**Разработка backend части для редактора схем диалпланов астериска**

**Параметры подключения по ssh к серверу телефонии:**

**Хост:**

**Логин:**

**Пароль:**

**Корневая директория:** /home/scripts

**Суть задачи:** в настоящий момент, мы используем редактор схем диалпланов доступный по адресу: [http://dialplan.editor.test.nextcontact.ru](http://dialplan.editor.test.nextcontact.ru/). Он используется для визуализации диалпланов в более наглядном графическом виде, нежели код на астериске. В нем мы составляем схемы диалпланов, которые здесь же преобразуются в код asterisk и в дальнейшем загружаются в конфигурацию на сервер телефонии. В качестве бэкенд части у данного проекта используется php. В виду того, что со времени релиза проекта, конфигурация астериска и подход к разработке функционала для него — существенно изменился и часть функционала стала не актуальной. Так, например, фаловый менеджер, встроенный в проект, смотрит на более неактуальные сервера телефонии. Черный список абонентов, исользует устаревшую БД, конвертация аудиофайлов более не работает. Нужно актуализировать бэкенд часть под текущие реалии и переписать его на golang, как основной использующийся в отделе ЯП для бэкенда.

Пока что, необходимо реализовать следующие моменты:

1) Развернуть основу для новой версии

2) Реализовать менеджер файлов

3) Конвертация аудио-файлов

**1. Развернуть основу для новой версии**

Бэкенд часть должна представлять собой классический http-сервер, с использованием библиотеки gin для приема запросов и clean arch в качестве архитектуры проекта. Для логирования просьба использовать zerolog. Http-запросы так же должны писать длительность своей работы в jager.

**2. Менеджер файлов**

Менеджер файлов доступен в редакторе через пункт меню Файлы → Менеджер файлов. Его задачей, является отображение всех находящихся в **корневой директории** (см. Параметры подключения по ssh…) файлов и поддиректорий, возможность закачки фалов на сервер, конвертация аудиофайлов в форматы raw и wav (перевариваемые астериском), а также редактирование файлов с кодом (на даный момент только python и bash).

Необходимо реализовать следующие запросы:

1) **/get-files**

**Функционал:**

**Метод**: Get

**Принимаемые параметры:**

**path** - Директория, содержимое которой нужно вывести. Если пустая или отсутствует — вывести содержимое корневой директории. Если присутствует — выводим содержимоепо пути: *корневая\_директория/path*

**server** — Необязательно. Номер сервера, к которому будем подключаться. Сейчас, пока что используем только 1 сервер, но по факту у нас их несколько, по этому на будущее, нужно сделать задел, для возможности выбора конкретного сервера и работать уже непосредственно с ним.

**Пример ожидаемого ответа (json):**

{

"result": true,

"message": "",

"path": "test/sounds",

"content": [

{

"type": "d",

"changed": "2023-07-12 12:53",

"name": "test",

"size": "4.0K"

},

{

"type": "f",

"changed": "2022-06-20 12:56",

"name": "1\_1\_1.wav",

"size": "208K"

}

]

}

где:

*result* — булево значение, результат выполнения запроса, с возвращением true, если запрос корректно отработал и false, если возникли какие-либо ошибки.

*message* — описание ошибки в случае ее возникновния либо какая-то дополнительная информация от бэкенда.

*path* — относительный от корневой директории путь, по которому был выведен список файлов и директорий.

*сontent* — массив со списком файлов и директорий. В массиве содержатся объекты следующего содержания:

*type* — тип объекта. f — в случае, если это файл. d — в случае, если это директория.

*сhandeg* — время последнего изменения в формате гггг-мм-дд чч:ми

*name* — полное название файла с расширением

*size* — размер файла с человеко-понятным округлением по размерам. Правила округления:

- файл менее 1Кб — выводить размер в байтах с постфиксом B.

- Файл менее 1Мб и более или равен 1Кб — выводить размер в килобайтах с постфиксом K.

- Файл менее 1Гб и более или равен 1Мб — выводить размер в мегабайтах с постфиксом М.

- Файлы от 1Гб и более — выводить размеры в гигабайтах с постфиксом Г.

2) **/create-dir**

**Функционал:** Создание новой директории по указаному пути

**Метод**: Get

**Принимаемые параметры:**

**path** - Директория, в которой нужно создать дочернюю директорию. Если пустая или отсутствует — создаем в корневой директории.

**dir\_name** — Название новой директории.

**Пример ожидаемого ответа (json):**

{

"result":true,

"message":""

}

где:

*result* — булево значение, результат выполнения запроса, с возвращением true, если запрос корректно отработал и false, если возникли какие-либо ошибки.

*message* — описание ошибки в случае ее возникновния либо какая-то дополнительная информация от бэкенда.

Примечание: вообще, по хорошему, тут бы использовать метод Post вместо Get, но на фронте уже используется get. Не знаю под какими веществами я это делал в те далекие времена, но когда дойдут руки до правок во фронтенде — хотелось бы сделать все как надо) Поэтому просьба сделать еще и post метод, который делал бы тоже самое.

3) **/upload-files**

**Функционал:** Закачка файлов на сервер телефонии.

**Метод**: Post

**Принимаемые параметры:**

**file\_0 … file\_n** — файлы в бинарном виде. Может быть передано несколько файлов, при этом перый файл будет называться file\_0, последующий file\_1 и т.д.

**path** - Директория, в которой нужно создать дочернюю директорию. Если пустая или отсутствует — создаем в корневой директории.

**convert\_list** — Массив с названием файлов, которые необходимо будет конвертировать. (Подробнее о конвертации в разделе 3. Конвертация аудио-файлов)

Закачиваемые файлы с названиями на русском языке — транслитерировать, т.к. иначе в дальнейшем могут быть проблемы, если это например аудио-файл, который планируется подставлять в диалплан астериска. Пробелы и спецсимволы также заменять на нижнее подчеркивание либо похожие буквы латинского алфавита (№ на N, $ на S).

В php функция занимающаяся очисткой названий от лишних символов выглядела так:

*public static function* clearSpecialSymbols(*string* $text):*string* {  
 $symbols = *array*('!', '@', '"', '#', '№', '$', '%', '^', '{', '}','(',')',"'",'~','`',' ','«','+','=','[',']');  
 $changes = *array*('\_', '\_', '\_', '\_', 'N', 'S', '\_', '\_', '\_', '\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_','\_');  
  
 *return str\_replace*($symbols, $changes, $text);  
}

Тут каждому символу из массива $symbols под тем же индексом соответствует символ замены в массиве $changes. Рекомендую заменять их такими же символами как и в этом примере.

**Пример ожидаемого ответа (json):**

{

"result":true,

"message":""

}

где:

*result* — булево значение, результат выполнения запроса, с возвращением true, если запрос корректно отработал и false, если возникли какие-либо ошибки.

*message* — описание ошибки в случае ее возникновния либо какая-то дополнительная информация от бэкенда.

4) **/get-audio**

**Функционал:** Отдача аудио-файла в бинарном виде на фронтенд.

**Метод**: Get

**Принимаемые параметры:**

**file** — название файла на сервере

**path** - путь до файла, относительно корневой директории

**Ожидаемый ответ:** аудио-файл в бинарном виде

5) **/get-script**

**Функционал:** Отдать содержимое файла скрипта для редактирования.

**Метод**: Get

**Принимаемые параметры:**

**file** — название файла на сервере

**path** - путь до файла, относительно корневой директории

**Пример ожидаемого ответа (json):**

{

"result":true,

"message":"",

"content":"print(1)",

}

где:

*result* — булево значение, результат выполнения запроса, с возвращением true, если запрос корректно отработал и false, если возникли какие-либо ошибки.

*message* — описание ошибки в случае ее возникновния либо какая-то дополнительная информация от бэкенда.

*content* — текстовое содержимое файла.

6) **/update-script**

**Функционал:** Обновление содержимого файла скрипта.

**Метод**: Post

**Принимаемые параметры:**

**file** — название файла на сервере

**path** - путь до файла, относительно корневой директории

**content** - новое содержимое, которым нужно заменить текущее содержимое указанного файла

**Пример ожидаемого ответа (json):**

{

"result":true,

"message":""

}

где:

*result* — булево значение, результат выполнения запроса, с возвращением true, если запрос корректно отработал и false, если возникли какие-либо ошибки.

*message* — описание ошибки в случае ее возникновния либо какая-то дополнительная информация от бэкенда.

**3. Конвертация аудио-файлов**

При закачке файлов (метод /upload-files), мы можем указывать массив аудио-файлов, которые нужно конвертировать в читаемые для астериска форматы. Мы используем 2 формата: raw и wav. На момент написания первой версии приложения, предполагалось, что будем использовать только wav, т. к. он более удобен к применению — его можно воспроизвести в любом аудио-плеере без лишних трудностей, в отличии от raw, для воспроизведения которого, нужны специальные редакторы вроде Audacity с указанием корректных настроек воспроизведения. Поэтому мы приводили все к wav формату. Команда ffmpeg выглядела для файлов mp3, ogg и wav (многоканального) следующим образом:

ffmpeg -i *source\_file* -acodec pcm\_s16le -ar 8000 -vol 550 -ac 1 -y *output\_file*

К wav приводили в том числе и raw файлы астериска, в таком случае, команда выглядела так:

ffmpeg -f s16le -ar 8k -ac 1 -i *source\_file* *output\_file*

Однако, как показала практика, сам астериск лучше работает с raw файлами.

Поэтому, на первом этапе, предлагаю повторить уже существующий функционал конвертации всего к wav чтобы натянуть уже существующий фронтенд на новый бэкенд без его кардинального переписывания.

Но предусмотреть дополнительную, необязательную опцию — к какому формату приводить файл — raw или wav. По-умолчанию, приводить к wav. Но также добавить возможность приведения и к raw с частотой 8000 Гц и моноканалом. (Подробности потом можно будет посмотреть вместе, т. к. тут пока я сам точные настройки для конвертации не знаю, нужно будет поэксперементировать).