**Промежуточный отчет по программному проекту**

**1. Основные планы и этапы проекта**

1.1 Краткое описание проекта:

Название проекта: «Сервис Item composition»

Цель проекта: Реализовать сервис для централизованного форматирования и представления данных по заданным шаблонам и схемам с возможностью интеграции с внутренними сервисами и компонентами.

Краткое описание задач:

1. Реализовать runtime кодогенерацию по описанию шаблонов представления данных.
2. Реализовать runtime кодогенерацию клиентов для интеграции с внутренними сервисами и компонентами.
3. Оптимизировать код с помощью батч-запросов.
4. Настроить качественное логирование и трейсинг.
5. Выполнить интеграционное и unit тестирование кода.

1.2 Планы и этапы выполнения проекта:

| Этап проекта | Описание работ | Ожидаемые результаты | Сроки выполнения |
| --- | --- | --- | --- |
| Подготовительные работы | * Постановка задачи. * Обоснование возможности решения поставленной задачи. * Предварительный выбор методов решения задач. | * Проведена декомпозиция проекта на подзадачи. * Выбран технологический стек и методы решения задач. | 21.11.2024 |
| Разработка и утверждение технического задания | * Определение требований к программе. * Определение стадий. Этапов и сроков разработки программы и документации на нее. * Согласование и утверждение технического задания. | * Утвержденное техническое задание. | 04.12.2024 |
| Разработка программы | * Разработка и тестирование программы | * Полностью работоспособная программа. | 01.04.2025 |
| Разработка программной документации | * Разработка программных документов в соответствии с требованиями ГОСТ 19.101-77. | * Написана “Пояснительная записка”. * Написано “Руководство оператора”. * Написан “Текст программы”. | 15.04.2025 |
| Испытания программы | * Разработка, согласование и утверждение порядка и методики испытаний. * Отладка программы и программной документации по результатам испытаний. | * Написана и утверждена “Программа и методика испытаний”. * Проведены финальные правки программных документов. | 20.04.2025 |
| Подготовка и передача программы | * Подготовка и передача программы и программной документации для сопровождения. | * Написан и отлажен контейнер для программы. * Сервис полностью готов к передаче и установке в необходимой системе. | 01.05.2025 |

**2. Используемый технологический стек и его обоснование**

2.1 Перечень используемых технологий:

| Технология/Инструмент | Описание | Причины выбора |
| --- | --- | --- |
| Golang | Быстрый , простой и эффективный язык, удобный для разработки веб и микро сервисов | Удобное взаимодействие с GRPC, удобен для разработки веб-сервисов |
| gRPC | Высокопроизводительный фреймворк для создания распределённых систем, использующий протобуферы для определения интерфейсов и передачи данных. | Высокая производительность, эффективная сериализация данных, поддержка различных языков программирования |
| MongoDB | NoSQL база данных, хранящая данные в формате BSON (похожем на JSON), обеспечивающая гибкость и масштабируемость. | Гибкая схема данных, масштабируемость, простота горизонтального масштабирования |
| Docker | платформа для контейнеризации, позволяющая упаковывать приложения и их зависимости в изолированные контейнеры для легкого запуска и развертывания. | Легкое развертывание, воспроизводимость среды, изоляция зависимостей |
| Git | это система контроля версий, позволяющая отслеживать изменения в коде и сотрудничать над проектами. | Управление версиями кода, отслеживание изменений, сотрудничество в команде |

2.2 Обоснование выбранного технологического стека:

* Golang: Использование Golang обеспечивает высокую производительность и эффективность обработки JSON-данных. Его встроенная поддержка конкурентности позволяет обрабатывать множество запросов одновременно, что критично для сервиса с высокой нагрузкой. Простота и читаемость кода Go способствует быстрой разработке и сопровождению проекта.
* gRPC: gRPC выбран в качестве фреймворка для межсервисного взаимодействия благодаря его высокой производительности и эффективности. Использование протобуферов обеспечивает быструю сериализацию/десериализацию JSON-данных, что критически важно для производительности сервиса.
* MongoDB: MongoDB идеально подходит для хранения JSON-данных благодаря своей гибкой схеме. Это позволяет легко адаптироваться к изменениям структуры данных без необходимости модификации схемы базы данных. Масштабируемость MongoDB обеспечивает возможность обработки больших объемов данных и высоких нагрузок.
* Docker: Использование Docker позволяет легко упаковывать и развертывать сервис, обеспечивая воспроизводимость среды и упрощая процесс развертывания на различных платформах. Контейнеризация способствует стабильности работы сервиса и упрощает его масштабирование.
* Git: Git используется для управления версиями кода, что позволяет отслеживать изменения, сотрудничать в команде и легко возвращаться к предыдущим версиям кода при необходимости.

**3. Критерии оценивания проекта**

| **Критерий** | **Описание** |
| --- | --- |
| Работа с базой данных | Будут использованы |
| Функциональность - Процент выполнения функциональных требований | Выполненные требования в процентах от общего количества |
| Функциональность - Количество реализованных функций | Абсолютное количество функций, которые работают правильно |
| Производительность и эффективность - Время отклика | Среднее время ответа программы на запросы пользователя (в секундах) |
| Производительность и эффективность - Потребление ресурсов | Среднее потребление памяти (в МБ) и процессорного времени (в %) |
| Качество кода - Покрытие кода тестами (%) | Процент строк кода, покрытых тестами |
| Документация и оформление - Полнота документации (%) | Процент от требуемого объема документации |
| Тестирование - Количество написанных тестов | Общее количество юнит-тестов, интеграционных тестов и других видов тестов |
| Тестирование - Процент успешных тестов (%) | Процент успешно пройденных тестов из общего количества |
| Совместимость и кроссплатформенность - Количество поддерживаемых платформ/ОС | Количество платформ, на которых работает проект |
| Совместимость и кроссплатформенность - Количество интеграций | Число успешных интеграций с другими системами или сервисами |
| Соблюдение сроков и плана - Процент выполнения работы в срок (%) | Процент задач, выполненных в срок |
| Соблюдение сроков и плана - Количество дней отклонения от плана | Общее число дней отклонения от плана |
| Использование технологического стека - Процент использования функциональности стека (%) | Процент использования функциональности выбранного стека технологий |
| Оценка командной работы - Среднее время коммуникации (в часах) | Среднее время, потраченное на обсуждение задач и решение вопросов |
| Оценка командной работы - Количество завершенных задач на каждого участника | Общее число задач, выполненных каждым членом команды |

**4. Особые пометки**

* Мониторинг производительности: Необходимо реализовать систему мониторинга производительности сервиса, чтобы отслеживать задержки, количество обрабатываемых запросов и другие ключевые метрики. Это позволит своевременно выявлять и устранять узкие места и обеспечивать стабильную работу сервиса.
* Обработка ошибок: Сервис должен надежно обрабатывать различные типы ошибок, включая ошибки в формате входных JSON-данных, ошибки подключения к базе данных и другие. Необходимо реализовать механизмы логирования ошибок и уведомления администраторов о критических событиях.
* Безопасность: Необходимо обеспечить безопасность сервиса, защитив его от различных угроз, таких как SQL-инъекции (хотя и используется MongoDB, всё равно следует придерживаться безопасной практики), DoS-атаки и другие. Для этого следует использовать проверку входных данных, защиту от переполнения буфера и другие меры безопасности.
* Тестирование: Необходимо провести всестороннее тестирование сервиса, включая юнит-тесты, интеграционные тесты и нагрузочное тестирование. Это позволит выявить и устранить ошибки на ранних этапах разработки и обеспечить высокое качество сервиса.
* Масштабируемость: Хотя выбранный стек технологий (MongoDB, Golang с goroutines) ориентирован на масштабируемость, необходимо заранее продумать стратегию масштабирования сервиса при увеличении нагрузки. Это может включать в себя использование балансировки нагрузки, кэширования и других методов оптимизации.
* Документация: Необходимо вести подробную документацию как по коду, так и по архитектуре сервиса. Это упростит дальнейшее развитие и поддержку проекта.
* Зависимости: Необходимо отслеживать версии используемых библиотек и фреймворков, чтобы избежать проблем с совместимостью в будущем. Регулярное обновление зависимостей — важная часть поддержки проекта.