

Министерство науки и высшего образования Российской Федерации
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ»
ИНСТИТУТ ИНТЕЛЛЕКТУАЛЬНЫХ КИБЕРНЕТИЧЕСКИХ СИСТЕМ
КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И ТЕХНОЛОГИИ

На правах рукописи
УДК 004.457

ОРЛОВ ЯРОСЛАВ АНДРЕЕВИЧ

РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ СОПРОВОЖДЕНИЯ
API ДОКУМЕНТАЦИИ

Выпускная квалификационная работа бакалавра

Направление подготовки 09.03.01 Информатика и вычислительная техника

Выпускная квалификационная
работа защищена

«__»_____2021 г.

Оценка _____

Секретарь ГЭК _____

г. Москва

2021

Студент-дипломник	_____	/ Орлов Я.А. /
Руководитель работы	_____	/ Галаев А.О. /
Рецензент	_____	/ Овчаренко Е.С. /
Заведующий кафедрой №12	_____	/ Иванов М.А. /

АННОТАЦИЯ

Содержание пояснительной записки: 44 страниц, 8 рисунков, 3 таблиц, 10 используемый источник литературы.

Целью работы является разработка автоматизированной системы сопровождения API-документации.

В ходе работы над ВКР был проведен анализ аналогов и прототипов. Составлен перечень критериев качества, по которым происходило сравнение систем. Выбран и обоснована система Postman с реализацией дополнительного ПО в виде скрипта публикации API-документации.

В первой части проводится общий анализ аналогов и прототипов.

Во второй части определяются функциональные требования.

В третьей части пояснительной записки.

В заключении подводятся итоги разработанной системы сопровождения API документации.

СОДЕРЖАНИЕ

АННОТАЦИЯ.....	3
СОДЕРЖАНИЕ	4
ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ.....	6
ВВЕДЕНИЕ	7
1. ОБЗОРНАЯ ЧАСТЬ	9
1.1. Описание предметной области.....	9
1.1.1. Описание серверной архитектуры.....	9
1.2. Исследование существующих технологий	10
2.3.1. Перечень функций, подлежащих автоматизации	10
2.3.2. Выбор и обоснование критериев качества	11
2.3.3. Анализ аналогов и прототипов	12
1.2.3.1. Swagger.....	12
1.2.3.2. API Blueprint	15
1.2.3.3. Ручной метод сопровождения API-документации	16
1.2.3.4. Postman	18
2.3.4. Сравнение аналогов и прототипов	20
2.3.5. Вывод.....	22
2. РАСЧЕТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ	22
2.1. Определение требований к системе	22
2.2. Разработка структуры автоматизированной системы	23
2.3. Выбор языка программирования	24
2.4. Разработка интерфейса взаимодействия пользователя с системой.....	24
2.3.1. Создание структуры документов в Postman.....	25
2.3.2. Публикация документации с помощью CLI.....	32
2.5. Разработка алгоритмов программных модулей	35
2.6. Разработка плана проведения тестирования.....	37
3. ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ	38
3.1. Реализация разработанных алгоритмов	38
3.2. Тестирование и отладка системы.....	40

3.2.1. Тестирование алгоритма установки программного обеспечения	40
3.2.2. Тестирование алгоритма создания структуры и заполнения форм API методов в интерфейсе Postman.....	41
3.2.3. Тестирование алгоритма выгрузки структуры документации на локальный компьютер	41
3.2.4. Тестирование алгоритма публикации документации в Confluence .	42
ЗАКЛЮЧЕНИЕ	42
СПИСОК ЛИТЕРАТУРЫ.....	44
ПРИЛОЖЕНИЕ А: Листинг программы публикации документации	45

ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ

- API-документация – это техническая документация, в которой фиксируются инструкции о том, как использовать программное API.
- CLI (Command line interface) – Интерфейс командной строки
- JSON (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript
- REST API – это набор правил, по которым следует обращаться к серверу для отправки или получения данных.
- TT (Template Toolkit) – perl-библиотека для работы с шаблонами, позволяющая разделять код, данные и представление
- UI (User Interface) – пользовательский интерфейс
- URL (Uniform Resource Locator) – адрес сайта или отдельной страницы в сети интернет
- АС – автоматизированная система
- Клиент – любое приложение которое делает запросы на сервер. Например, в роли клиента может выступать веб браузер, когда пользователь открывает веб-сайт
- ПО – Программное обеспечение
- ЯП – язык программирования

ВВЕДЕНИЕ

На сегодняшний день большинство крупных IT компаний для взаимодействия сервера и клиента используют REST API [1].

Компании вроде Яндекса, Google и т.п. Предоставляют открытые API методы своих сервисов чтобы разработчики могли интегрироваться с ними.

Например, при получении данных о пользователе, информация о котором храниться в БД (базе данных) на сервере необходимо указать путь до сервера (URI), идентификатор пользователя (ID) и метод (Method) по которому сервер поймет, что нужно сделать с ресурсом, в данном случае вернуть информацию о пользователе. Данный процесс «общения» клиента и сервера, представлен на рисунке 1.1.

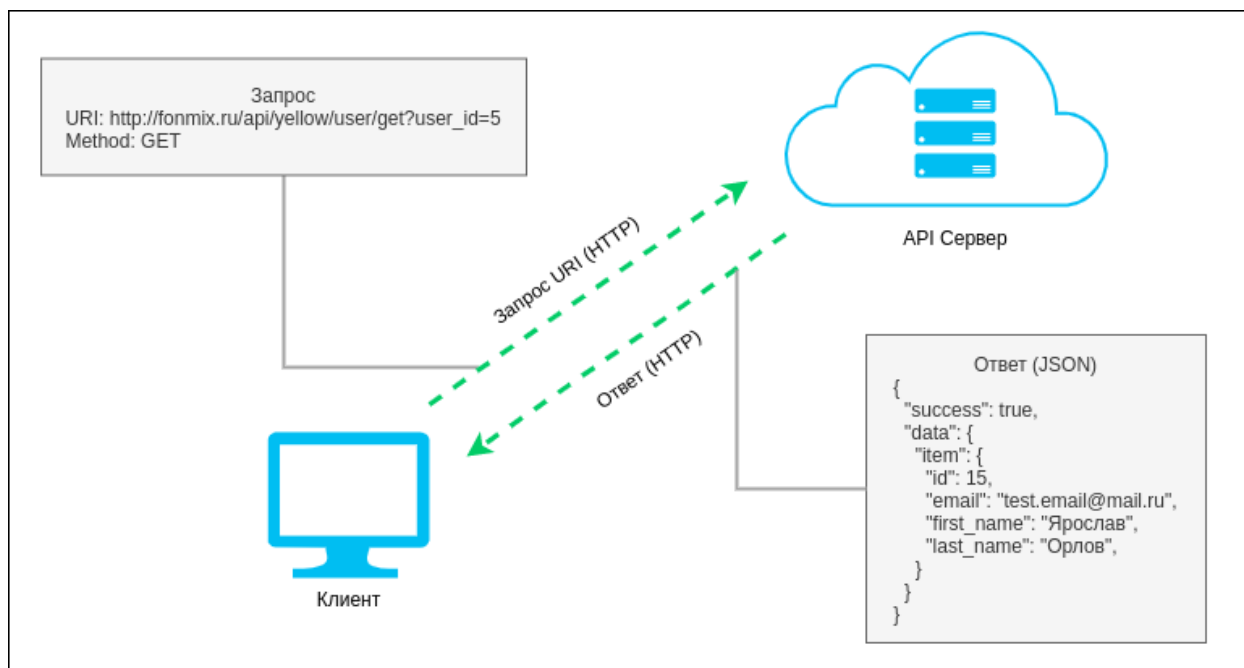


Рисунок 1.1 – Схема получения информации о пользователе

Помимо написания самих API методов необходимо написание подробной документации по ним, поскольку без нее попросту не удастся воспользоваться методом. А также не менее важно поддерживать документацию в актуальном

состоянии поскольку если документация будет неправильная или устаревшая, то велика вероятность ошибок и в конечном итоге может сказываться на качестве и стоимости продуктов. Поэтому написание API-документации очень важная и актуальная тема.

Компания ООО «ФорМакс» разрабатывает продукт Fonmix, серверная часть которого полностью базируется на технологии REST API, то есть взаимодействие любого пользователя с сервером Fonmix осуществляется через REST API.

Основными клиентами [3] для сервера Fonmix являются:

- 1) Веб-сайт fonmix.ru – представляет собой веб интерфейс, в котором пользователи [2] могут управлять музыкой в своих заведениях: создавать плейлисты, составлять музыкальное расписание, добавлять рекламу в перерывах между песнями и т.п.
- 2) FM.Player – кроссплатформенный медиа проигрыватель разрабатываемый также в компании ООО «ФорМакс», с помощью которого воспроизводится медиа контент правообладателей.
- 3) Правообладатель – это исполнитель и изготовитель фонограмм, с которым заключается договор о дистрибуции контента и предоставлении отчетов об использовании.

Целью данной работы является создание системы автоматического сопровождения API-документации, позволяющей ускорить и повысить качество разработки. В соответствии с поставленной целью, работа над АС (автоматизированной системой) была разделена на несколько этапов, в рамках которых решались следующие задачи:

- анализ предметной области
- обзор и сравнение современных технологий по сопровождению API-документации

- выделение перечня функций, подлежащих автоматизации
- определение требований к системе
- разработка интерфейса взаимодействия пользователя с системой
- разработка алгоритмов и программных модулей системы
- проведение тестирования и отладки системы
- написание руководства для пользователей системы

1. ОБЗОРНАЯ ЧАСТЬ

1.1. Описание предметной области

1.1.1. Описание серверной архитектуры

Серверная часть проекта Fonmix на разделена на микросервисы.

Микросервисная архитектура¹ – вариант сервис-ориентированной архитектуры программного обеспечения, направленный на взаимодействие насколько это возможно небольших, слабо связанных и легко изменяемых модулей – микросервисов.

Основными микросервисами являются:

- FM.Core – Основной сервис для работы с клиентам. Количество API методов 253
- FM.CRM – Сервис для получения данных о пользователях для дальнейшего их анализа. Количество API методов 153
- FM.ID – Сервис для авторизации пользователей. Количество API методов 23
- FM.Notify – Сервис для отправки уведомлений пользователям. Количество API методов 34

¹ https://ru.wikipedia.org/wiki/Микросервисная_архитектура

- FM.Store – Сервис для хранения и обработки файлов пользователей. Количество API методов 15
- FM.Media – Сервис для хранения и распространения медиа контента правообладателей. Количество API методов 36

Итого, общее количество методов 514

Схема взаимодействия клиентов и сервера представлена на рисунке 1.2.

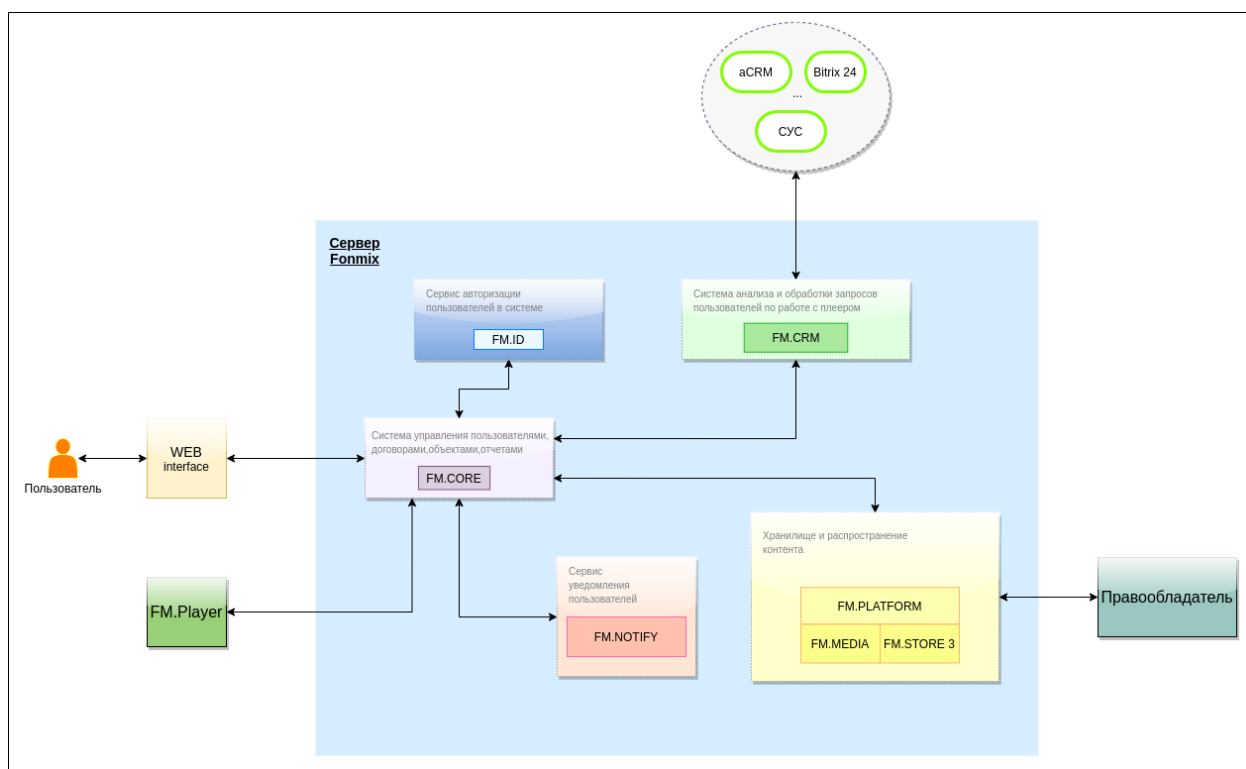


Рисунок 1.2 – Схема взаимодействие клиентов и сервера

1.2. Исследование существующих технологий

2.3.1. Перечень функций, подлежащих автоматизации

В ходе анализа было выявлено, что автоматизации подлежат следующие функции:

- 1) Ввод данных API методов

- 2) Публикация документации в единой системе справочной документации
- 3) Предоставление данных по API методу
- 4) Поиск и фильтрация данных в единой системе справочной документации

2.3.2. Выбор и обоснование критериев качества

Для проведения сравнительного анализа аналогов и прототипов выбраны следующие критерии:

- 1) Трудозатраты на изучение технологии
- 2) Потребность в дополнительном ПО
- 3) Настраиваемость системы
- 4) Время, затрачиваемое на сопровождение документации
- 5) Публикация документации в единую справочную систему (ЕСС) компании

Критерий «Трудозатраты на изучение технологии» определяет уровень трудозатрат для сроков обучения персонала навыками владения новой технологии.

Критерий «Потребность в дополнительном ПО» определяет объем дополнительного ПО для полного сопровождения API-документации.

Критерий «Настраиваемость системы» определяет уровень трудозатрат, требуемых на первичную и дальнейшую настройку системы.

Критерий «Время, затрачиваемое на сопровождение документации» определяет продолжительность времени необходимое на сопровождение документации.

Критерий «Публикация документации в единую справочную систему компании» возможность системы в отображении документации в единой

справочной системе компании. На данный момент вся программная документация по проекту Fonmix храниться вики-системе Confluence²

2.3.3. Анализ аналогов и прототипов

Рассмотрим аналоги и прототипы с точки зрения выбранных критериев качества.

1.2.3.1. Swagger

Swagger представляет собой фреймворк состоящий из нескольких отдельных, независимых утилит

- 1) Swagger Editor – онлайн редактор API-документации. Представляет собой двухоконный текстовый редактор, слева пишется документация на специальном языке разметки YAML. Графический интерфейс Swagger Editor представлен на рисунке 1.3.
- 2) Swagger UI – веб интерфейс для отображения API-документации
- 3) Swagger Codegen – автоматический генератор API-документации на основе исходного кода
- 4) Swagger Hub – предоставляет собой платное программное решение для проектирования, управления и публикации документации API.

² <https://ru.wikipedia.org/wiki/confluence>

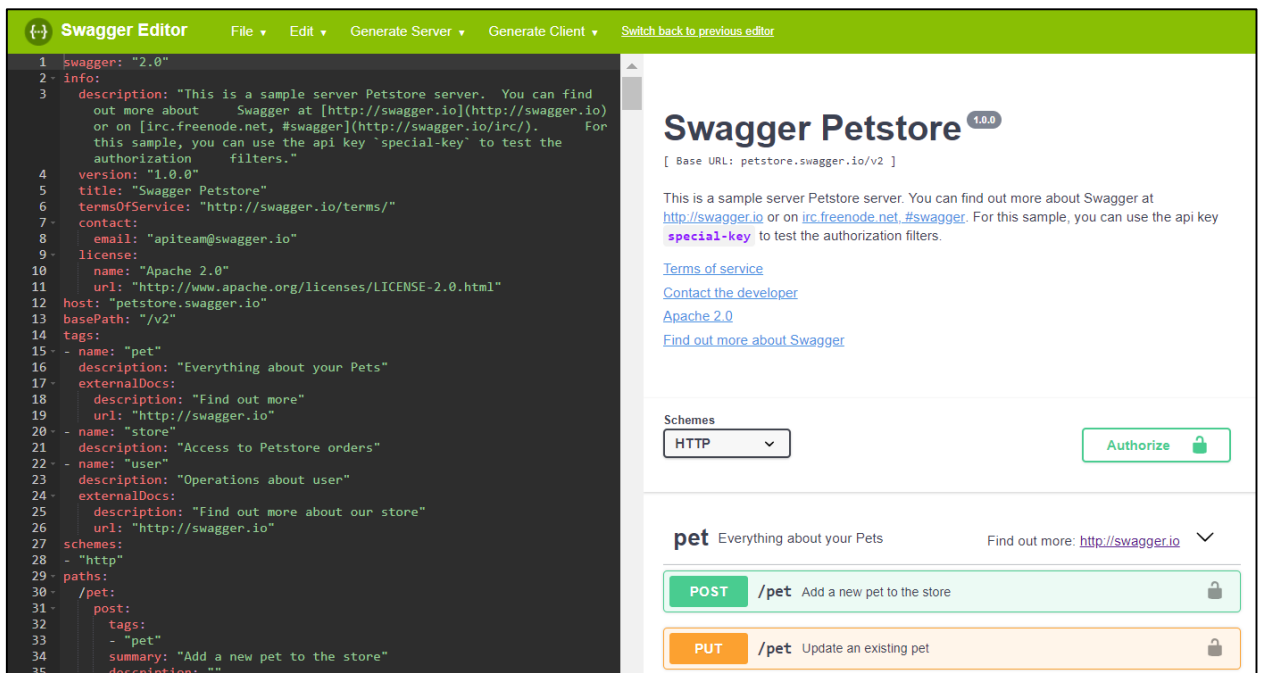


Рисунок 1.3 – Графический интерфейс Swagger Editor

Существует два подхода использования Swagger

- 1) Документация генерируется из комментариев в исходном коде наподобие Javadoc. Отсюда есть ряд существенных недостатков
 - Код становится трудно читаем, даже если комментарии вынесены вне функций или классов
 - При автоматической генерации документации необходимо настраивать CI/CD проекта
- 2) Написание документации отдельно от кода. Данный способ не засоряет исходный код и достаточно гибок поэтому будет рассматривать его

Перед тем как начать писать документацию, необходимо пройти учебное пособие на официальном сайте swagger.

Для того чтобы начать писать документацию необходимо открыть страницу <https://editor.swagger.io/> после чего в левой части можно будет редактировать уже готовую API-документацию.

Для написания документации на персональном компьютере, необходимо установить Swagger Editor и Swagger UI. Так как в Swagger Editor нет интерактивного взаимодействия, пользователь описывает документацию на специальном языке разметки YAML, то стоит также установить Swagger Hub.

Достоинства:

- Основным достоинством является выполнение запросов на сервер непосредственно из браузера. Swagger UI позволяет выполнить запрос и вывести ответ от сервера чтобы продемонстрировать работу API
- Автоматическая генерация клиента на разных языках программирования.
- Создания mock сервера. Это очень удобная возможность описать то как будет работать API до ее фактического написания.

Недостатки:

- Высокий порог вхождения. Необходимо изучать спецификацию Open API на которой базируется Swagger. Необходимо изучить синтаксис по работе со спецификацией Open API.
- Высока вероятность что документирование каких-то сложных API методов будет затруднительно поскольку Swagger рассчитан на базовые, простые API методы
- Явная нехватка формы обратной связи или комментариев к API методам. Если клиент захочет уточнить по поводу API метода, обратить внимание на неточность, опечатку и т.п. то скорее всего нужно будет обращаться непосредственно к разработчику API. Комментарии к документации доступны только при платной подписки на Swagger Hub

1.2.3.2. API Blueprint

API Blueprint представляет собой инструмент для ведения API-документации с использованием специального языка разметки Markdown. Отличительной особенностью от других инструментов является то что можно описывать документацию в довольно гибком формате. Из основных минусов является то что нету автоматической поддержки публикаций документации.

Графический интерфейс API Blueprint представлен на рисунке 1.4.

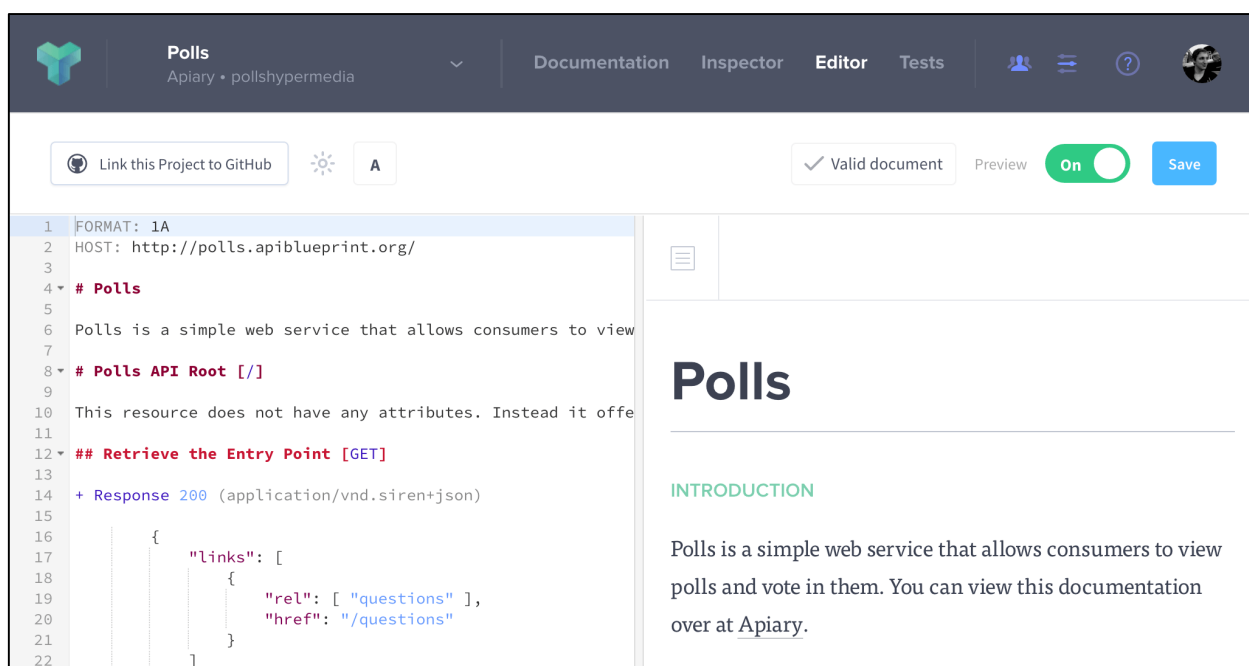


Рисунок 1.4 – Графический интерфейс API Blueprint

Достоинства:

- Удобная навигация по API-документации.
- По сравнению с Swagger у которого используется язык разметки YAML, у Blueprint используется Markdown который лучше человек читаем.
- Также, как и у Swagger есть возможность отправлять запросы на сервер из формы API-документации

Недостатки:

- Необходимо изучать язык разметки и его особенности по работе с API Blueprint
- Нету версионирования API-документации. Последняя опубликованная документация является самой актуальной и нет никакой возможности откатить ее до предыдущей версии.
- Также, как и в Swagger, нету формы обратной связи. Нету возможности связаться с автором документации чтобы уточнить детали или указать на ошибку.

1.2.3.3. Ручной метод сопровождения API-документации

При ручном сопровождении документации необходимо выполнить ряд действий:

- 1) Авторизоваться в ECC Confluence
- 2) Перейти в раздел с общей технической документацией
- 3) Перейти в раздел с API-документацией проекта
- 4) Нажать на «Создать новую страницу»
- 5) Добавить необходимые компоненты на страницу
 - 5.1) Описание и название API метода
 - 5.2) Путь (URL) до API метода на сервере
 - 5.3) Описать каким образом будет осуществляться авторизация для получения доступа к API методу
 - 5.4) Описать перечень входящих параметров
 - Название параметра
 - Тип параметра
 - Указать, является данный параметр обязательным или нет
 - Указать какое значение по умолчанию установлено у параметра на сервере
 - 5.5) Примеры запросов и ответов от сервера

- 6) Выбрать «Сохранить и выйти» после чего передать готовую документацию в отдел клиентской разработки или в отдел тестирования для написания авто тестов.

Пример готовой API-документации представлено на рисунке 1.5.

Достоинства:

- Нет необходимости в приобретении дополнительного ПО
- Можно описывать документацию в любом удобном формате, однако стоит придерживаться единого формата всех документов

Недостатки:

- Время на создание и редактирование API-документации занимает очень много времени
- Необходимы дополнительные навыки по работе с Confluence

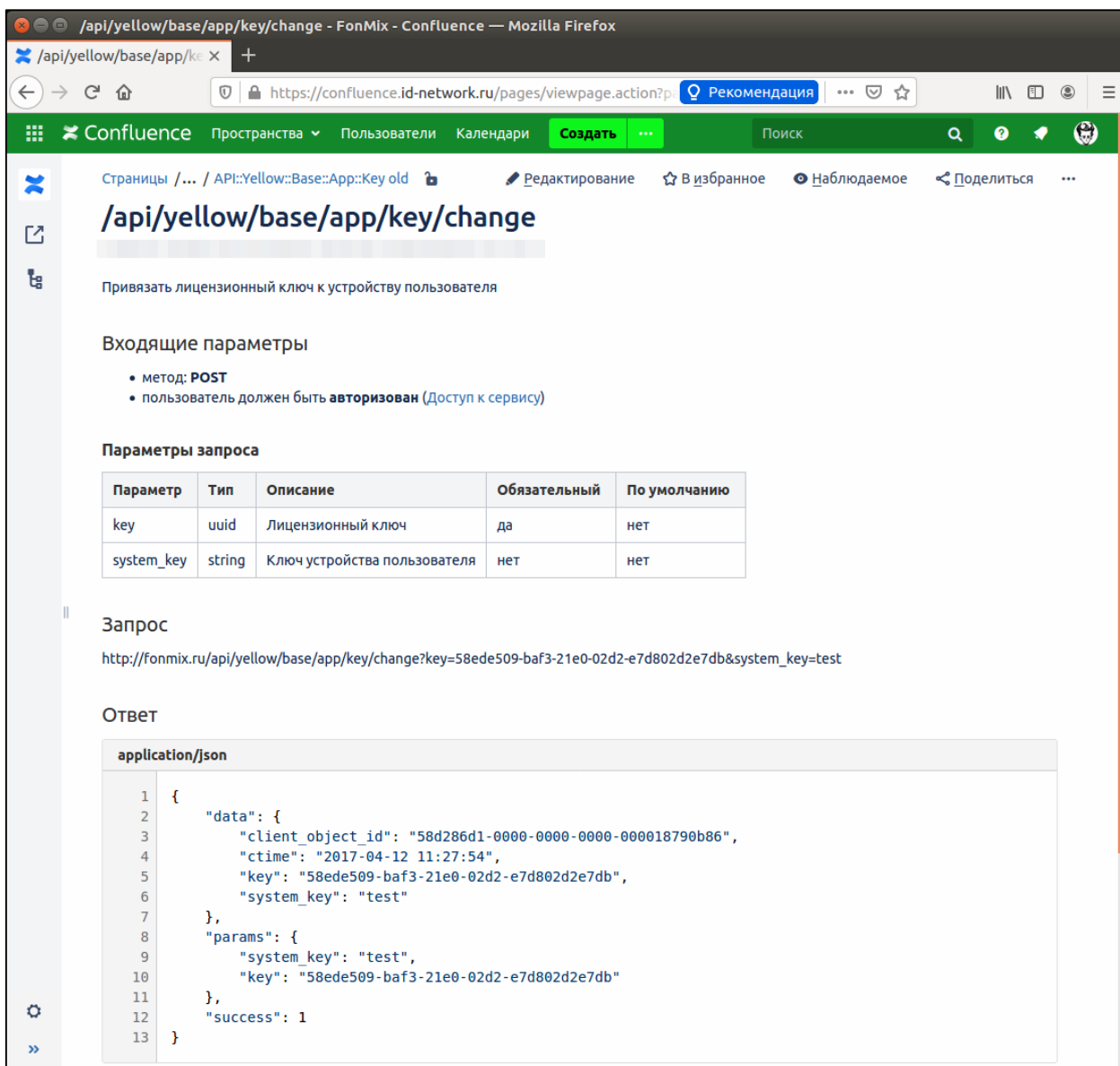


Рисунок 1.5 – Пример ручного создания API-документации

1.2.3.4. Postman

Postman представляет собой кроссплатформенное приложение с графическим интерфейсом для отправки запросов на сервер, получение ответа и его отображения.

Для установки на персональный компьютер необходимо открыть страницу в браузере <https://www.getpostman.com/>, выбрать из выпадающего

списка операционную систему (ОС), скачать и установить. Графический интерфейс Postman представлен на рисунке 1.6.

Приложение является условно бесплатным. Основной функционал доступен после авторизации на сайте.

Данный программный продукт активно используется на проекте и находится в перечне обязательных предустановленных программных продуктов компании.

Однако Postman не предоставляет возможности для документации API в единую справочную систему Confluence. Для реализации данного функционала было принято решение разработать отдельную утилиту.

Так как у Postman есть возможность экспорта и импорта всех необходимых данных можно разработать специальную утилиту, которая агрегировала бы данные и публиковала в Confluence.

Достоинства:

- Пользователю не нужно изучать дополнительные языки программирования чтобы редактировать и создавать документацию. Вся информация заполняется в интерактивных формах Postman после чего публикуется в Confluence.
- Отправка запроса на сервер и получение ответа. Демонстрация работоспособности API методов
- Экспорт и импорт коллекции для передачи сотрудникам компании
- Написание специальных скриптов для автоматического тестирования API методов.

Недостатки:

- Для реализации публикации API-документации в confluence требуется использование программного кода
- Возможны проблемы с реализацией возможных алгоритмов по сопровождению API-документации

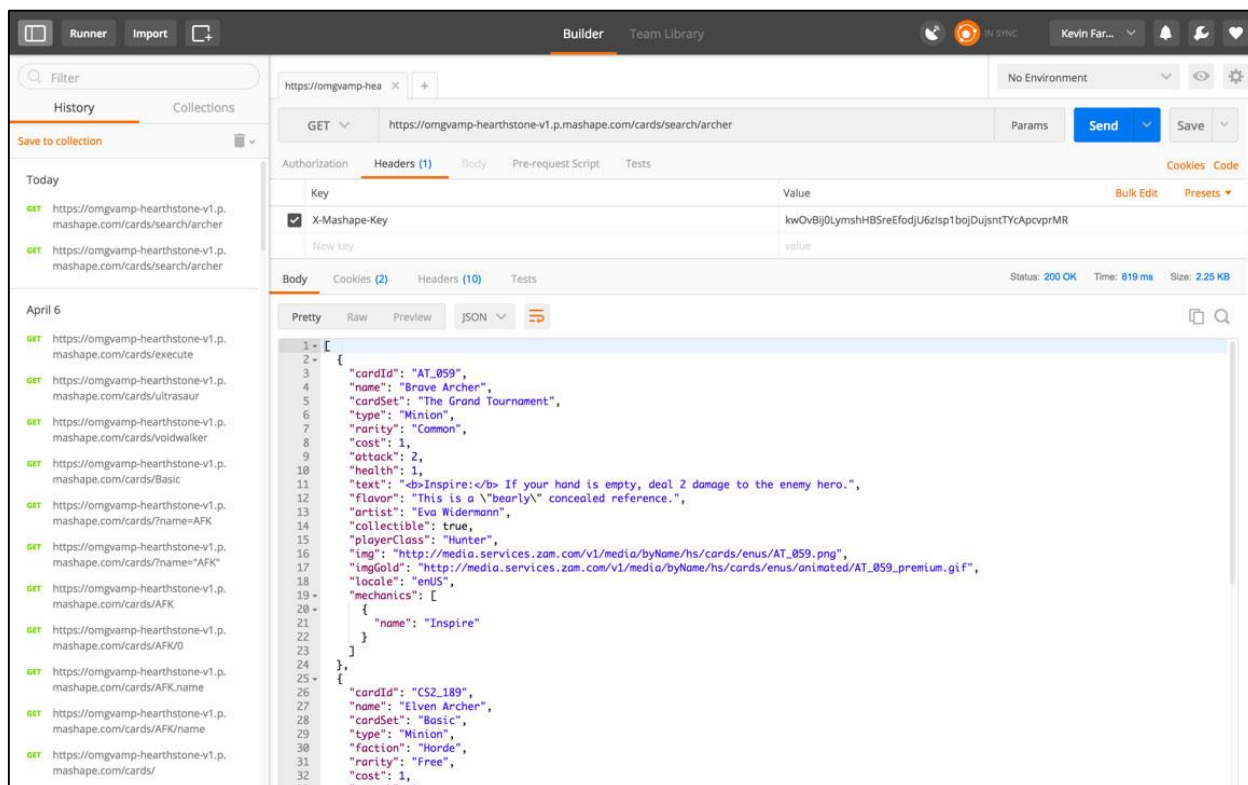


Рисунок 1.6 – Графический интерфейс Postman

2.3.4. Сравнение аналогов и прототипов

Соответствие рассматриваемых аналогов указанным критерием представлено в таблице 1.2.

В каждой ячейке стоит соответствие критерия и степень качества критерия. Степень качества и его целочисленный аналог представлен в таблице 1.1.

Таблица 1.1. – Шкала перевода степени качества критерия, в числовые

Отлично	100
Очень хорошо	80
Хорошо	60
Удовлетворительно	40

Плохо	20
Очень плохо	0

Таблица 1.2. – Качественные характеристики аналогов

	Swagger	Blueprint	Ручное сопровождение документации	Postman
Трудозатраты на изучение технологии	20	40	60	40
Потребность в дополнительном ПО	0	40	60	80
Настраиваемость системы	40	40	100	60
Время, затрачиваемое на сопровождение документации	20	40	0	80
Публикация документации в ЕСС	60	0	100	100
Σ	140	160	320	360

2.3.5. Вывод

По результатам сравнения аналогов видно, что утилита для Postman имеет наивысший балл и соответственно разработка утилиты для Postman обоснована.

2. РАСЧЕТНО-КОНСТРУКТОРСКАЯ ЧАСТЬ

2.1. Определение требований к системе

Автоматизированная система должна обеспечивать следующих функциональных требований:

- 1) Система должна работать на персональном компьютере под управлением операционной системы Linux и macOS
- 2) Система должна позволять заполнять всю необходимую информацию об API методах:
 - 2.1) Путь (URL) до API метода на сервере
 - 2.2) Название и подробное описание API метода
 - 2.3) Предусмотреть возможность заполнения данных об возможностях авторизации на сервере для взаимодействия с API методом.
 - 2.4) Обеспечить возможность заполнения информации о входящих параметрах API метода
 - Название параметра
 - Тип параметра
 - Указать, является данный параметр обязательным или нет
 - Указать какое значение по умолчанию установленное на сервере

- 2.5) Система должна позволять добавлять один или несколько примеров запросов на сервер и ответов от сервера
- 3) Должна быть удобная навигация по структуре документа или документов если их несколько
- 4) Необходимо предусмотреть поиск по документации, а также фильтрацию по: типам методов (GET, POST, PUT и т.д.), по версии документации
- 5) Необходимо обеспечить публикацию готовой документации в единой справочной системе компании

2.2. Разработка структуры автоматизированной системы

Структура взаимодействия системы выглядит следующим образом:

Пользователь, в данном случае бэкенд разработчик, в процессе разработки нового API метода проверяет его работоспособность через Postman.

По завершению кодирования функционала, заполняет необходимую информацию об API методе. Подробный список представлен в разделе «Требования к системе».

Затем добавляет примеры запросов и ответов от сервера.

В интерфейсе Postman нажимает на «Export» JSON структуры.

Выбирает место на компьютере куда происходит выгрузка структуры.

После этого запускает скрипт с указанием куда осуществлять публикацию API и путь до JSON структуры.

Авторизовывается в ECC Confluence для того чтобы проверить корректность созданной API-документации.

Передаёт в отдел клиентской разработки либо в отдел тестирования.

Структура и схема взаимодействия системы представлена на рисунке 2.1.

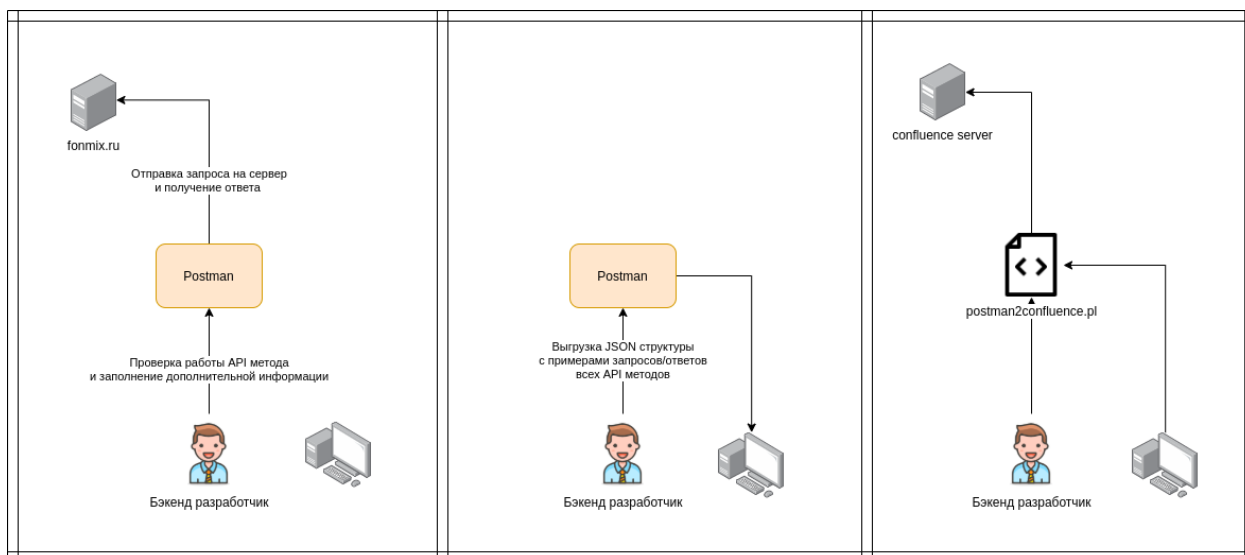


Рисунок 2.1 – Схема взаимодействия систем

2.3. Выбор языка программирования

Для реализации публикации документации в ECC Confluence лучше всего подходит язык программирования `perl`

Основные причины данного выбора:

- Компания ООО «ФорМакс» разрабатывает продукт Fonmix серверная часть которого написана с использованием языка программирования `perl`, тем самым бэкенд разработчики смогут поддерживать функционал публикации документации и добавлять его по мере необходимости
- ЯП `perl` скриптовый язык и на нем довольно легко пишутся программы
- На многих персональных компьютерах, под управлением операционной системы Linux и macOS, ЯП уже установлен с нужным набором библиотек для запуска скрипта из CLI

2.4. Разработка интерфейса взаимодействия пользователя с системой

2.3.1. Создание структуры документов в Postman

При запуске приложения Postman пользователь проходит авторизацию введя логин и пароль. При нажатии на «Выход» приложение прекращает работу и закрывается.

В случае не успешной авторизации пользователь возвращается к форме авторизации. В случае успешной авторизации пользователь переходит в основное меню. При нажатии кнопки «Выход» приложение прекращает работу и закрывается.

При повторных запусках приложения, форма авторизации отобразиться только если пользователь выйдет из системы путем нажатия «Выход».

Вид экранной формы авторизации, представлен на рисунке 2.2.

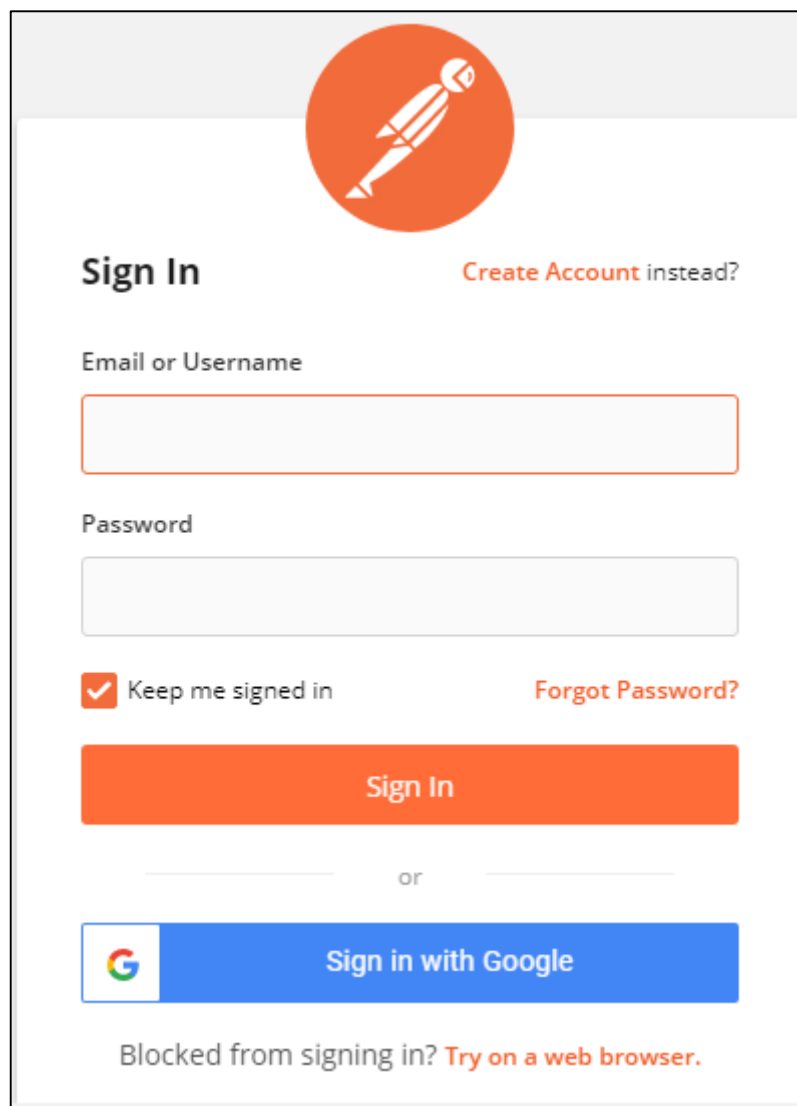
The image shows the 'Sign In' form in Postman. At the top, there is an orange circular icon with a white stylized figure. Below it, the text 'Sign In' is displayed in bold, with a link 'Create Account instead?' to its right. The form includes two input fields: 'Email or Username' and 'Password'. Below the password field, there is a checkbox labeled 'Keep me signed in' which is checked, and a link 'Forgot Password?'. A large orange 'Sign In' button is positioned below these elements. Underneath the button, the word 'or' is centered between two horizontal lines. Below this is a blue button with the Google logo and the text 'Sign in with Google'. At the bottom, there is a link 'Blocked from signing in? Try on a web browser.'

Рисунок 2.2 – Форма авторизации в Postman

Основными элементами Postman являются:

- 1) Collection (коллекции) – верхнеуровневые каталоги в которых находятся запросы или папки. Для удобства навигации коллекции называть в соответствии с разрабатываемыми приложениями.
- 2) Folder (папка) – используется для группировки запросов
- 3) Request (запрос) – основной объект для отправки запроса и получения ответа.

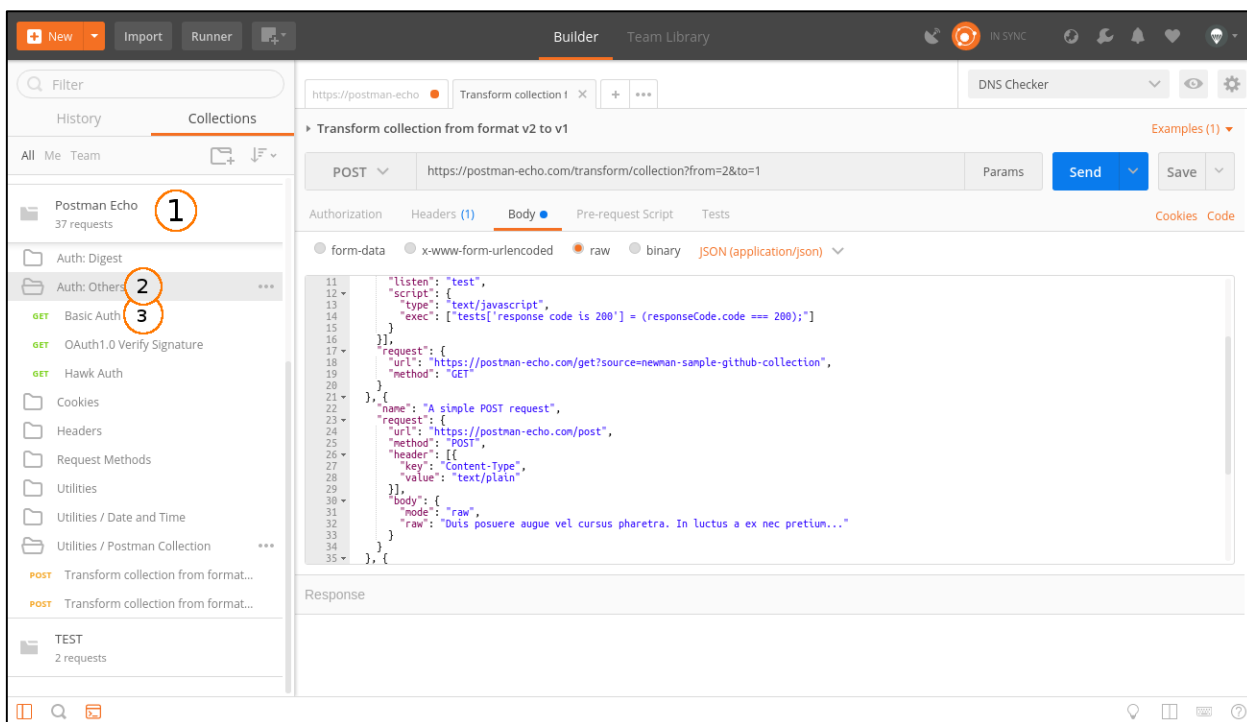


Рисунок 2.3 – Основные объекты в Postman

При первом запуске, пользователю необходимо создать коллекцию, а также необходимую структуру документов которая в дальнейшем будет опубликована в Confluence. Для создания коллекции и папок, используется контекстное меню, которое вызывается нажатием правой клавиши по панели навигации в Postman. Отличительной особенностью является то что не нужно создавать структуру в Confluence, достаточно расположить удобный формат в Postman. После запуска скрипта, структура будет полностью продублирована в Confluence.

Пример соответствия коллекций и структуры вложенностей в Confluence представлено на рисунке 2.4.

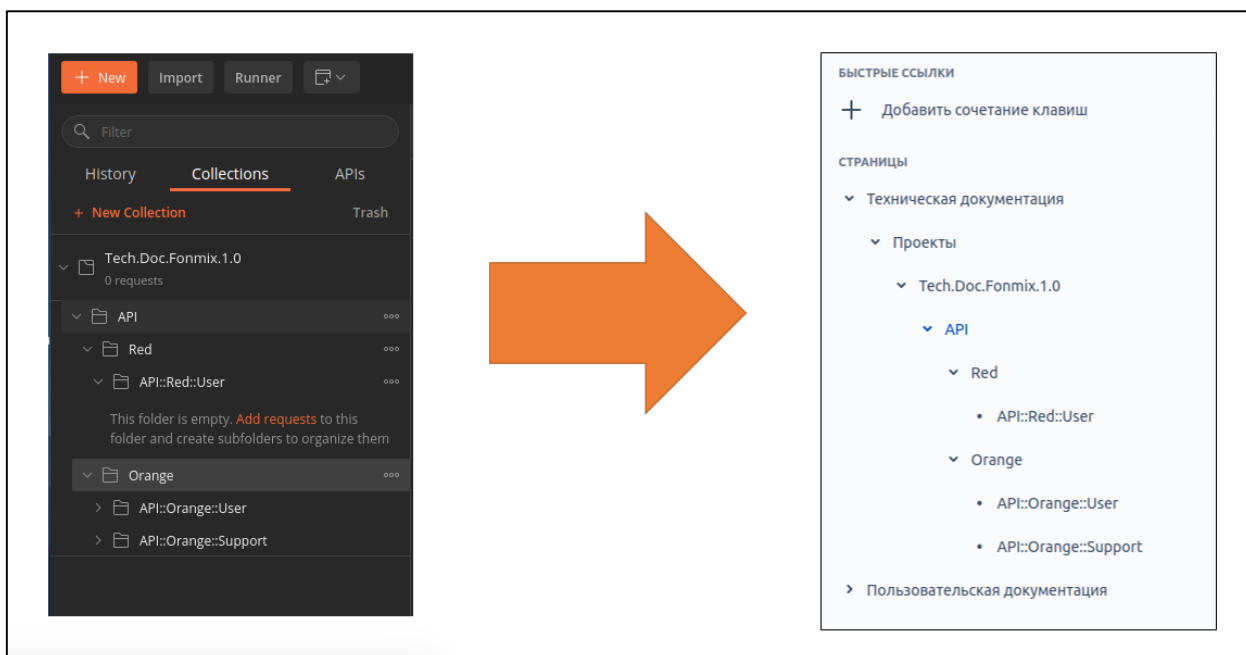


Рисунок 2.4 – Соответствие структуры Postman и Confluence

После создания структуры необходимо заполнить информацию об API методах.

При нажатии на кнопку «Add request» появляется форма в которой необходимо указать:

- 1) Название API метода
- 2) Заполняется описание API метода с помощью языка разметки Markdown. Опционально
- 3) Выбирается место в структуре куда будет размещен метод

Пример формы создания метода представлен на рисунке 2.5.

Затем заполняется вся необходимая информация о методе с примером запроса и ответа от сервера:

- 1) Выбирается какой метод будет использоваться в API (Method)
- 2) Заполняется путь до метода (URL)
- 3) В случае если метод поддерживает параметры из тела запроса, выбирается Body и form-data
- 4) Заполняется информация о входящих параметрах метода

- 5) Делается запрос на сервер чтобы убедиться в работоспособности API метода
- 6) Добавляется пример ответа от сервера
- 7) Сохранение изменений происходит при нажатии клавиши «Save»

Пример последовательности действий представлен на рисунке 2.6.

SAVE REQUEST

Requests in Postman are saved in collections (a group of requests).
[Learn more about creating collections](#)

Request name

api/login

Request description (Optional)

Авторизация пользователя в системе Fonmix

Descriptions support [Markdown](#)

Select a collection or folder to save to:

Search for a collection or folder

API::Red::User + Create Folder

Cancel Save to API::Red::User

Рисунок 2.5 – Создание API метода в Postman

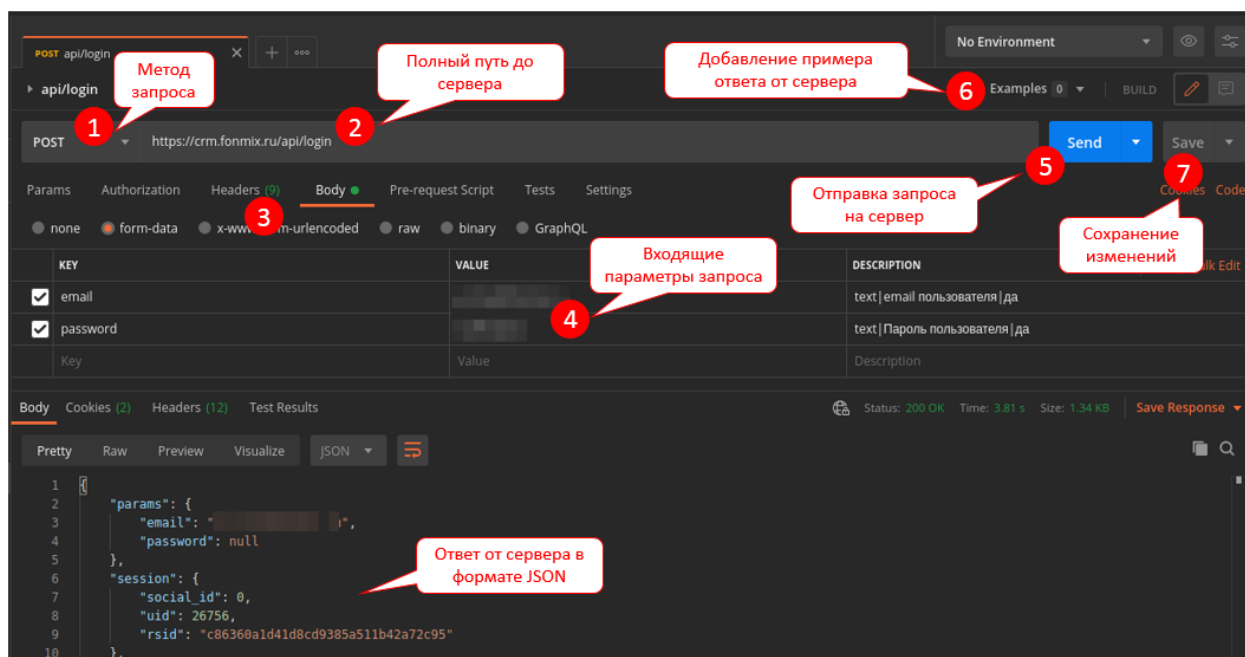


Рисунок 2.6 – Схема заполнения информации в Postman

Для заполнения данных о входящих параметрах необходимо указать:

- Название
- Пример
- Тип
- Описание
- Указать, является данный параметр обязательным или нет
- Указать какое значение по умолчанию установлено на сервере

Пример заполнения входящих параметров представлен на рисунке 2.7.

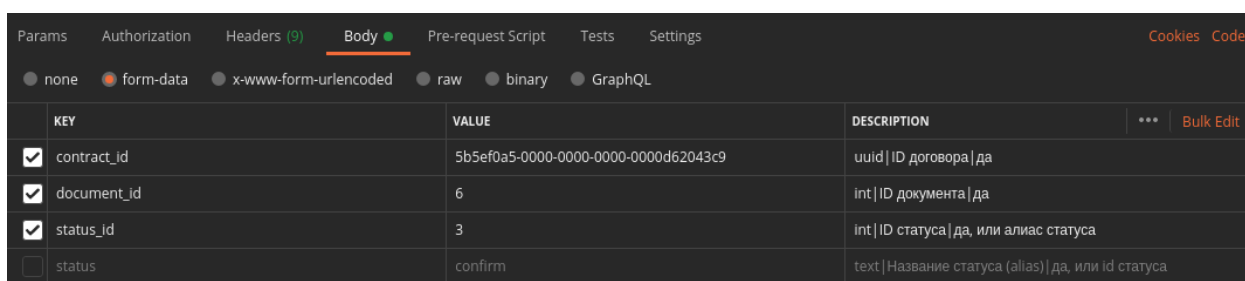


Рисунок 2.7 – Пример заполнения входящих параметров в Postman

Для создания примера ответа от сервера, необходимо после получения ответа от сервера выбрать «Examples» а затем «Add example». После чего выбрать «Save example». При необходимости можно изменить параметры, с которыми выполнялся запрос. Схема создания примера ответа от сервера представлен на рисунке 2.8.

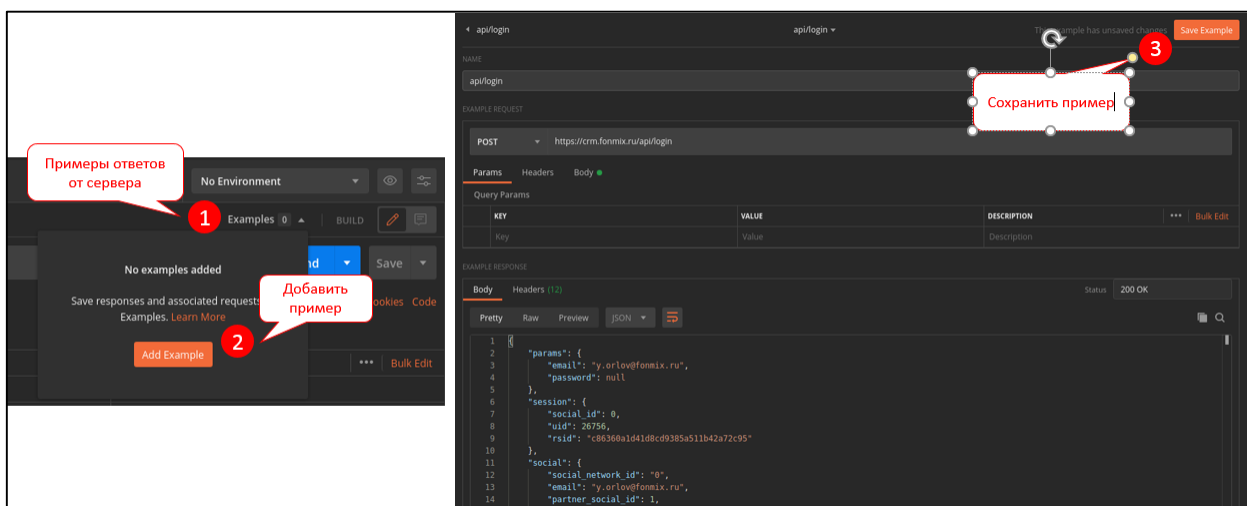


Рисунок 2.8 – Схема создания примера ответа от сервера в Postman

Для того чтобы выгрузить документацию на локальный компьютер необходимо нажать правой клавишей мыши на выгружаемую коллекцию тем самым вызвав контекстное меню и выбрать «Export».

В зависимости от версии программы, будет показана форма, в которой нужно указать версию экспортируемой коллекции и путь куда ее сохранить. Необходимо указать «Collection v2» так как там присутствует рекурсивная структура данных в отличие от первой версии.

Пример формы выгрузки коллекции представлен на рисунке 2.9.

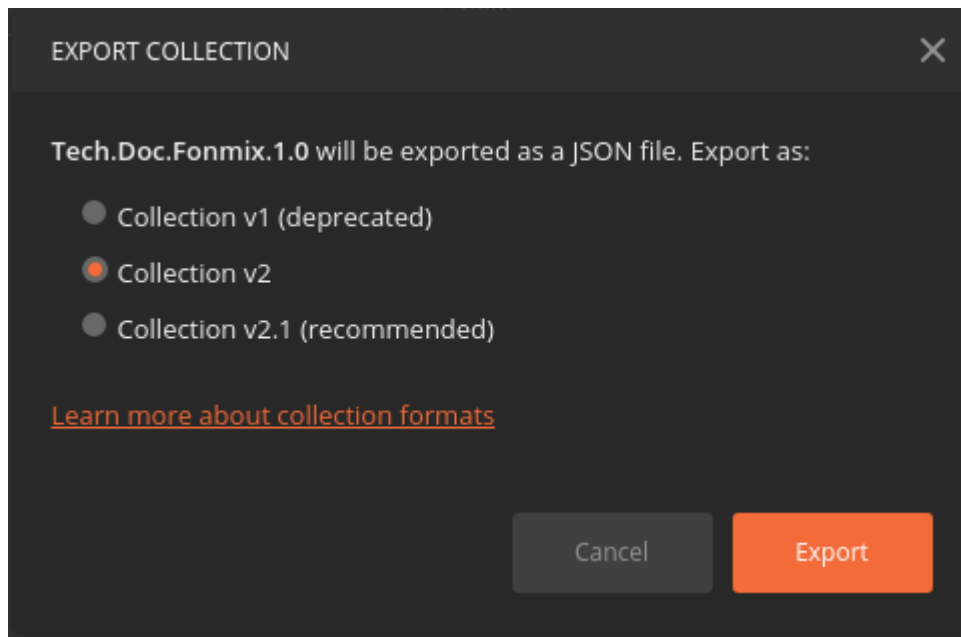


Рисунок 2.8 – Схема создания примера ответа от сервера в Postman

2.3.2. Публикация документации с помощью CLI

После того как структура документов из Postman выгружена на локальный компьютер можно приступить к ее публикации в Confluence.

Для этого необходимо скачать скрипт, который публикует документацию на локальный компьютер. Для этого можно воспользоваться утилитой git или скачать zip архив.

Основными параметрами скрипта являются:

- host – путь до сервера confluence где располагается документация
- login – логин пользователя из-под которого будет производиться публикация документов
- password – пароль пользователя
- file – путь до файла со структурой документов выгруженная из Postman
- space – название пространства имен в Confluence
- start_page – не обязательный параметр начальной страницы

Пример запуска скрипта с минимальным перечнем параметров, представлен на рисунке 2.9.

```
avis@avis[21:19:43]:~/projects/postman/lib/External/script$ perl ./postman2confluence.pl \  
> -host='https://avis20.atlassian.net/wiki/' \  
> -login= \  
> -password= \  
> -file=/home/avis/postman_doc/Tech.Doc.Fonmix.1.0.postman_collection.json \  
> -space=NOVA \  
> -start_page='API::Red::User' \  
> -v=2  
Use of uninitialized value $ENV{"CONFIG_DIR"} in concatenation (.) or string at ./postman2confluence.pl line 45.  
$VAR1 = 'Confluence';  
$VAR2 = {  
    'host' => 'https://avis20.atlassian.net/wiki/',  
    'space' => 'NOVA'  
};  
$VAR3 = 'Sign_in';  
$VAR4 = {  
    'password' => ' ',  
    'login' => ' '  
};  
$VAR5 = 'file';  
$VAR6 = '/home/avis/postman_doc/Tech.Doc.Fonmix.1.0.postman_collection.json';
```

Рисунок 2.9 – Пример запуска скрипта публикации документации в Confluence

По результату работы скрипта необходимо проверить качество созданной документации. Для этого необходимо авторизоваться в системе Confluence и перейти по навигации на опубликованную страницу. Пример готовой документации представлен на рисунке 2.10.

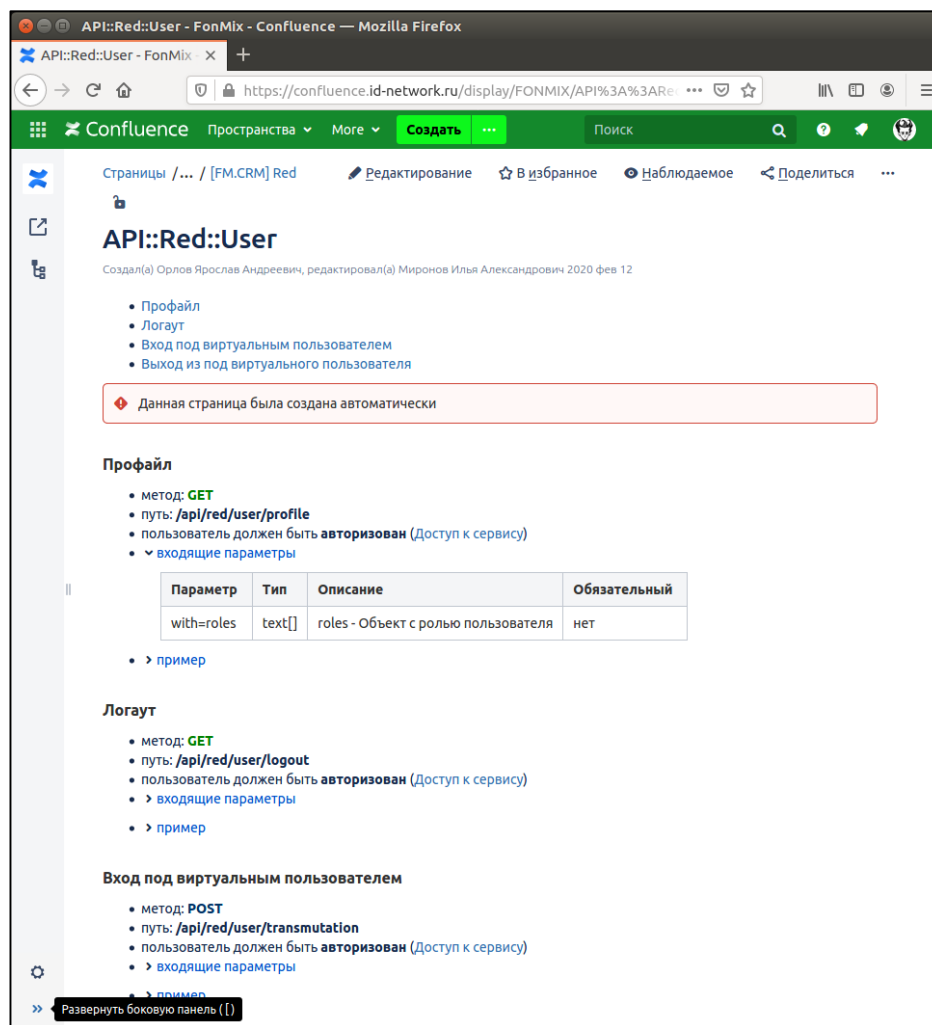


Рисунок 2.10 – Пример опубликованной документации в Confluence

Для удобства использования каждый из перечисленных параметров можно вынести в отдельный файл. Для этого необходимо указать в переменные окружения CONFIG_DIR путь до конфигурационного файла с названием postman.yaml.

Формат файла использовался YAML так как он более всего подходит под эту задачу. Формат файла должен быть соответствовать входящим параметрам при вызове скрипта. Пример конфигурационного файла представлен на рисунке 2.11.

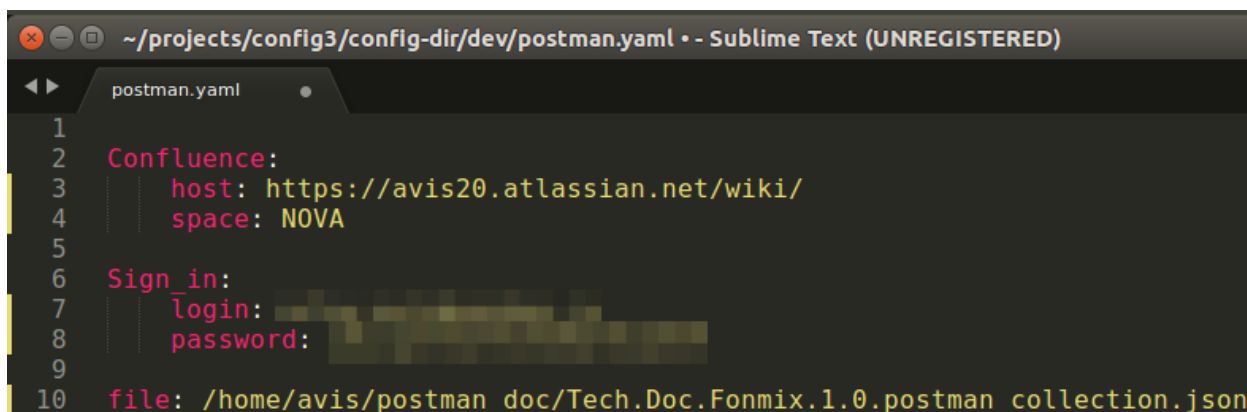


Рисунок 2.11 – Пример конфигурационного файла для публикации документации в Confluence

2.5. Разработка алгоритмов программных модулей

Для реализации публикации документации в ECC Confluence необходимо выполнить ряд условий:

- 1) Авторизоваться в системе Confluence
- 2) Найти корневую страницу с помощью API методов предоставляемых Confluence
- 3) Опубликовать новую версию документации

Соответственно, авторизации в системе происходит путем передачи заголовка Basic, где значением является логин и пароль пользователя зашифрованный с помощью алгоритма base64.

Корневая страница в пространстве Confluence является начальной страницей, от которой в дальнейшем будет публиковаться остальная часть. Для ее нахождения используется API метод Confluence, rest/api/content/ с передачей параметра title равный названию проекта.

Для публикации API-документации используется API-метод rest/api/content/add.

Алгоритм публикации документации представлен на рисунке 2.12.

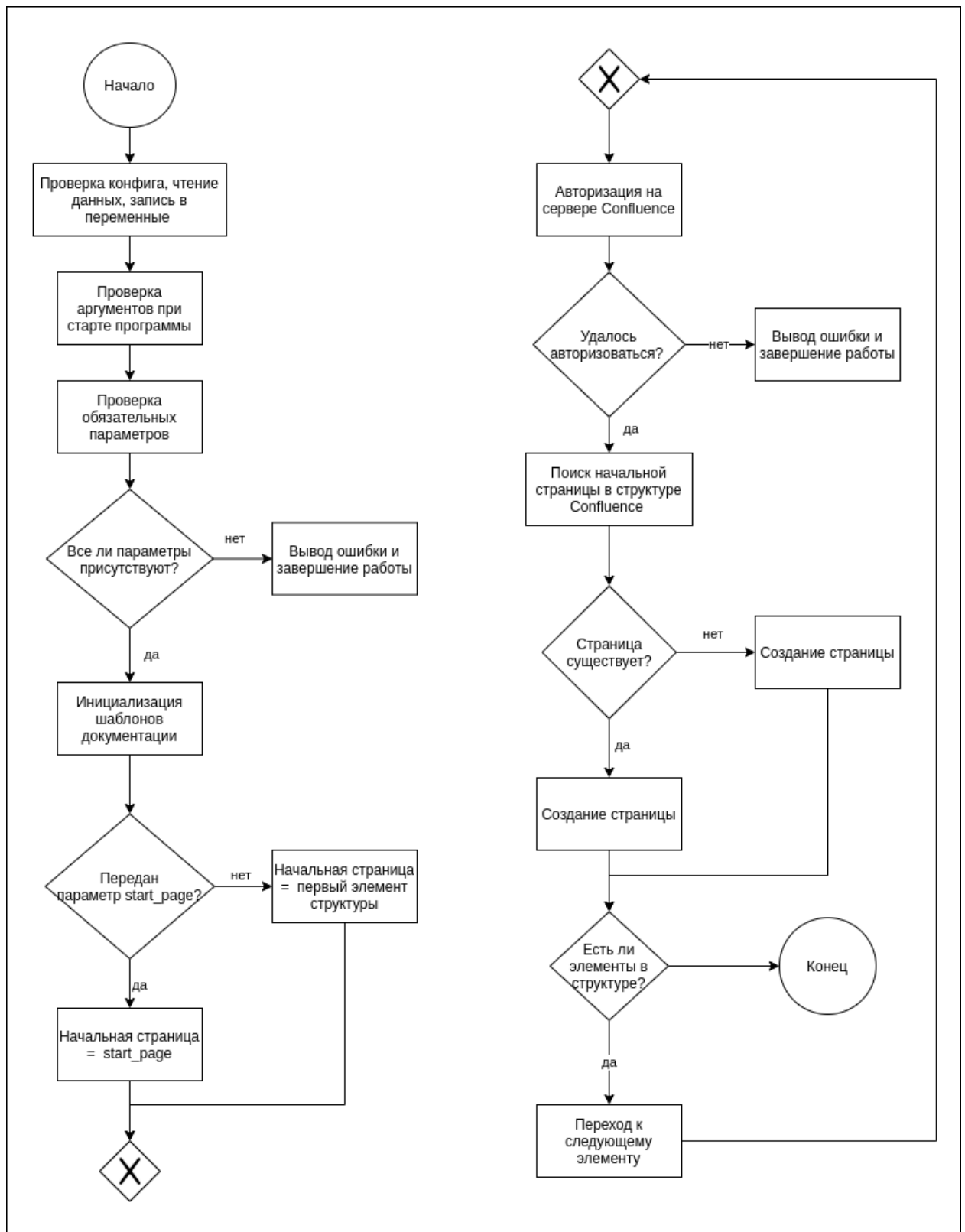


Рисунок 2.12 – Алгоритм публикации документации в Confluence

2.6. Разработка плана проведения тестирования

Тестирование программного обеспечения – процесс анализа программного продукта и сопутствующей документации с целью выявления недостатков в работе и повышения его качества.

Обычно для проверки работоспособности разработанного ПО проводят три вида тестирования: функциональное, интеграционное и нагрузочное.

Функциональное тестирование (Functional testing) – вид тестирования, направленный на проверку корректности работы функциональности приложения (корректность реализации функциональных требований).

Интеграционное тестирование (Integration testing) направлено на проверку взаимодействия между несколькими частями приложения (каждая из которых, в свою очередь, проверена отдельно на стадии модульного тестирования).

Нагрузочное тестирование (Load testing, capacity testing) – исследование способности приложения сохранять заданные показатели качества при нагрузке в допустимых пределах и некотором превышении этих пределов.

Таким образом для проведения тестирования системы необходимо выполнить все три вида тестирования.

1) Функциональное тестирование

- 1.1) Проверить что система работает на персональном компьютере под управлением операционной системы Linux либо macOS
- 1.2) Проверить заполнение форм в Postman
- 1.3) Проверить выгрузку структуры документации на персональный компьютер пользователя

2) Интеграционное тестирование

- 2.1) Проверить взаимодействия скрипта публикации документации со структурой, выгружаемой из Postman

3. ЭКСПЕРИМЕНТАЛЬНАЯ ЧАСТЬ

3.1. Реализация разработанных алгоритмов

При запуске скрипта, алгоритм публикации API-документации состоит из нескольких этапов.

В первую очередь проверяется наличие конфигурационного файла. Если он присутствует на локальной машине, то считывается в переменную с помощью стандартной библиотеки работы с файлами и в дальнейшем используются параметры из конфигурационного файла.

Затем проверяются входящие аргументы при запуске скрипта и записываются в словарь, который в дальнейшем используется в скрипте.

Если при запуске обнаружится параметр, который присутствовал в конфигурационном файле, то он окажется приоритетнее и будет использоваться значение из аргументов.

После этого проверяется наличие обязательных параметров, без которых дальнейшая работа будет невозможной:

- host – путь до сервера confluence где располагается документация
- login – логин пользователя из-под которого будет производиться публикация документов
- password – пароль пользователя
- file – путь до файла со структурой документов выгруженная из Postman
- space – название пространства имен в Confluence

В дальнейшем происходит инициализация и считывание шаблонов в которые затем происходит вставка параметров из структуры Postman.

Для работы с шаблонами документов, используется сторонний модуль Template Toolkit. Это достаточно гибкий шаблонизатор который предлагает довольно широкий набор функций, которые активно используются при генерации документации. Например, нет никакой необходимости держать шаблоны документов в коде программы, гораздо лучше вынести их в

отдельные файлы и подключать по мере необходимости тем самым облегчая чтение исходного кода.

Если при запуске был передан параметр `start_page`, то в структуре документации происходит поиск подходящей страницы. В противном случае берется первый элемент структуры. Это необходимо для того чтобы при запуске скрипта, указать страницу, с которой нужно начать публиковать.

Дальнейшая работа заключается в нахождении страницы в ЕСС Confluence. Для этого:

- 1) Происходит авторизация на сервере Confluence по переданным логином и паролем при запуске. Если не удалось авторизоваться, выводится сообщение об ошибке и программа завершается.
- 2) Происходит поиск стартовой страницы по ее названию. Если страница не найдена, то она создается.
- 3) Опубликовать новую версию документации
- 4) Перейти к следующему элементу списка документации.

Дальнейшая обработка происходит аналогичным образом пока не опубликуется последний элемент в структуре Postman, то есть реализована в виде цикла по структуре, выгруженной из Postman.

Ключевой особенностью является то что структура из Postman представляет собой древовидную структуру, листьями которой являются запросы (Request) с необходимым набором данных. Благодаря формату JSON этого довольно легко добиться и даже без наличия необходимых инструментов можно увидеть структуру и понять ее ключевые особенности.

Например, структура API методов для авторизации пользователя представлена на рисунке 3.1.

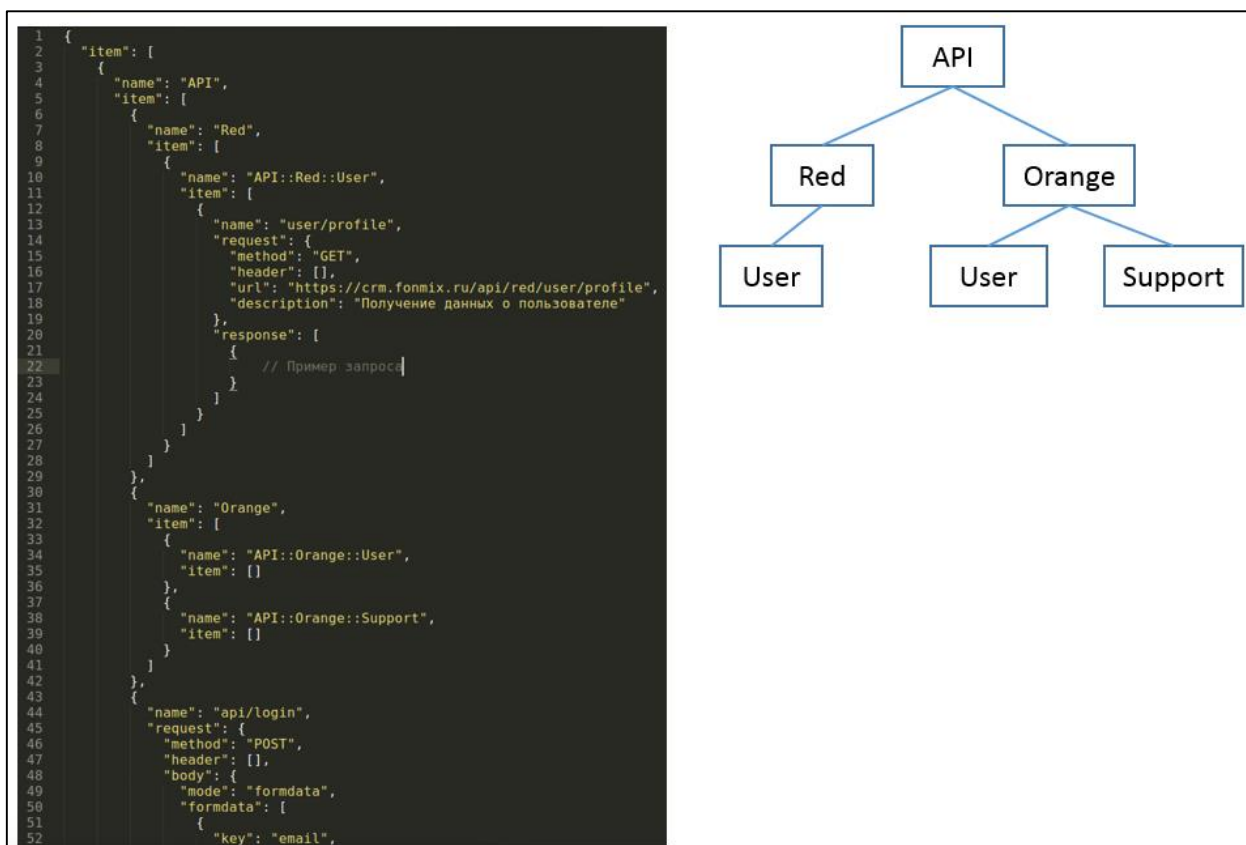


Рисунок 3.1 – Структура документации, выгруженная из Postman

3.2. Тестирование и отладка системы

Для тестирования автоматизированной системы, согласно плану тестирования, необходимо:

3.2.1. Тестирование алгоритма установки программного обеспечения

Алгоритм:

- 1) Установить на компьютер программу Postman под управлением операционной системы Linux и macOS
- 2) Скачать, или клонировать с помощью утилиты git, скрипт публикации документации
- 3) Подготовить конфигурационный файл с указанием необходимых параметров

Ожидаемый результат:

Программное обеспечение установлено на операционные системы Linux и macOS.

Фактический результат:

Программное обеспечение установлено и соответствует ожидаемому результату.

3.2.2. Тестирование алгоритма создания структуры и заполнения форм API методов в интерфейсе Postman

Алгоритм:

Алгоритм по созданию структуры и заполнению форм в интерфейсе Postman описан в пункте «2.3.2 Создание структуры документов в Postman»

Ожидаемый результат:

Структура документации создается и формы для заполнения данными работают.

Фактический результат:

Фактический результат, полученный после выполнения алгоритма, соответствует ожидаемому результату.

3.2.3. Тестирование алгоритма выгрузки структуры документации на локальный компьютер

Алгоритм:

Алгоритм по выгрузке структуры документации описан в пункте «2.3.2 Создание структуры документов в Postman», раздел «Выгрузка структуры документации»

Ожидаемый результат:

Структура документации выгружается на локальный компьютер под управлением операционной системы Linux и macOS

Фактический результат:

Фактический результат, полученный после выполнения алгоритма, соответствует ожидаемому результату.

3.2.4. Тестирование алгоритма публикации документации в Confluence

Алгоритм:

Алгоритм по публикации документации описан в пункте «2.3.3 Публикация документации с помощью CLI»

Ожидаемый результат:

Документация публикуется в ЕСС Confluence и доступна авторизованным пользователям.

Фактический результат:

Фактический результат, полученный после выполнения алгоритма, соответствует ожидаемому результату.

ЗАКЛЮЧЕНИЕ

В рамках выпускной квалификационной работы разработана система по сопровождению API-документации.

В ходе выполнения работы проведен анализ аналогов и прототипов. Составлен перечень критериев качества, по которым происходило сравнение систем. Выбран и обоснована система Postman с реализацией дополнительного ПО в виде скрипта публикации API-документации.

Составлен список функциональных требований, которые полностью реализованы в рамках ВКР.

Разработан и протестирован интерфейс взаимодействия пользователя с системой. Выбран и обоснован язык программирования perl. Разработан и протестирован скрипт публикации документации в единую справочную систему компании.

Таким образом, работа выполнена в полном объеме. Описанная система удовлетворяет всем требованиям технического задания и пригодна для функционирования в ООО «ФорМакс».

СПИСОК ЛИТЕРАТУРЫ

1. The OpenAPI Specification – [Электронный ресурс]:
<https://github.com/OAI/OpenAPI-Specification> (Дата обращения: 24.12.2020)
2. API Blueprint – [Электронный ресурс]:
<https://apiblueprint.org/documentation> (Дата обращения: 24.12.2020)
3. API Documentation with Postman – [Электронный ресурс]:
<https://learning.postman.com/docs/publishing-your-api/documenting-your-api/> (Дата обращения: 24.12.2020)
4. RESTful API Modeling Language (RAML) – [Электронный ресурс]:
<https://raml.org/> (Дата обращения: 24.12.2020)
5. Git How To – [Электронный ресурс]: <https://githowto.com/> (Дата обращения: 24.12.2020)
6. Confluence Server REST API – [Электронный ресурс]:
<https://developer.atlassian.com/server/confluence/confluence-server-rest-api/> (Дата обращения: 24.12.2020)
7. Template Toolkit – [Электронный ресурс]: <http://www.template-toolkit.org/> (Дата обращения: 24.12.2020)
8. Bootstrap Documentation – [Электронный ресурс]:
<https://getbootstrap.com/docs/3.3/> (Дата обращения: 28.09.2020)
9. PostgreSQL Database Documentation – [Электронный ресурс]:
<https://www.postgresql.org/docs/> (Дата обращения: 28.09.2020)
10. Скотт Б., Нейл Т. Проектирование веб-интерфейсов. – СПб.: Символ-Плюс, 2010. – 352 с.

ПРИЛОЖЕНИЕ А: Листинг программы публикации документации

```
#!/usr/bin/perl
use utf8;
use strict;
use warnings;
use lib::abs      qw| ../../lib |;
use Encode        qw| encode decode |;
use JSON::XS      qw| decode_json encode_json |;
use Data::Dumper;
use URI::Escape;
use HTTP::Request;
use LWP::UserAgent;
use MIME::Base64;
use Template;
use External::Config2;
our $VERSION = v1.5.7;
use constant {
    SEND_TO_SERVER_OFF => 1,
    API_CONTENT => 'rest/api/content/',
};
# 0 - nothing, 1 - show config, 2 - show template, 3 - show requests
my $DEBUG = 0;
# Загружаем конфиг
my %config_sample = (
    'Confluence' => {
        'host' => "",
        'space' => "",
    },
},
```

```

'Sign_in' => {
  'login' => ",
  'password' => ",
},
'file' => ",
);

```

```

my $path = ( ( $ENV{CONFIG_DIR}.'/' || '/spool/' ).'postman.yaml');
my %config = -e $path ? %{ External::Config2->new( 'project' => 'postman' ) }
: %config_sample;

```

```

my %args = map { /^-
(file|login|password|space|host|help|template|v|start_page|only_page)=(.*)/ ? ($1
=> $2) : () } @ARGV;

```

```

$DEBUG = $args{v} if $args{v};
print_help() if ( $args{'help'} );

```

Если есть параметры, то изменяем/дополняем конфиг

```

my $is_found = 0;
while ( my ($key, $value) = each %args ) {
  if ( defined $config{$key} ){
    $config{$key} = $value;
    $is_found = 1;
    next;
  } else {
    for ( grep ref $_ eq 'HASH', %config ){
      if ( defined $_->{$key} ){
        $_->{$key} = $value;

```

```

        $is_found = 1;
        next;
    }
}
}
$config{$key} = $value unless $is_found;
}

# Проверяем обязательные параметры
my $bad_param = 0;
foreach my $key ( keys %config_sample ){
    if ( ref $config{$key} eq 'HASH' ){
        for ( my ($k2, $v2) = each %{ $config{$key} } ){
            $bad_param = $k2 unless $v2;
        }
    } else {
        $bad_param = $key unless $config{$key};
    }
}

die "param: $bad_param is required" if $bad_param;
die "$config{file} does not exist" unless -f $config{file};

warn Dumper %config if $DEBUG > 0;

# Подготавливаем шаблон
my $tt = Template->new({
    INCLUDE_PATH => 'template/',
    ABSOLUTE    => 1,

```

```
});
```

```
my $api_page_template = {  
  'type'    => 'page',  
  'title'   => "",  
  'version' => {  
    'number' => 1,  
  },  
  'space'   => {  
    'key'    => "",  
  },  
  'body' => {  
    'storage' => {  
      'representation' => 'storage',  
    }  
  },  
};
```

```
$api_page_template->{'space'}->{'key'} = $config{'Confluence'}->{'space'};  
$config{'path_to_template'} = lib::abs::path('/template/').'/';  
$config{'main_template'} = $config{'path_to_template'}.($config{'template'} ||  
'fonmix').'.tt';
```

```
open my $fh, $config{file} or die "Can't open file $config{file}: $!";
```

```
my $json;  
while ( <$fh> ) {  
  chomp;  
  s/\r$//g;
```



```

$json .= $_;
}

close $fh;

my $hash;
eval { $hash = decode_json( encode('utf8', $json ) ) };
die "$@" if $@;

# Находим или создаем корневую страницу, от которой будут
наследоваться остальные
my $root_page_name = $hash->{info}->{name};
if ( $args{start_page} ){
    ( $hash->{item}, my $prev_page_name ) = find_start_page( $hash->{item},
$args{start_page} );
    die "Can't find start page: $args{start_page}" unless $hash->{item} ||
$prev_page_name;
    $root_page_name = $prev_page_name;
} else {
    die "OK, don't update\n" unless ask("Are you shure want to update all
workspace?");
}

my $root_page_id = find_to_confluence( $root_page_name )->{results}->[0]-
>{id};

unless ( $root_page_id ){
    $api_page_template->{'title'} = $root_page_name;
    my $root_page_in_space_id = find_to_confluence()->{results}->[0]->{id};

```

```

    push @{ $api_page_template->'ancestors' }, { 'id' =>
$root_page_in_space_id };
    $root_page_id = add_to_confluence( $api_page_template )->{id} unless
$config{start_page};
}

```

```

die decode('utf8', "Не удалось найти или создать корневую страницу" )
unless defined $root_page_id;

```

```

# Создаем вложенности
print_item( $hash->{item}, $root_page_id );

```

```

exit;

```

```

sub find_start_page {
    my ( $parent, $page_name, $prev ) = @_ ;

    our @found;

    foreach my $item ( @{ $parent || [] } ){
        $item->{name} eq $page_name ? @found = ( [ $item ], $prev ) :
find_start_page( $item->{item}, $page_name, $item->{name} );
    }

    return @found;
}

```

```

sub print_item {
    my ( $parent, $parent_id ) = @_ ;

```

```

my $page_id = 0;
$api_page_template->{'ancestors'} = [];
$api_page_template->{'body'}->{'storage'}->{'value'} = {};

foreach my $item ( @{ $parent || [] } ){
    return if $item->{request};
    my ( $output, $page_in_confluence ) = "";

    $tt->process( $config{'path_to_template'}.'/block/toc.tt', $item, \$output)
or die $tt->error();

    # Исключаем дубли и добавляем шаблон запросу
    my %hash;
    foreach my $page ( grep $_->{request}, @{ $item->{item} || [] } ){
        my $url = ( ref $page->{request}->{url} eq 'HASH' ? $page-
>{request}->{url}->{raw} : $page->{request}->{url} );
        next if $hash{$url}++;
        $tt->process($config{'main_template'}, processing( $page ), \$output)
or die $tt->error();
    }

    warn decode('utf8', $output) if $DEBUG > 1;

    $api_page_template->{'title'} = $item->{name};
    push @{ $api_page_template->{'ancestors'} }, { 'id' => $parent_id };
    $api_page_template->{'body'}->{'storage'}->{'value'} = decode('utf8',
$output);
    $page_in_confluence = find_to_confluence( $item->{name} )->{results}-
>[0];

```

```

if ( exists $page_in_confluence->'id' ){
    $api_page_template->'id' = $page_in_confluence->'id';

    # Инкремент версии страницы
    $api_page_template->'version'->'number' =
        ( defined $page_in_confluence->'version'->'number' ?
          ++$page_in_confluence->'version'->'number' : 1 );

    # Не обновлять страницу если в описании DONT CHANGE
    ORIGINAL!
    $page_id = $item->{description} =~ m/^DONT CHANGE
    ORIGINAL!/ ? $page_in_confluence->'id' :
        update_to_confluence( $page_in_confluence->'id',
    $api_page_template )->'id';
    } else {
        $page_id = add_to_confluence( $api_page_template )->'id';
    }

    warn "\t$item->{name} --- my $page_id, $parent_id\n" if $DEBUG > 1;

    print_item( $item->{item}, $page_id ) unless $args{only_page};
}
}

sub processing {
    my ( $page ) = @_ ;

    my $request = $page->{request};

```

```

# Обработка заголовка запроса
if ( $request->{description} ) {
    if ( $request->{description} =~ s/^# (.*)\n// ){
        $request->{desc}->{header} = $1;
        $request->{more} = [ map $_, split /\n/, $request->{description} ];
    } else {
        $request->{desc}->{header} = $request->{description};
    }
}

my $url_hash = ref $request->{url} eq 'HASH' ? $request->{url} : {};
if ( $request->{method} eq 'POST' || !$url_hash->{path} ){

    if ( $request->{url} ){
        my $url = exists $url_hash->{raw} ? $url_hash->{raw} : $request-
>{url};
        ( undef, $request->{path} ) = $url =~
m/^(?:\{\{(\w+)?host\}\}|http[s]?://(?:[da-z\.-]+))(.*)\?/?;
    }

    foreach my $raw ( @{ $request->{body}->{formdata} || [] } ){
        $raw = undef if $raw->{disabled};
        $raw->{name} = $raw->{key};
        $raw->{data} = [ split /\|/, $raw->{description} ] if $raw-
>{description};
        $raw->{data}->[1] = [ split /\n/, $raw->{data}->[1] ] if $raw->{data}-
>[1];
    };
}

```

```

} else {
    $request->{path} = ' '. join ' ', @ { $request->{url}->{path} || [] };

    foreach my $raw ( @ { $request->{url}->{query} || [] } ){
        $raw->{name} = $raw->{key};
        $raw->{data} = [ split /\|/, $raw->{description} ] if $raw->{description};
        $raw->{data}->[1] = [ split /\n/, $raw->{data}->[1] ] if $raw->{data}->[1];
    };
}

$request->{host} = $config{host_replace};

my $j = JSON::XS->new->pretty(1);
if ( @ { $page->{response} || [] } && $page->{response}->[0]->{body} ){
    $request->{response_params} = $page->{response}->[0]->{originalRequest};
    $request->{response} = $j->decode( $page->{response}->[0]->{body} );
    $request->{response} = $j->encode( $page->{request}->{response} );
}

return $page;
}

sub find_to_confluence {
    my $page_name = shift;
    $page_name = uri_escape($page_name);

```

```

    request('GET', $config{Confluence}-
>{host}.API_CONTENT.'?expand=version&spaceKey='.$config{Confluence}-
>{space}.'&title='.$page_name );
}

```

```

sub add_to_confluence {
    my $template = shift;
    request('POST', $config{Confluence}->{host}.API_CONTENT,
encode_json( $template ) );
}

```

```

sub update_to_confluence {
    my ( $id, $template ) = @_;
    request('PUT', $config{Confluence}->{host}.API_CONTENT.$id,
encode_json( $template ) );
}

```

```

sub request {
    my ( $method, $uri, $json ) = @_;

    my $req = HTTP::Request->new( $method, $uri );
    $req->header(
        'accept'      => 'application/json',
        'content-type' => 'application/json',
        'authorization' => 'Basic ' . encode_base64($config{Sign_in}->{login} .
        ':' . $config{Sign_in}->{password}),
    );
    $req->content( $json ) if defined $json;
    warn decode('utf8', Dumper $req ) if $DEBUG > 2;
}

```

```

my $ua = LWP::UserAgent->new();
my $response = $ua->request( $req );

my $res_hash;
eval { $res_hash = decode_json($response->decoded_content) };

if ( $@ || defined $res_hash->{statusCode} ){
    warn $@;
    warn $response->decoded_content;
}

return $res_hash;
}

sub ask {
    my ( $question ) = @_ ;
    return 1 if ( $args{force} == 1 );
    warn "$question (y/n): ";
    while( <STDIN> ) {
        chomp;
        last if /^y$/;
        return 0;
    }
    return 1;
};

```