. |
| tmin | tmax | T |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 1 | Разработка технического задания (ТЗ) | Изучение предметной области | 12 | 24 | 17 | Программист |
| Разработка и утверждение ТЗ | 2 | 8 | 5 | Программист, руководитель |
| Выбор способов создания онлайн-платформы «SupremeCode» | 10 | 24 | 16 | Программист |
| 2 | Подготовитель-ные работы | Изучение особенностей функционирования аналогичных платформ и составление отчета | 6 | 10 | 8 | Программист |
| Разработка разделов, связанных с созданием онлайн-платформы «SupremeCode» | 32 | 40 | 36 | Программист |
| Проверка разделов, связанных с созданием онлайн-платформы «SupremeCode» | 2 | 5 | 4 | Руководитель |
| Изучение реализации интерфейса онлайн-платформы «SupremeCode» | 20 | 30 | 24 | Программист |
| Разработка системного дизайна онлайн-платформы «SupremeCode» | 22 | 34 | 27 | Программист |
| Проверка и утверждение системного дизайна онлайн-платформы «SupremeCode» | 8 | 20 | 13 | Руководитель |
| 3 | Разработка  программного  обеспечения | Программирование по требованиям,  предъявленным к разработке приложения | 150 | 250 | 190 | Программист |
| Анализ и исправление ошибок в программных модулях | 40 | 80 | 56 | Программист |

Окончание таблицы 5.2

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 4 | Разработка  экономического  раздела | Утверждение задания на разработку экономического раздела | 1 | 3 | 2 | Преподаватель-консультант по экономической части |
| Проведение организационно-экономических расчетов, составление отчета | 9 | 12 | 11 | Программист |
| Проверка и утверждение отчета экономического раздела | 1 | 5 | 3 | Преподаватель-консультант по экономической части |
| 5 | Разработка  экологического раздела | Утверждение задания на разработку раздела БЖД | 1 | 2 | 2 | Преподаватель-консультант по части БЖД |
| Написание раздела по БЖД | 5 | 8 | 7 | Программист |
| Проверка и утверждение раздела БЖД | 1 | 5 | 3 | Преподаватель-консультант по части БЖД |
| 6 | Утверждение ВКР | Проверка отчета ВКР | 3 | 8 | 5 | Руководитель |
| Устранение ошибок | 6 | 24 | 14 | Программист |
| Составление и утверждение отчета и документации | 3 | 6 | 5 | Программист, руководитель |
| Итого | | | | | 448 |  |

Для определения продолжительности работ по разработке программного продукта используется формула 5.1.

|  |  |
| --- | --- |
|  | (5.1) |

Где T – ожидаемая длительность работы в часах, tmin – минимальная длительность работы, tmax – максимальная длительность работы.

Общее время для разработки вышло равным 448 часам. Для определения суммарной деятельности этапов создания ПП был построен график Ганта, изображенный на рисунке 5.2.



Рисунок 5.2 – График Ганта

## 5.5 Определение затрат на разработку и внедрение программного продукта

Затраты на разработку определяются по следующим статьям расходов: материальные затраты, заработная плата сотрудников, дополнительная заработная плата сотрудников, страховые взносы в государственные внебюджетные фонды, стоимость специального оборудования и программного обеспечения и накладные расходы. Общая величина материальных затрат рассчитывается по таблице 5.3.Таблица 5.3 – Материальные затраты, связанные с созданием и внедрением программного продукта

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № | Наименование материалов | | Ед. измерения | Количество | Цена за ед. руб. | Сумма, руб. | |
| 1 | 2 | | 3 | 4 | 5 | 6 | |
| 1 | Бумага для печати | | пачка | 1 | 400 | 400 | |
| 2 | Картридж для принтера | | шт. | 1 | 800 | 800 | |
| 3 | Услуги связи и интернета | | мес. | 2 | 550 | 1100 | |
| 4 | | Затраты на электроэнергию | - | - | - | 3000 |
| 5 | | Канцелярия | - | - | - | 400 |
| ИТОГО: | | | | | | 5700 |

К затратам по статье «Основная заработная плата» относится основная заработная плата, расчет которой выполняется исходя из трудоемкости выполнения каждого этапа и величины месячного должностного оклада исполнителя. Заработная плата определяется делением размера оклада на количество рабочих дней в месяце (21 день) с учетом восьмичасового рабочего дня. Расчет часов работы руководителя и разработчика, в случае совместно отрабатываемых часов, распределяется одинаково в полной мере. Расчет заработной платы сотрудников, используемых в разработке и внедрении программного продукта произведен в таблице 5.4.

Таблица 5.4 – Затраты по заработной плате

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель | Оклад, руб. | Часовая плата, руб. | Трудоемкость, часы | Затраты по заработной плате, руб. |
| Программист | 100000 | 595,2 | 416 | 247603 |
| Руководитель | 120000 | 714,2 | 32 | 22848 |
| Преподаватель-консультант по экономической части | 50000 | 327,3 | 5 | 1636 |
| Преподаватель-консультант по части БЖД | 50000 | 327,3 | 5 | 1636 |
| ИТОГО: | | | | 273723 |

Дополнительная заработная плата (ЗПД) определяется в процентах от основной заработной платы программистов и руководителя. Норматив доплат принимается в размере 20% (а). Расчет представлен в формуле 5.2.

|  |  |
| --- | --- |
|  | (5.2) |
|  |  |

Страховые взносы в государственные внебюджетные фонды (СВ) с заработной платы (основной и дополнительной) определяются в соответствии с установленным законодательством РФ процентом (в 2024 году b=30%). Расчет представлен в формуле 5.3.

|  |  |  |  |
| --- | --- | --- | --- |
|  | (5.3) | | |
|  | |  |

Расчет стоимости использованного оборудования и программного обеспечения произведен в таблице 5.5.

Таблица 5.5 – Стоимость использованного оборудования и программного

обеспечения

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование компьютерного оборудования,  оргтехники, программного обеспечения | Количество, шт. | Цена за ед. руб. | Сумма, руб. |
| 1 | 2 | 3 | 4 | 5 |
| 1 | Ноутбук Lenovo | 1 | 40000 | 40000 |
| 2 | ПО Office 365 | 1 | 5450 | 5450 |
| ИТОГО: | | | | 45450 |

Расчет суммы амортизационных отчислений по методу ускоренной амортизации произведен в таблице 5.6.

Таблица 5.6 – Амортизационные отчисления основных средств производства

(основного оборудования и программного обеспечения) за год

|  |  |  |  |
| --- | --- | --- | --- |
| Наименование  ОСП | Общая первоначальная  стоимость, руб | Годовая норма  амортизации, % | Сумма годовой  амортизации, руб. |
| 1 | 2 | 3 | 4 |
| 1 | 40000 | 20 | 8000 |
| 2 | 5450 | 20 | 1090 |
| ИТОГО: | | | 9090 |

Поскольку основные средства производства (оборудование и программное обеспечение) для разработки использовались в течении определенного периода времени, необходимо учесть только часть суммы годовой амортизации. Расчет процента длительности разработки происходит путем деления длительности разработки в рабочих днях (tобщ) на число рабочих дней в году равное 255 (РД). Расчет процента длительности разработки произведен в формуле 5.4.

|  |  |
| --- | --- |
|  | (5.4) |
|  |  |

Результаты расчета амортизации основного оборудования и программного обеспечения в расчет на проект представлены в таблице 5.7.

Таблица 5.7 – Амортизационные отчисления использованного оборудования и программного обеспечения

|  |  |  |  |
| --- | --- | --- | --- |
| № | Наименование основных средств | Сумма годовой амортизации, руб. | Сумма амортизации  средств в расчёте на проект, руб. |
| 1 | 2 | 3 | 4 |
| 1 | Ноутбук Lenovo | 8090 | 1780 |
| 2 | ПО Office 365 | 1090 | 240 |
| ИТОГО: | | | 2020 |

К статье накладные расходы относят затраты, понесенные на организацию и обслуживание производства. Накладные расходы составили 40000 рублей на аренду помещения, 5300 рублей на оплату коммунальных услуг. Итого сумма накладных расходов составляет 45300 рублей.

Результаты расчёта себестоимости проекта приведены в таблице 5.8.

Таблица 5.8 – Сводная таблица себестоимости проекта

|  |  |  |
| --- | --- | --- |
| № | Наименование статей затрат | Сумма, руб |
| 1 | 2 | 3 |
| 1 | Материальные затраты | 5700 |
| 2 | Заработная плата сотрудников | 273723 |
| 3 | Дополнительная заработная плата | 54744 |
| 4 | Страховые взносы | 98540 |
| 5 | Амортизационные отчисления | 1940 |
| 6 | Накладные расходы | 45300 |
| Итого (Ед.з.): | | 479947 |

Таким образом, по итогу всех необходимых расчетов была выявлена себестоимость проекта, итоговое значение которой составило 479947 рублей.

Приложение распространяется на бесплатной основе.

## 5.7 Выводы по главе

В данной главе была описана цель разработки и функциональное назначение программного средства, а также обоснована актуальность разработки.

Кроме того, были выполнены организационно-экономические расчёты, включающие в себя: характеристику программного продукта с позиций маркетинга, стратегический маркетинговый анализ целесообразности применения программного продукта, определение продолжительности работ по разработке программного продукта (448 часов), определение затрат на разработку и внедрение программного продукта. По итогам расчётов плановая себестоимость создания онлайн-платформы с алгоритмическими задачами составила 479947 рублей.

# 6 Безопасность и экологичность выпускной квалификационной работы

## 6.1 Анализ вредных и опасных факторов, влияющих на здоровье программиста

Многие не задумываются о том, какие факторы негативно влияют на здоровье программиста, а также приводят к серьезным заболеваниям. Вот некоторые из них:

* повышенная нагрузка на зрение;
* освещенность рабочего места;
* монотонные однообразные мелкие движения руками;
* длительное пребывание в сидячем положении;
* пыль.

Согласно [санитарным правилам 2.2.3670-20 "Санитарно-эпидемиологические требования к условиям труда"](https://docs.cntd.ru/document/573230583#6560IO), которые регламентируют расстояние между экраном монитора и работником, уровень освещенности при работе с компьютером, а также правильную организацию рабочего стола по средствам правильно подобранной мебели.

Вредное влияние на глаза проявляется в необходимости постоянного напряжения глаз при работе за компьютером или ноутбуком, что усугубляется при несоблюдении регламентированных норм уровня освещенности рабочего места. Также постоянное напряжение рук может привести к нарушению кровообращения, вызываемому повторяющимися нагрузками, а статичная напряженная поза при продолжительной работе - к воспалению мышц спины, заболеваний суставов. Поэтому для профилактики негативных воздействий четко установлен режим работы и отдыха.

В свою очередь офисная пыль также является распространенной проблемой. Она может содержать до 80% вредных веществ, среди которых угарный газ, аллергены и возбудители заболеваний. Наибольшее количество пыли скапливается в вентиляционных системах. Система воздуховодов загрязняется уже через год после начала эксплуатации. В результате развиваются микроорганизмы, которые попадают по вентиляционным системам во все помещения здания.

Поэтому межгосударственный стандарт ГОСТ ИСО 14644-1-2017 «Чистые помещения и связанные с ними контролируемые среды. Часть 1. Классификация чистоты воздуха» устанавливает классы чистоты воздуха по счетной концентрации частиц в единице объема воздуха. Несоблюдение подобных норм может повлечь отрицательное воздействие на здоровье сотрудников и вызвать как легкие, так и тяжелые, и хронические заболевания дыхательных путей.

Учитывая все вышесказанное можно выделить основные мероприятия, позволяющие сократить отрицательное влияние опасных факторов на состояние здоровья сотрудников офисного помещения, требующие регулярного исполнения:

* проведение зарядки для глаз;
* проведение физической разминки во время отдыха;
* регулярное проветривание помещения;
* использование ионизатора воздуха в помещении согласно инструкции;
* следить за техническим состоянием вентиляции;
* распределение искусственного освещения согласно нормативным актам.

## 6.2 Расчет системы искусственного освещения помещений

На рисунке 6.1 представлена схема помещения с указанием его размеров и расположением окон и компьютерных столов.



Рисунок 6.1 – Схема помещения

Необходимо произвести расчет искусственного освещения, выполненного светильниками ЛСП01 с двумя люминесцентными лампами длинной 𝑙𝐶𝐵 = 1534 мм. Исходные данные представлены в таблице 1.

Таблица 6.1 – Исходные данные

|  |  |
| --- | --- |
| Наименование | Величина |
| Длина помещения, А, м | 6 |
| Ширина помещения, В, м | 3.3 |
| Высота помещения, h, м | 2,7 |
| Коэффициент отражения потолка | 50 |
| Коэффициент отражения стен | 30 |
| Коэффициент отражения пола | 10 |
| Нормируемая минимальная освещенность, , лк | 300 |
| Число ламп в светильнике, n, шт | 2 |
| Длина светильника, , м | 1,534 |

Окончание таблицы 6.1

|  |  |
| --- | --- |
| Наименование | Величина |
| Коэффициент запаса, К, | 1,5 |
| Коэффициент использования светового потока, η | 46 |
| Мощность лампы, P, Вт | 30 |
| Световой поток, , лм | 1840 |

Определим высоту подвеса светильника над рабочей поверхностью по следующей формуле:

, (1)

где м, высота рабочей поверхности,

ℎ𝑐 = 0,25 (ℎ − ℎ𝑝). (2)

Поскольку высота помещения , применяем потолочное крепление светильников , возьмем значение ℎ𝑐 = 0,2 м

Определим индекс помещения:

(3)

Используя дополнительные материалы, определим коэффициент запаса K, а также коэффициент использования светового потока 𝜂. В нашем случае эти переменные будут иметь следующие значения:

𝐾 = 1,5

𝜂 = 46

Определим расстояние между рядами светильников и от стен до ближайшего ряда по формуле:

L = λHП, (4)

где 𝜆 = 1,1 − коэффициент оптимального расстояния между геометрическими центрами светильников по светотехническим требованиям.

L = 1,1 1,7 ≈ 1,87 м.

Расстояние от стен до ближайшего ряда по рекомендациям вычитывается по следующей формуле:

0,3 L = 0,3 1,87 = 0,56 м.

Далее необходимо определить общее количество светильников N при условии равномерного освещения, а также количество светильников по длине и ширине:

(5)

. (6)

(7)

.

Определим световой поток одной лампы по следующей формуле:

(8)

где Z = 1,1 – коэффициент минимальной освещенности.

Световой поток лампы должен соответствовать соотношению:

(9)

где – расчетный световой поток, лм;

– световой поток, лм, табличное значение;

1775 ≈ 1840

Для определения соответствия проектируемой системы освещения требованиям СП 52.13330.2016 выполним проверочный расчет по следующей формуле:

(10)

Так как , то освещение удовлетворяет требованиям.

Определим потребляемую мощность осветительной установки:

(11)

Где – мощность лампы, Вт;

– коэффициент, учитывающий потери пускорегулирующей аппаратуры. Для ЛЛ 1,25.

На рисунке 6.2 представлен эскиз размещения светильников на плане помещения.



Рисунок 6.2 – Схема размещения светильников

## 6.3 Вывод по главе

В данной главе произведен расчет системы искусственного освещения рабочего помещения, а также определено, что проектируемая система освещения соответствует требованиям СП52.13330.2016.

Показаны и построены схемы размещения светильников в помещении с указанием расстояния от стен до ближайшего ряда по длине и оптимального расстояния между геометрическими центрами светильников по ширине между рядами светильников.

Также в данной главе мы описали возможные мероприятия для сокращения пагубного влияния на здоровье сотрудников офиса.

Заключение

В ходе выполнения выпускной квалификационной работы были решены следующие задачи.

Проведен обзор предметной области и выполнен сравнительный анализ онлайн-платформ с алгоритмическими задачами по программированию.

Выполнено проектирование структуры данных и микросервисной архитектуры платформы.

Обоснован выбор программного инструментария: серверная часть система реализована на Java, клиентская часть системы реализована посредством TypeScript, для хранения данных выбрана СУБД PostgreSQL.

Проведено программное конструирование информационной системы и разработана система, отвечающая всем требованиям технического задания.

Разработана документация на информационную систему, обоснована безопасность и экологичность информационной системы.

Перечень использованных информационных ресурсов

1. Docker Контейнеризация [Электронный ресурс], URL: https://www.docker.com/ (дата обращения 14.05.2024).
2. Leetcode платформа с задачами по программированию [Электронный ресурс], URL: https://leetcode.com/ (дата обращения 14.05.2024).
3. Информатикс Олимпиадное программирование [Электронный ресурс], URL: https://informatics.msk.ru/?redirect=0 (дата обращения 14.05.2024).
4. Codewars Обучение программированию [Электронный ресурс], URL: https://www.codewars.com/dashboard (дата обращения 14.05.2024).
5. Apache Kafka Система обмена сообщениями [Электронный ресурс], URL: https://kafka.apache.org/ (дата обращения 14.05.2024).
6. REST API Архитектурный стиль взаимодействия компонентов [Электронный ресурс], URL: https://ru.wikipedia.org/wiki/REST (дата обращения 14.05.2024).
7. Java Язык программирования [Электронный ресурс], URL: https://www.oracle.com/java/ (дата обращения 14.05.2024).
8. Spring Фреймворк для разработки Java-приложений [Электронный ресурс], URL: https://spring.io/ (дата обращения 14.05.2024).
9. TypeScript Статически типизированный язык программирования [Электронный ресурс], URL: https://www.typescriptlang.org/ (дата обращения 14.05.2024).
10. React JavaScript библиотека для создания пользовательских интерфейсов [Электронный ресурс], URL: https://react.dev/ (дата обращения 14.05.2024).
11. PostgreSQL Объектно-реляционная система управления базами данных [Электронный ресурс], URL: https://www.postgresql.org/ (дата обращения 14.05.2024).
12. IntelliJ IDEA Community Edition Интегрированная среда разработки [Электронный ресурс], URL: https://github.com/JetBrains/intellij-community (дата обращения 14.05.2024).
13. Git Система контроля версий [Электронный ресурс], URL: https://git-scm.com/ (дата обращения 14.05.2024).
14. CMS Система управления содержимым [Электронный ресурс], URL: https://ru.wikipedia.org/wiki/Система\_управления\_содержимым (дата обращения 14.05.2024).
15. YAML Язык разметки данных [Электронный ресурс], URL: https://yaml.org/ (дата обращения 14.05.2024).
16. JSON Schema Язык описания схем для JSON [Электронный ресурс], URL: https://json-schema.org/ (дата обращения 14.05.2024).
17. Карта сайта [Электронный ресурс], URL: https://ru.wikipedia.org/wiki/Карта\_сайта (дата обращения 14.05.2024).React Router [Электронный ресурс], URL: https://github.com/remix-run/react-router#readme (дата обращения 14.05.2024).
18. Библиотека UI компонентов Mantine [Электронный ресурс], URL: https://mantine.dev/ (дата обращения 14.05.2024).
19. Библиотека атомарных классов Tailwind CSS [Электронный ресурс], URL: https://tailwindcss.ru/ (дата обращения 14.05.2024).
20. Токен доступа JSON Web Token [Электронный ресурс], URL: https://ru.wikipedia.org/wiki/JSON\_Web\_Token (дата обращения 14.05.2024).
21. Веб хранилище Local Storage [Электронный ресурс], URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage (дата обращения 14.05.2024).
22. Фрагменты данных Cookies [Электронный ресурс], URL: https://developer.mozilla.org/ru/docs/Web/HTTP/Cookies (дата обращения 14.05.2024).
23. Межсайтовый скриптинг [Электронный ресурс], URL: https://ru.wikipedia.org/wiki/Межсайтовый\_скриптинг (дата обращения 14.05.2024).
24. Библиотека асинхронных запросов Axios [Электронный ресурс], URL: https://axios-http.com/ (дата обращения 14.05.2024).
25. Библиотека управления состоянием Zustand [Электронный ресурс], URL: https://docs.pmnd.rs/zustand/getting-started/introduction (дата обращения 14.05.2024).
26. Событие хранилища [Электронный ресурс], URL: https://developer.mozilla.org/en-US/docs/Web/API/Window/storage\_event (дата обращения 14.05.2024).
27. Загрузчик данных react-router: loader [Электронный ресурс], URL: https://reactrouter.com/en/main/route/loader (дата обращения 14.05.2024).
28. Библиотека управления асинхронным состоянием TanStack-Query [Электронный ресурс], URL: https://tanstack.com/query/v3 (дата обращения 14.05.2024).
29. Язык разметки страниц Markdown [Электронный ресурс], URL: https://ru.wikipedia.org/wiki/Markdown (дата обращения 14.05.2024).
30. Библиотека редактора кода Monaco Editor [Электронный ресурс], URL: https://microsoft.github.io/monaco-editor/ (дата обращения 14.05.2024).
31. Текстовый редактор Visual Studio Code [Электронный ресурс], URL: https://code.visualstudio.com/ (дата обращения 14.05.2024).
32. Веб механизм взаимодействия Long Polling [Электронный ресурс], URL: https://learn.javascript.ru/long-polling (дата обращения 14.05.2024).
33. Библиотека совместного состояния Yjs [Электронный ресурс], URL: https://yjs.dev/ (дата обращения 14.05.2024).
34. Структуры данных CRDT [Электронный ресурс], URL: https://neerc.ifmo.ru/wiki/index.php?title=CRDT (дата обращения 14.05.2024).
35. Библиотека с реализацией протокола Y-Websocket [Электронный ресурс], URL: https://docs.yjs.dev/ecosystem/connection-provider/y-websocket (дата обращения 14.05.2024).
36. Веб механизм взаимодействия Server-Sent Events [Электронный ресурс], URL: https://learn.javascript.ru/server-sent-events (дата обращения 14.05.2024).
37. Библиотека отрисовки графиков React Charts [Электронный ресурс], URL: https://react-charts.tanstack.com/ (дата обращения 14.05.2024).

Приложение А Техническое задание

|  |  |
| --- | --- |
| СОГЛАСОВАНО  Зав. кафедрой «ПОВТиАС»  \_\_\_\_\_\_\_\_\_\_ В.В. Долгов  «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. | УТВЕРЖДЕНО  Зав. кафедрой «ПОВТиАС»  \_\_\_\_\_\_\_\_\_\_ В.В. Долгов  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_2024 г. |

## А.1. Общие сведения

### А.1.1 Имя сайта

«SupremeCode»

### А.1.2 Полное наименование системы

Онлайн-платформа с алгоритмическими задачами по программированию «SupremeCode»

### А.1.3 Перечень документов, на основании которых создается система

Основанием для разработки является задание к выпускной квалификационной работе (ВКР), согласованное с руководителем ВКР заведующим кафедры «ПОВТиАС» Долговым Василием Валерьевичем с одной стороны, студентом гр. ВПР43 Голосуевым Данилом Витальевичем, именуемым в дальнейшем исполнителем с другой стороны, утвержденному заведующим кафедры «ПОВТиАС» Долговым Василием Валерьевичем.

### А.1.4 Порядок оформления и предъявления заказчику результатов работ по созданию системы

Система передается в виде функционирующего комплекса на базе средств вычислительной техники Заказчика и Исполнителя. Приемка системы осуществляется комиссией в составе уполномоченных представителей Заказчика и Исполнителя.

### А.1.5 Перечень нормативно–технических документов, методических материалов, использованных при разработке ТЗ

При разработке автоматизированной системы и создании проектно–эксплуатационной документации Исполнитель должен руководствоваться требованиями следующих нормативных документов:

ГОСТ 19.201–78. Техническое задание. Требования к содержанию и оформлению.

ГОСТ 34.601–90. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Стадии создания;

ГОСТ 34.201–89. Информационная технология. Комплекс стандартов на автоматизированные системы. Виды, комплексность и обозначение документов при создании автоматизированных систем;

РД 50–34.698–90. Методические указания. Информационная технология. Комплекс стандартов на автоматизированные системы. Автоматизированные системы. Требования к содержанию документов.

## А.2 Назначение информационной системы

Основным назначением ИС автоматизация процесса проверки решений алгоритмических задач, предоставленных пользователями онлайн-платформы.

## А.3.Требования к информационной системе

### А.3.1 Требования к системе в целом

#### А.3.1.1 Требования к графическому дизайну ИС

Дизайн ИС должен соответствовать дизайну, утвержденному Заказчиком

#### А.3.1.2 Требования к шрифтовому оформлению ИС

Основной шрифт определяется браузером пользователя.

Размер (кегль) шрифтов должен обеспечивать удобство восприятия текста при минимально допустимом размере экрана.

#### А.3.1.3 Требования к средствам просмотра Сайта

Сайт должен обеспечивать корректное отображение данных в следующих браузерах:

Яндекс Браузер (версия 24.4.2.887 и выше);

Microsoft Edge (версия 124.0.2478.97 и выше);

Google Chrome (версия 124.0.6367.201 и выше).

#### А.3.1.4 Требования к контенту и наполнению ИС

Первичная разработка и верстка контента (информационного содержимого) Сайта должна производиться силами Исполнителя при согласовании с Заказчиком. Заказчик предоставляет все необходимые Исполнителю текстовые и графические материалы, а также комментарии, касающиеся их содержания, объема, оформления и размещения.

#### А.3.1.5 Требования к компоновке страниц ИС

Компоновка страниц ИС должна обеспечивать минимальный размер страниц в зависимости от ширины рабочего поля браузера пользователя. Минимальный размер (ширина) рабочего поля браузера, при котором необходимо обеспечить полноценное отображение страниц (без полосы горизонтальной прокрутки), составляет 1024 пиксела.

#### А.3.1.6 Язык сайта

Русский.

### А.3.2 Требования к функциям (возможностям), выполняемых сайтом

#### А.3.2.1 Основные требования

Система должна представлять собой веб-сайт, размещенный в сети Интернет.

## A.4 Требования к программе или программному изделию

### А.4.1 Требования к функциональным характеристикам

Программное средство должно осуществлять следующие функции:

* предоставлять возможность регистрации и аутентификации;
* отображать список задач, доступных для решения;
* предоставлять форму для отправки решений задач;
* предоставлять пользователю возможность проверки решения;
* сохранять данные о пользователях, задачах и их решениях в базе данных;
* предоставлять возможность добавления, изменения, удаления задач;
* отображать пользователю результаты проверки решения задач.

### А.4.2 Требования к надежности

Надежное функционирование программы должно быть обеспечено выполнением совокупности нижеописанных мероприятий:

* выполнение требований ГОСТ 51188-98;
* защита информации;
* организацией бесперебойного питания технических средств;
* использованием лицензионного программного обеспечения.

### А.4.3 Условия эксплуатации

Для функционирования программного продукта необходимо соблюдение всех требований и правил эксплуатации мобильной техники.

Высокая квалификация пользователя программного средства не требуется. Дополнительных требований и ограничений не вводится.

Требования к персоналу, работающему с данным программным продуктом – общие знания вычислительной техники.

### А.4.4 Требования к составу и параметрам технических средств

Состав технических средств на стороне клиента:

* процессор с тактовой частотой от 3 ГГц;
* необходимо наличие адаптера подключения к сети;
* графический дисплей шириной от 1280 CSS-пикселей;
* оперативная память объемом не менее 4 Гб.

Состав технических средств на стороне сервера:

* процессор архитектуры x64 с тактовой частотой от 3 ГГц;
* минимальный объём требуемого дискового пространства составляет 1 Гб;
* необходимо наличие адаптера подключения к сети;
* оперативная память объемом не менее 4 Гб.

### А.4.5 Требования к информационной и программной совместимости

Для функционирования программного средства на стороне клиента необходимо следующее программное обеспечение: интернет-браузер.

Для функционирования программного средства на стороне сервера необходима jdk temurin версии 21 и выше, установленная PostgreSQL версии 16 и выше, операционная система Ubuntu 22.04 и выше, установленный Docker 25.0.3 и выше.

#### А.4.5.1 Язык программирования

Для разработки программного средства должны быть использованы языки программирования Java и TypeScript.

#### А.4.5.2 Операционная система

Программное средство должно работать под управлением ОС Linux.

### А.4.6 Требования к упаковке и маркировке

Требования к упаковке и маркировке программного средства не предъявляется.

### А.4.7 Требования к транспортировке и хранению

Условия транспортирования, места хранения, условия складирования и сроки хранения в различных условиях должны соответствовать требованиям, предъявляемым к носителям информации, на которых будет содержаться данное программное изделие.

Допустимы все способы транспортирования и хранения, не нарушающие целостность используемого носителя данных. Программное средство может храниться на любом носителе информации, имеющее возможность подключения к персональному компьютеру.

### А.4.8 Специальные требования

Для корректной работы программного средства необходимо разрешение на работу с памятью устройства, чтобы обеспечить корректное открытие и сохранение файлов.

## А.5 Требования к программной документации

Программная документация должна состоять из следующих листов:

* титульный лист;
* пояснительная записка к производственной практике;
* техническое задание по ГОСТ 19.201-78 ЕСПД;
* исходный код программного средства по ГОСТ 19.401-79 ЕСПД.

## А.6 Стадии и этапы разработки

* постановка задачи (с 20.04.20 по 22.04.20);
* изучение предметной области (с 23.04.20 по 26.04.20);
* разработка алгоритмов решения задачи (с 27.04.20 по 02.05.20);
* разработка программы (с 04.05.20 по 09.05.20);
* тестирование программы (с 11.05.20 по 14.05.20).

## А.7 Порядок контроля и приемки

Порядок и контроль приёмки определяются заведующим кафедрой «ПОВТиАС» и основаны на демонстрации знаний технологии и умении создавать программные средства для различных предметных областей.

Главным требованием к приемке является наличие правильно работающего программного средства с тестовым примером и отчета, представленного в печатном виде.

Разработчик технического задания /Голосуев Д.В./

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_2024 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись

Приложение Б Диаграмма последовательности обработки запроса на проверку решения



Приложение В Листинг программы

Листинг Б.1 – Основная активность приложения

package net.danil.web;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.kafka.annotation.EnableKafka;

@SpringBootApplication(scanBasePackages = {"org.danil", "net.danil"})

@EnableKafka

public class WebApplication {

public static void main(String[] args) {

SpringApplication.run(WebApplication.class, args);

}

}

package net.danil.web.problem.service;

import lombok.RequiredArgsConstructor;

import net.danil.web.problem.dto.TestResult;

import net.danil.web.problem.model.SolutionResult;

import net.danil.web.problem.repository.SolutionRepository;

import net.danil.web.problem.repository.SolutionResultRepository;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.kafka.annotation.KafkaListener;

import org.springframework.kafka.support.KafkaHeaders;

import org.springframework.messaging.Message;

import org.springframework.messaging.handler.annotation.Header;

import org.springframework.messaging.handler.annotation.Payload;

import org.springframework.messaging.support.GenericMessage;

import org.springframework.stereotype.Service;

import reactor.core.publisher.Mono;

import reactor.core.publisher.MonoSink;

import java.util.concurrent.ConcurrentHashMap;

@Service

@RequiredArgsConstructor

public class TestRunnerChannelService {

public static final String TOPIC\_NAME = "test-result-topic";

private final ConcurrentHashMap<String, MonoSink<Message<?>>> messageHandlers = new ConcurrentHashMap<>();

private final Logger logger = LoggerFactory.getLogger(TestRunnerChannelService.class);

private final SolutionResultRepository solutionResultRepository;

private final SolutionRepository solutionRepository;

private final TestResultAnalyzerService testResultAnalyzerService;

@KafkaListener(topics = TOPIC\_NAME)

protected void listen(@Payload TestResult testResult, @Header(value = KafkaHeaders.RECEIVED\_KEY, required = false) String messageId) {

if (messageId == null) {

logger.error("received null message id");

return;

}

logger.info("received result forId({}): {}", messageId, testResult);

final var sink = messageHandlers.get(messageId);

try {

final var verdict = testResultAnalyzerService.judgeResults(testResult);

final var solution = solutionRepository.findById(testResult.solutionId()).get();

SolutionResult solutionResult = new SolutionResult(solution.getId(), testResult.tests(), testResult.failures(), testResult.errors(), testResult.statusCode(), testResult.time(),

testResult.logs(), testResult.xml(), verdict.solved(), solution);

solutionResultRepository.save(solutionResult);

solution.setSolutionResult(solutionResult);

sink.success(new GenericMessage<>(solution));

} catch (Exception e) {

sink.error(new RuntimeException(testResult.toString(), e));

}

}

public Mono<Message<?>> subscribe(String id) {

return Mono.create(sink -> messageHandlers.put(id, sink));

}

}

package net.danil.web.problem.service;

import lombok.RequiredArgsConstructor;

import net.danil.web.problem.dto.TestMessage;

import net.danil.web.problem.model.Solution;

import net.danil.web.problem.repository.SolutionRepository;

import net.danil.web.user.repository.UserRepository;

import org.danil.model.Language;

import org.springframework.kafka.core.reactive.ReactiveKafkaProducerTemplate;

import org.springframework.stereotype.Service;

import reactor.core.publisher.Mono;

@Service

@RequiredArgsConstructor

public class TestRunnerSenderService {

public static final String TOPIC\_NAME = "test-topic";

final private ReactiveKafkaProducerTemplate<String, TestMessage> kafka;

final private SolutionRepository solutionRepository;

final private UserRepository userRepository;

public Mono<Long> send(Long userId, String code, String slug, Language language) {

final var user = userRepository.getReferenceById(userId);

final var solution = solutionRepository.save(new Solution(null, user, code, slug, language, null));

final var solutionId = solution.getId();

return kafka.send(TOPIC\_NAME, solutionId.toString(), new TestMessage(

solutionId, solution.getCode(), solution.getProblemSlug(), solution.getLanguage()

)).map(m -> solutionId);

}

}

package net.danil.web.problem.service;

import net.danil.web.problem.dto.TestResult;

import org.springframework.stereotype.Service;

@Service

public class TestResultAnalyzerService {

public record Verdict(boolean solved) {

}

public Verdict judgeResults(TestResult testResult) {

if(testResult.statusCode() != 0)

return new Verdict(false);

if(testResult.failures() != 0 || testResult.errors() != 0)

return new Verdict(false);

if(testResult.tests() == 0)

return new Verdict(false);

return new Verdict(true);

}

}