Тестовое задание №2 (backend)

Создание кеширующего веб-сервера

Задача

Следуя описанию и рекомендациям написать веб-сервер для сохранения и раздачи электронных документов. Вы можете выбрать СУБД для реализации приложения по своему усмотрению.

Язык реализации задания

go

Описание приложения

Реализовать http-сервер, который реализует описанное ниже REST API.

1. Регистрация (логин, пароль – создание нового пользователя)
2. Аутентификация (получить токен авторизации по логину и паролю)
3. Загрузка нового документа
4. Получение списка документов
5. Получение одного документа
6. Удаление документа
7. Завершение авторизованной сессии работы

Пользователь может загружать файлы, управлять коллекцией и выборочно делиться ими с другими. При этом система должна рассчитываться на значительную нагрузку при получении списков и файлов. Запросы HEAD не должны возвращать данных. Запросы GET/HEAD должны отдаваться из внутреннего кеша. Остальные запросы должны инвалидировать кеш, желательно выборочно.

Уровни сложности:

- **Первый**: запросы 4 и 5.
- **Второй**: добавить 3 и 6.
- **Третий**: все запросы.
- **Четвертый**: все запросы + выдача результата из кэша (если есть)

Общая модель ответа для всех методов:

```
{
   "error": {
       "code": 123,
       "text": "so sad"
   },
   "response": {
       ...
   },
   "data": {
       ...
   }
}
```

- Поля *error, response, data* присутствуют, только если заполнены
- Поле *response* для подтверждения действий
- Поле *data* для выдачи содержимого
- HTTP-статусы:
    - Все ок — 200
    - Некорректные параметры — 400
    - Не авторизован — 401
    - Нет прав доступа — 403
    - Неверный метод запроса — 405
    - Нежданчик — 500
    - Метод не реализован - 501

Описание REST API:

1. Регистрация [POST] /api/register

- Вход
    - *token* — токен администратора
        - Фиксированный, задается в конфиге приложения
    - *login* — логин нового пользователя
        - Минимальная длина 8, латиница и цифры
    - *pswd* — пароль нового пользователя
        - минимальная длина 8,
        - минимум 2 буквы в разных регистрах
        - минимум 1 цифра
        - минимум 1 символ (не буква и не цифра)

- Выход

```
{
   "response": {
       "login": "test"
   }
}
```

2. **Аутентификация [POST] /api/auth**
   - Вход - форма
     - *login*
     - *pswd*
   - Выход

```
{
   "response": {
       "token": "sfuqwejqjoiu93e29"
   }
}
```

3. **Загрузка нового документа [POST] /api/docs**
   - Вход — multipart form
     - *meta* — параметры запроса. Модель:

```
{
   "name": "photo.jpg",
   "file": true,
   "public": false,
   "token": "sfuqwejqjoiu93e29",
   "mime": "image/jpg",
   "grant": [
       "login1",
       "login2",
   ]
}
```

     - *json* — данные документа
       - может отсутствовать
       - модель не определена
     - *file* — файл документа
   - Выход

```
{
   "data": {
       "json": { ... },
       "file": "photo.jpg"
   }
}
```

4. **Получение списка документов [GET, HEAD] /api/docs**
   - Вход

- *token*
- *login* - опционально — если не указан — то список своих
- *key* - имя колонки для фильтрации
- *value* - значение фильтра
- limit - кол-во документов в списке

- Выход - сортировать по имени и дате создания

```
{
   "data": {
       "docs": [
           {
               "id": "qwdj1q4o34u34ih759ou1",
               "name": "photo.jpg",
               "mime": "image/jpg",
               "file": true,
               "public": false,
               "created": "2018-12-24 10:30:56",
               "grant": [
                   "login1",
                   "login2",
               ]
           }
       ]
   }
}
```

5. Получение одного документа [GET, HEAD] /api/docs/<id>

- Вход
  - *token*
- Выход
  - Если файл — выдать файл с нужным mime
  - Если JSON:

```
{
   "data": {
       ...
   }
}
```

6. Удаление документа [DELETE] /api/docs/<id>

- Вход
  - *token*
- Выход

```
{
   "response": {
       "qwdj1q4o34u34ih759ou1": true
   }
}
```

7. **Завершение авторизованной сессии работы [DELETE] /api/auth/<token>**
    - Выход

```
{
   "response": {
       "qwdj1q4o34u34ih759ou1": true
   }
}
```

Библиотеки для реализации (*возможно использовать и другие)

- HTTP
    - https://golang.org/pkg/net/
    - https://golang.org/pkg/net/http/
    - https://golang.org/pkg/net/url/
    - https://golang.org/pkg/mime/
    - https://golang.org/pkg/net/http/httptest/
    - https://golang.org/pkg/mime/multipart/
    - https://godoc.org/github.com/gocraft/web
    - https://godoc.org/github.com/jarcoal/httpmock
- Утилиты
    - https://golang.org/pkg/regexp/
    - https://golang.org/pkg/sync/
    - https://godoc.org/github.com/satori/go.uuid
    - https://godoc.org/github.com/pkg/errors
- Базы данных
    - https://golang.org/pkg/database/sql/
    - https://godoc.org/github.com/lib/pq
    - https://github.com/jackc/pgx
    - https://godoc.org/github.com/globalsign/mgo
    - https://godoc.org/github.com/jmoiron/sqlx
    - https://godoc.org/github.com/gwenn/gosqlite