Техническое задание

REST-сервис управления пользовательскими подписками

1. Общая информация

Необходимо спроектировать и реализовать REST-сервис для управления и агрегации данных о пользовательских подписках на онлайн-сервисы.

Сервис предоставляет API для:

- управления подписками (создание, получение, обновление, удаление, список),
- расчёта суммарной стоимости подписок за заданный период с фильтрацией.

Сервис является **stateless**, не управляет пользователями и работает исключительно с данными подписок.

---

2. Архитектура и принципы проектирования

Сервис реализуется в соответствии с принципами **Clean Architecture** и использует **трёхслойную архитектуру**:

```
HTTP Handler  →  Service (Business Logic)  →  Repository (DB)
```

Основные принципы:

- Чёткое разделение ответственности между слоями
- Внедрение зависимостей (Dependency Injection) на уровне `main`
- Использование интерфейсов для `service` и `repository`
- Отсутствие бизнес-логики в HTTP-слое
- Передача `context.Context` через все уровни приложения

---

3. Модель данных «Подписка»

Каждая запись о подписке содержит следующие поля:

Поле	Описание
`id`	Уникальный идентификатор подписки (UUID)
`service_name`	Название сервиса
`price`	Стоимость месячной подписки (целое число рублей)
`user_id`	Идентификатор пользователя (UUID)
`start_date`	Дата начала подписки (формат `MM-YYYY`)

Поле	Описание
`end_date`	Дата окончания подписки (опционально, формат `MM-YYYY`)

Ограничения:

- Стоимость подписки — **целое число рублей**, копейки не учитываются
- Проверка существования пользователя **не выполняется**
- Управление пользователями находится вне зоны ответственности сервиса

---

4. HTTP API (CRUDL + агрегация)

4.1 CRUDL операции

Метод	Endpoint	Описание	Статусы
POST	`/subscriptions`	Создание подписки	`201 Created`, `400`
GET	`/subscriptions/{id}`	Получение подписки по ID	`200 OK`, `404 Not Found`
PUT	`/subscriptions/{id}`	Обновление подписки	`200 OK`, `400`
DELETE	`/subscriptions/{id}`	Удаление подписки	`204 No Content`
GET	`/subscriptions`	Список подписок	`200 OK`

Поддержка:

- Пагинации (`limit`, `offset`)
- Фильтрации по `user_id` и `service_name`

4.2 Агрегация подписок

Endpoint:

```
GET /subscriptions/summary
```

Назначение:
Подсчёт суммарной стоимости подписок за заданный период времени.

Параметры:

- `from` — начало периода (`MM-YYYY`)
- `to` — конец периода (`MM-YYYY`)
- `user_id` — фильтрация по пользователю (опционально)
- `service_name` — фильтрация по сервису (опционально)

Ответ:

```
{
  "total": 1200
}
```

---

5. Валидация данных

- Используется библиотека `go-playground/validator`
- Валидируются:
  - формат дат (`MM-YYYY`)
  - обязательные поля
  - минимальная длина строк
  - корректность UUID
- Ошибки валидации возвращаются с HTTP 400

---

6. Работа с базой данных

СУБД

- PostgreSQL

Миграции

- Используется библиотека **Goose**
- Реализованы **2 SQL-миграции**:
  - `0001_init_subscriptions.sql` — создание таблиц
  - `0002_add_indexes.sql` — добавление индексов
- Поддержка `up` / `down`
- Миграции выполняются автоматически при старте приложения

Индексы

- Индексы по:
  - `user_id`
  - `service_name`
  - диапазонам дат

---

7. Логирование

Реализовано логирование:

- HTTP-запросов (middleware `LoggingMiddleware`)
- На всех слоях приложения:
    - `handler`
    - `service`
    - `repository`

Уровни логов:

- `INFO`
- `ERROR`
- `DEBUG`

---

8. Конфигурация

- Используется `Viper`
- Конфигурация:
    - `config.yml` — основные параметры приложения
    - `.env` — чувствительные данные (пароли)
- Поддержка переопределения параметров через переменные окружения
- Раздельные настройки для production и тестов

---

9. Graceful Shutdown

- Реализовано корректное завершение работы сервиса
- Обработка `SIGINT` / `SIGTERM`
- Таймаут завершения — **5 секунд**
- Корректное закрытие HTTP-сервера и DB-соединений

---

10. Тестирование

Типы тестов:

- **Unit-тесты** сервисного слоя (с моками)
- **Integration-тесты**:
    - repository
    - HTTP handlers

- Используется реальная PostgreSQL БД (через Docker)

---

11. CI/CD

- GitHub Actions
- Pipeline включает:
  - проверку сборки
  - запуск тестов
  - базовую валидацию проекта

---

12. Документация

- Swagger (OpenAPI)
- Генерация через `swag`
- Swagger UI доступен по `/swagger/index.html`
- Все HTTP-эндпоинты аннотированы
- Добавлены **Godoc-комментарии** для публичных структур и методов

---

13. Пример запроса на создание подписки

```
{
  "service_name": "Yandex Plus",
  "price": 400,
  "user_id": "60601fee-2bf1-4721-ae6f-7636e79a0cba",
  "start_date": "07-2025"
}
```

---

14. Технологический стек

- Go
- PostgreSQL
- pgx
- Goose
- chi
- Viper
- go-playground/validator
- Swagger (swag)
- Docker / Docker Compose
- GitHub Actions