Untitled page 2024-12-25 (3)

Техническое задание: Разработка сервиса управления задачами с аналитикой

Цель: Разработать веб-приложение для управления задачами с расширенными функциями аналитики. Выполнение задания должно занять 3 дня.

Функциональные требования

1. **Управление задачами**:
   - Пользователь может:
     - Добавить новую задачу с полями:
       - `id` (уникальный идентификатор).
       - `title` (название задачи).
       - `description` (опционально).
       - `status` (pending, in_progress, done).
       - `priority` (low, medium, high).
       - `due_date` (дата завершения).
       - `created_at` (дата создания, автоматически).
       - `updated_at` (дата обновления, автоматически).
     - Редактировать задачи.
     - Удалять задачи.
   - Автоматическое удаление задач, просроченных на 7 дней.
2. **Фильтрация и поиск**:
   - Возможность фильтровать задачи по:
     - Статусу.
     - Приоритету.
     - Дате завершения.
   - Поиск задач по названию.
3. **Аналитика**:
   - Подсчет:
     - Количества задач в каждом статусе.
     - Среднего времени выполнения задач.
   - Генерация отчета за последние 7 дней:
     - Количество завершенных задач.
     - Количество просроченных задач.
4. **Авторизация и аутентификация**:
   - JWT-аутентификация.
   - Регистрация и вход для пользователя.
5. **Импорт/экспорт задач**:
   - Экспорт всех задач в формате JSON.
   - Импорт задач из файла JSON.

Технические требования

1. **Язык программирования**: Golang.
2. **База данных**: PostgreSQL.
3. **Архитектура**:
   - Применение паттерна Clean Architecture.
4. **Инфраструктура**:
   - Приложение должно быть завернуто в Docker-контейнер.
   - Использовать `docker-compose` для запуска:
     - Приложения.
     - PostgreSQL.
   - Redis для кэширования аналитики.
5. **API**:
   - Реализовать REST API с использованием фреймворка (например, Gin или Fiber):
     - **POST /auth/register** — регистрация пользователя.
     - **POST /auth/login** — вход и получение JWT.
     - **GET /tasks** — получение списка задач с фильтрацией и поиском.
     - **POST /tasks** — добавление задачи.
     - **PUT /tasks/:id** — обновление задачи.
     - **DELETE /tasks/:id** — удаление задачи.
     - **GET /analytics** — получение аналитики.
     - **POST /tasks/import** — импорт задач из JSON.
     - **GET /tasks/export** — экспорт задач в JSON.
6. **Асинхронность**:
   - Использовать горутины для:
     - Удаления устаревших задач (каждые 24 часа).
     - Генерации аналитики (раз в 6 часов).
   - Использовать каналы и мьютексы для синхронизации данных.
7. **Тесты**:
   - Покрытие unit-тестами следующих компонентов:
     - CRUD-операции с задачами.
     - Аналитика.
   - Интеграционные тесты для API.

Что предоставить

1. Исходный код проекта.
2. `README.md` с инструкциями:
   - Как запустить приложение через `docker-compose`.
   - Примеры API-запросов (curl или Postman).
3. SQL-скрипты или миграции для базы данных.
4. Пример файла JSON для импорта/экспорта задач.
5. Swagger-документацию для API.

Критерии оценки

1. **Код и архитектура**:
   - Чистота и читаемость кода.
   - Применение Clean Architecture.
2. **API**:
   - Корректность работы всех запросов.
   - Наличие фильтров и поиска.
3. **Асинхронность и производительность**:
   - Реализация периодических задач.
   - Использование кэширования (Redis).
4. **Документация**:
   - Понятные инструкции по запуску.
   - Полная API-документация.
5. **Тестирование**:
   - Покрытие ключевых модулей тестами.

Дополнительно

Будет плюсом:

- Логирование (zerolog или logrus).
- Поддержка локализации (например, для полей `priority`).
- Мониторинг приложения (Prometheus + Grafana).

Срок выполнения: 3 дня.