# Подходы к построению АПИ

## Версионность

api/v1/…

## Разделение ролей

api/v1/admin/… и api/v1/user

## Формат обмена данными

JSON

## Путь к коллекции

api/v1/user/users

К ним 2 метода можно применять: GET для получения списка и POST для создания нового элемента

В запросах на получение списков мы можем использовать множество дополнительных GET параметров, применяемых для постраничного вывода, сортировки, фильтрации, поиска etc, но они должны быть опциональными, т.е. эти параметры не должны передаваться как часть пути!

## Элементы коллекции

Для обращения к конкретному элементу коллекции мы используем в маршруте его уникальный идентификатор /users/25 . Это и есть уникальный путь к нему. Для работы с объектом применимы методы GET (получение объекта), PUT/PATCH (изменение) и DELETE (удаление).

## Уникальные объекты

Во множестве сервисов существуют уникальные для текущего пользователя объекты, например профиль текущего пользователя /profile , или персональные настройки /settings . Разумеется, с одной стороны, это элементы одной из коллекций, но они являются отправной точкой в использовании нашего Web API клиентским приложением, и к тому же позволяют намного более широкий спектр операций над данными. При этом коллекция, хранящая пользовательские настройки может быть вообще недоступна из соображений безопасности и конфиденциальности данных.

## Свойства объектов и коллекций

Для того, чтобы добраться до любого из свойств объекта напрямую, достаточно добавить к пути до объекта имя свойства, например получить имя пользователя /users/25/name . К свойству применимы методы GET (получение значения) и PUT/PATCH (изменение значения). Метод DELETE не применим, т.к. свойство является структурной частью объекта, как формализованной единицы данных.

В предыдущей части мы говорили о том, что у коллекций, как и у объектов, могут быть собственные свойства. На моей памяти мне пригодилось только свойство count, но ваше приложение может быть более сложным и специфичным. Пути к свойствам коллекций строятся по тому же принципу, что и к свойствам их элементов: /users/count . Для свойств коллекций применим только метод GET (получение свойства), т.к. коллекция – это только интерфейс для доступа к списку.

## Коллекции связанных объектов

Одной из разновидностей свойств объектов могут быть связанные объекты или коллекции связанных объектов. Такие сущности, как правило, не являются собственным свойством объекта, а лишь отсылками к его связям с другими сущностями. Например, перечень ролей, которые были присвоены пользователю /users/25/roles

## Функции объектов и коллекций

Для построения пути к интерфейсу вызова функции у коллекции или объекта мы используем тот же самый подход, что и для обращения к свойству. Например, для объекта /users/25/sendPasswordReminder или коллекции /users/disableUnconfirmed. Для вызовов функций мы в любом случае используем метод POST. Почему? Напомню, что в классическом REST не существует специального глагола для вызова функций, а потому нам придется использовать один из существующих. На мой взгляд, для этого больше всего подходит метод POST т.к. он позволяет передавать на сервер необходимые аргументы, не является идемпотентным (возвращающим один и тот же результат при многократном обращении) и наиболее абстрактен по семантике.

# Ответы

## Что такое «классы состояния»

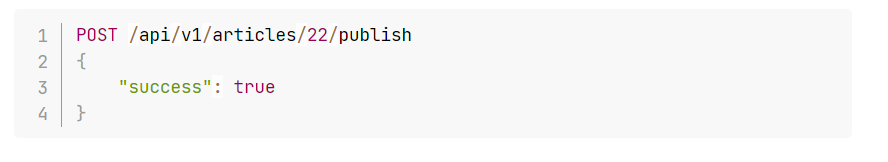
Это группы кодов, объединенных определенными признаками. На класс состояния указывает первая цифра в коде.

Выделяют пять классов:

1. **1xx** — информационные коды. Они отвечают за процесс передачи данных. Это временные коды, они информируют о том, что запрос принят и обработка будет продолжаться.
2. **2xx** — успешная обработка. Запрос был получен и успешно обработан сервером.
3. **3xx** — перенаправление (редирект). Эти ответы сервера гласят, что нужно предпринять дальнейшие действия для выполнения запроса. Например, сделать запрос по другому адресу.
4. **4xx** — ошибка пользователя. Это значит, что запрос не может быть выполнен по его вине.
5. **5xx** — ошибка сервера. Эти коды возникают из-за ошибок на стороне сервера. В данном случае пользователь всё сделал правильно, но сервер не может выполнить запрос. Для кодов этого класса сервер обязательно показывает сообщение, что не может обработать запрос и по какой причине.

## Success — маркер успешности выполнения запроса

Для того, чтобы при получении ответа от сервера сразу понять, увенчался ли запрос успехом, и передать его соответствующему обработчику, достаточно использовать маркер успешности «success». Самый простой ответ сервера, не содержащий никаких данных, будет выглядеть так:



## Коды ответов

201 - created

## Error — сведения об ошибке

В случае, если выполнение запроса завершилось неудачей — о причинах и разновидностях отрицательных ответов сервера поговорим чуть позже, — к ответу добавляется атрибут «error», содержащий в себе HTTP-код статуса и текст сообщения об ошибке. Прошу не путать с сообщениями об ошибках валидации данных для конкретных полей. Правильнее всего, на мой взгляд, возвращать код статуса и в заголовке ответа, но я встречал и другой подход — в заголовке всегда возвращать статус 200 (успех), а детали и возможные данные об ошибках передавать в теле ответа.



## Data — данные, возвращаемые сервером

Большинство ответов сервера призваны возвращать данные. В зависимости от типа запроса и его успеха ожидаемый набор данных будет разным, тем не менее атрибут«data» будет присутствовать в подавляющем большинстве ответов.

Пример возвращаемых данных в случае успеха. В данном случае ответ содержит запрашиваемый объект user.



Пример возвращаемых данных в случае ошибки. В данном случае содержит имена полей и сообщения об ошибках валидации.



## Pagination — сведения, необходимые для организации постраничной навигации

Помимо собственно данных, в ответах, возвращающих набор элементов коллекции, обязательно должна присутствовать информация о постраничной навигации (пагинации) по результатам запроса.

Минимальный набор значений для пагинации состоит из:

* общего числа записей;
* числа страниц;
* номера текущей страницы;
* числа записей на странице;
* максимального числа записей на странице, поддерживаемого серверной стороной.

Некоторые разработчики web API также включают в пагинацию набор готовых ссылок на соседние страницы, а также первую, последнюю и текущую.



## Коды ошибок (исключений)

#### 500 Internal server error — всё сломалось, но мы скоро починим

Это как раз тот случай, когда проблема произошла на стороне самого сервера, и клиентскому приложению остаётся только вздохнуть и уведомить пользователя о том, что сервер устал и прилёг отдохнуть. Например, утеряно соединение с базой данных или в коде завелся баг.

#### 400 Bad request — а теперь у вас всё сломалось

Ответ прямо противоположный предыдущему. Возвращается в тех случаях, когда клиентское приложение отправляет запрос, который в принципе не может быть корректно обработан, не содержит обязательных параметров или имеет синтаксические ошибки. Обычно это лечится повторным прочтением документации к web API.

#### 401 Unauthorized — незнакомец, назови себя

Для доступа к этому ресурсу требуется авторизация. Разумеется, наличие авторизации не гарантирует того, что ресурс станет доступным, но не авторизовавшись, вы точно этого не узнаете. Возникает, например, при попытке обратиться к закрытой части API или при истечении срока действия текущего токена.

#### 403 Forbidden — вам сюда нельзя

Запрашиваемый ресурс существует, но у пользователя недостаточно прав на его просмотр или модификацию.

#### 404 Not found — по этому адресу никто не живёт

Такой ответ возвращается, как правило, в трёх случаях: путь к ресурсу неверен (ошибочен), запрашиваемый ресурс был удалён и перестал существовать, права текущего пользователя не позволяют ему знать о существовании запрашиваемого ресурса. Например, пока просматривали список товаров, один из них внезапно вышел из моды и был удалён.

#### 405 Method not allowed — нельзя такое делать

Эта разновидность исключения напрямую связана с использованным при запросе глаголом (GET, PUT, POST, DELETE), который, в свою очередь, свидетельствует о действии, которое мы пытаемся совершить с ресурсом. Если запрошенный ресурс не поддерживает указанное действие, сервер говорит об этом прямо.

#### 422 Unprocessable entity — исправьте и пришлите снова

Одно из самых полезных исключений. Возвращается каждый раз, когда в данных запроса существуют логические ошибки. Под данными запроса мы подразумеваем либо набор параметров и соответствующих им значений, переданных методом GET, либо поля объекта, передаваемого в теле запроса методами POST, PUT и DELETE. Если данные не прошли валидацию, сервер в секции «data» возвращает отчет о том, какие именно параметры невалидны и почему.

Протокол HTTP поддерживает намного большее число различных статус-кодов на все случаи жизни, но на практике они используются редко и в контексте web API не несут практической пользы

# Запросы

## Получение элементов коллекции

Одним из наиболее частотных запросов является запрос на получение элементов коллекции. Информационные ленты, списки товаров, различные информационные и статистические таблицы и многое другое клиентское приложение отображает посредством обращения к коллекционным ресурсам. Для осуществления этого запроса мы обращаемся к коллекции, используя метод GET и передавая в строке запроса дополнительные параметры. Как мы уже обозначили выше, в качестве ответа мы ожидаем получить массив однородных элементов коллекции и информацию, необходимую для пагинации — подгрузки продолжения списка или же конкретной его страницы. Содержимое выборки может быть особым способом ограничено и отсортировано с помощью передачи дополнительных параметров.

## Постраничная навигация

*page* — параметр указывает на то, какая страница должна быть отображена. Если этот параметр не передан, то отображается первая страница. Из первого же успешного ответа сервера будет ясно, сколько страниц имеет коллекция при текущих параметрах фильтрации. Если значение превышает максимальное число страниц, то разумнее всего вернуть ошибку *404 Not found*.

GET /api/v1/news?page=1

perPage — указывает на желаемое число элементов на странице. Как правило, API имеет собственное значение по умолчанию, которое возвращает в качестве поля perPage в секции pagination, но в ряде случаев позволяет увеличивать это значение до разумных пределов, предоставив максимальное значение maxPerPage:

GET /api/v1/news?perPage=100

## Сортировка результатов

Зачастую результаты выборки требуется упорядочить по возрастанию или убыванию значений определенных полей, которые поддерживают сравнительную (для числовых полей) или алфавитную (для строковых полей) сортировку. Например, нам нужно упорядочить список пользователей по имени или товары по цене. Помимо этого мы можем задать направление сортировки от A до Я или в обратном направлении, причём разное для разных полей.

sortBy — существует несколько подходов к передаче данных о сложной сортировке в GET параметрах. Здесь необходимо четко указать порядок сортировки и направление.

В некоторых API это предлагается сделать в виде строки:

GET /api/v1/products?sortBy=name.desc,price.asc

В других вариантах предлагается использовать массив:

GET /api/v1/products?

sortBy[0][field]=name&

sortBy[0][direction]=desc&

sortBy[1][field]=price&

sortBy[1][direction]=asc

В целом оба варианта равносильны, так как передают одни и те же инструкции.

## Простая фильтрация по значению

Для того, чтобы отфильтровать выборку по значению какого либо поля, в большинстве случаев достаточно передать в качестве фильтрующего параметра имя поля и требуемое значение. Например, мы хотим отфильтровать статьи по ID автора:

GET /api/v1/articles?authorId=25

## Усложнённые варианты фильтрации

Многие интерфейсы требуют более сложной системы фильтрации и поиска. Перечислю основные и наиболее часто встречаемые варианты фильтрации.

Фильтрация по верхней и нижней границе с использованием операторов сравнения from (больше или равно), higher (больше), to (меньше или равно), lower (меньше). Применяется к полям, значения которых поддаются ранжированию.

GET /api/v1/products?price[from]=500&price[to]=1000

Фильтрация по нескольким возможным значениям из списка. Применяется к полям, набор возможных значений которых ограничен, например, фильтр по нескольким статусам:

GET /api/v1/products?status[]=1&status[]=2

Фильтрация по частичному совпадению строки. Применяется к полям, содержащим текстовые данные или данные, которые могут быть приравнены к текстовым, например, числовые артикулы товаров, номера телефонов и т. д.

GET /api/v1/users?name[like]=John

GET /api/v1/products?code[like]=123

## Именованные фильтры

В некоторых случаях, когда определенные наборы фильтрационных параметров часто употребимы и подразумеваются системой как нечто целостное, особенно если затрагивают внутреннюю, зачастую сложную механику формирования выборки, целесообразно сгруппировать их в так называемые именованные фильтры. Достаточно передать в запросе имя фильтра, и система построит выборку автоматически.

GET /api/v1/products?filters[]=recommended

Именованные фильтры могут также иметь свои параметры.

GET /api/v1/products?filters[recommended]=kidds

# Разные рекомендации

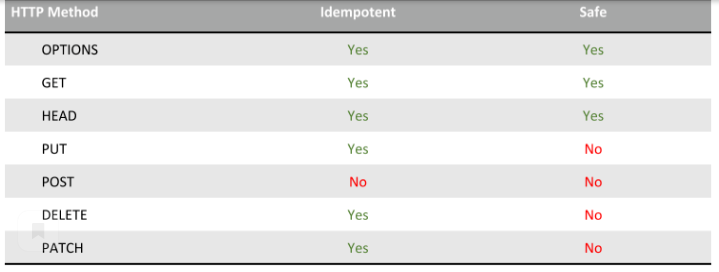
## URI

* Скройте технологические расширения файлов сценариев на стороне сервера (.jsp, .php, .asp), если они есть, для возможности преобразования в другой язык сценариев без изменения URI.
* Набирайте все символы в нижнем регистре.
* Подставляйте вместо пробелов дефисы или знаки подчеркивания.
* Максимально избегайте использования строк запросов.
* Если URI запроса предназначен для частичного пути, вместо использования кода 404 Not Found всегда предоставляйте в качестве ответа страницу по умолчанию или ресурс.
* URI также должны быть статическими, чтобы при изменении ресурсов или реализации службы не менялись ссылки. Это позволяет пользоваться закладками. Также важно, чтобы взаимосвязи между ресурсами, закодированными в URI, оставались независимыми от способа представления этих взаимосвязей там, где они хранятся.

## Общее

* **SSL повсюду — самое важное в вашем сервисе, т. к. без SSL авторизация и аутентификация бессмысленны.**
* **Документация и версионность сервиса — с первого дня работы.**
* **Методы POST и PUT должны возвращать обратно объект, который они изменили или создали, — это позволит сократить время обращения к сервису вдвое.**
* **Поддержка фильтрации, сортировки и постраничного вывода — очень желательно, чтобы это было стандартно и работало «из коробки».**
* **Поддержка MediaType. MediaType — способ сказать серверу, в каком формате вы хотите получить содержимое. Если вы возьмете какую-либо стандартную реализацию web API и зайдете туда из браузера, API отдаст вам XML, а если зайдете через какой-нибудь Postman, он вернет JSON.**
* **Prettyprint & gzip. Не минимизируйте запросы и не делайте компакт для JSON (того ответа, который придет от сервера). Накладные расходы на prettyprint —единицы процентов, что видно, если посмотреть, сколько занимают табы по отношению к общему размеру сообщения. Если вы уберете табы и будете присылать все в одну строку, запаритесь с отладкой. Что касается gzip, он дает выигрыш в разы. Т. ч. очень советую использовать и prettyprint, и gzip.**
* **Используйте только стандартный механизм кэширования (ETag) и Last-Modified (дата последнего изменения) — этих двух параметров серверу достаточно, чтобы клиент понял, что содержимое не требует обновления. Придумывать что-то свое тут не имеет смысла.**
* **Всегда используйте стандартные коды ошибок HTTP. Иначе вам однажды придется кому-нибудь объяснять, почему вы решили, что ошибку 419 в вашем проекте клиенту нужно трактовать именно так, как вы почему-то придумали. Это неудобно и некрасиво — за это клиент вам спасибо не скажет!**

## Свойства HTTP-методов



Сегодня мы будем говорить только про GET, POST, PUT, DELETE.

Если говорить вкратце об остальных, представленных в таблице, OPTIONS — получение настроек безопасности, HEAD — получение заголовков без тела сообщения, PATCH — частичное изменение содержимого.

Как вы видите, все методы, кроме POST, представленные в таблице, идемпотентны. Идемпотентность — возможность выполнить одно и то же обращение к сервису несколько раз, при этом ответ каждый раз будет одинаковым. Другими словами, не важно, по какой причине и сколько раз вы выполнили это действие. Допустим, вы выполняли действие по изменению объекта (PUT), и вам пришла ошибка. Вы не знаете, что ее вызвало и в какой момент, вы не знаете, изменился объект или нет. Но, благодаря идемпотентности, вы гарантированно можете выполнить этой действие еще раз, т. ч. клиенты могут быть спокойны за целостность своих данных.

"Safe" же значит, что обращение к серверу не изменяет содержимое. Так, GET может быть вызван много раз, но он не изменит никакого содержимого. Если бы он изменял содержимое, в силу того, что GET может быть закэширован, вам пришлось бы бороться с кэшированием, изобретать какие-нибудь хитрые параметры.

# Непосредственно АПИ

Схемы (сущности):

* Чат
* Сообщение
* Пользователь (отдельная сущность)

openapi: 3.0.0

info:

version: '1.0'

title: 'Messenger`s server API'

description: API for server of OSU messenger

servers:

# Added by API Auto Mocking Plugin

- description: SwaggerHub API Auto Mocking

url: https://virtserver.swaggerhub.com/MyPersonalOrg3/MessengerServer/1.0

tags:

- name: chat

description: Everything about chat

- name: user

description: Everything about user

- name: document

description: Everything about document

paths:

########## Все чаты ##########

/api/v1/{userId}/chats:

parameters:

- $ref: '#/components/parameters/userId'

get:

tags:

- chat

summary: Returns a list of chats of user

responses:

'200':

description: Successfully returned a list of chats if this student

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/Chat'

post:

tags:

- chat

summary: Add new chat

responses:

'201':

description: Created chat and returns some metainfo

#какой инбудь код ошибки

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/Chat'

########## Конкретный чат ##########

/api/v1/chats/{chatId}:

parameters:

- $ref: '#/components/parameters/chatId'

get:

tags:

- chat

summary: Returns a chat by {chatId}

responses:

'200':

description: Successfully returned chat

content:

application/json:

schema:

$ref: '#/components/schemas/Chat'

put:

tags:

- chat

summary: Change information about chat by {chatId}

responses:

'200':

description: Successfully changed info about chat

'403':

description: Allowed only for admin

'404':

description: There is no such chat

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/Chat'

delete:

tags:

- chat

summary: Delete chat

description: This only can be done by admin

responses:

'200':

description: Chat deleted successfully

'403':

description: Allowed only for admin

'404':

description: There is no such chat

########## Пользователи одного чата ##########

/api/v1/chats/{chatId}/users:

parameters:

- $ref: '#/components/parameters/chatId'

get:

tags:

- user

summary: Returns all users of {chatId} chat

responses:

'200':

description: Successfully returned users

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/User'

post:

tags:

- user

summary: Add new user to {chatId} chat

responses:

'200':

description: Successfully added new user

#код ошибки добавления

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/User'

########## Конкретный пользователь чата ##########

/api/v1/chats/{chatId}/users/{userId}:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

get:

tags:

- user

summary: Returns info about {userId} user

responses:

'200':

description: Successfully returned info users

content:

application/json:

schema:

$ref: '#/components/schemas/User'

#нужно ли? Скорее всего это выносится вверх, между /v1 и /chats

put:

tags:

- user

summary: Change info about {userId} user

responses:

'200':

description: Successfully changed info

'404':

description: There is no such user

delete:

tags:

- user

summary: Change info about {userId} user

responses:

'200':

description: Successfully changed info

'403':

description: Allowed only for admin

'404':

description: There is no such user

########## Документы чата ##########

/api/v1/chats/{chatId}/documents:

parameters:

- $ref: '#/components/parameters/chatId'

get:

tags:

- document

summary: Returns all documents of chat {chatId}

responses:

'200':

description: Successfully returned documents

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/Document'

post:

tags:

- document

summary: Create metadata about new document in chat {chatId}

responses:

'200':

description: Successfully changen info about chat

'201':

description: Created metadata and returns that with doc`s id

requestBody:

content:

application/json:

schema:

$ref: '#/components/schemas/DocumentMetainfo'

########## Метаданные об одном документа чата ##########

/api/v1/chats/{chatId}/documents/{documentId}:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/documentId'

get:

tags:

- document

description: Returns a metadata about {documentId} document

responses:

'200':

description: Successfully returned a metadata about {documentId} document

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/DocumentMetainfo'

put:

tags:

- document

description: Change a metadata about {documentId} document

responses:

'200':

description: Successfully changed a metadata about {documentId} document

content:

application/json:

schema:

$ref: '#/components/schemas/DocumentMetainfo'

########## Один документ ##########

/api/v1/chats/{chatId}/documents/{documentId}/data:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/documentId'

get:

tags:

- document

description: Returns a metadata about {documentId} document

responses:

'200':

description: Successfully returned a document by {documentId} document

content:

application/json:

schema:

$ref: '#/components/schemas/DocumentMetainfo'

########## Все документы пользователя ##########

/api/v1/chats/{chatId}/users/{userId}/documents:

parameters:

- $ref: '#/components/parameters/chatId'

- $ref: '#/components/parameters/userId'

get:

tags:

- user, document

description: Returns info about all documents of {userId} user

responses:

'200':

description: Successfully returned info of all docs

content:

application/json:

schema:

type: array

items:

$ref: '#/components/schemas/DocumentMetainfo'

components:

schemas:

Chat:

type: object

properties:

name:

type: string

example: 71ПГ

messages:

type: array

items:

# ----- Added line --------------------------------

$ref: '#/components/schemas/Message'

# ---- /Added line --------------------------------

messagesCount:

type: integer

minimum: 0

example: 10

participants:

type: array

items:

# ----- Added line --------------------------------

$ref: '#/components/schemas/User'

# ---- /Added line --------------------------------

attachments:

type: array

items:

# ----- Added line --------------------------------

$ref: '#/components/schemas/DocumentMetainfo'

# ---- /Added line --------------------------------

Message:

type: object

properties:

data:

type: string

example: Какой-то текст

sender:

$ref: '#/components/schemas/User'

recipient: #получатель

$ref: '#/components/schemas/User'

date:

type: string

format: date-time

example: 01.03.2021T17:21:20

#attachments:

#type: array

#items:

User:

type: object

properties:

id:

type: integer

example: 1

minimum: 1

role:

type: string

enum: [student, teacher, admin]

example: student

name:

type: string

example: Владислав

surname:

type: string

example: Шорин

patronymic:

type: string

example: Дмитриевич

DocumentMetainfo:

type: object

properties:

id:

type: integer

example: 1

minimum: 1

date:

type: string

example: 01.03.2021T17:21:20

#################################### Исправить, когда будет инфа как передавать

Document:

type: object

properties:

id:

type: integer

example: 1

minimum: 1

date:

type: string

example: 01.03.2021T17:21:20

parameters:

chatId:

name: chatId

in: path

required: true

description: Numeric ID of the chat to get

schema:

type: integer

minimum: 1

documentId:

name: documentId

in: path

required: true

description: Numeric ID of the document to get

schema:

type: integer

minimum: 1

userId:

name: userId

in: path

required: true

description: Numeric ID of the user to get

schema:

type: integer

minimum: 1

Date:

name: date

in: query

schema:

type: string #string?

Time:

name: time

in: query

schema:

type: string #string?