

API+OAS 3.0+Yaml

При проектировании API в первую очередь думаем о потребителе. Четко определить цели, которые потребители могут достичь при использовании API, чтобы обеспечить создание простого для понимания и несложного в применении API.

При разработке API важно иметь глубокие и точные знания о том,

- кто может использовать API;
- что он может делать;
- как он это делает;
- что ему нужно для этого;
- что он получает взамен.

Два основных вопроса, задаваемых при определении списка целей API:

- Что могут делать пользователи?
- Как они это делают?

С помощью этих вопросов мы примерно описываем, что можно сделать с помощью API (что), и разбиваем их на этапы (как), при этом каждый этап становится целью API.

Для достижения цели могут потребоваться входные данные.
Цель может даже вернуть выходные данные, когда она будет достигнута.

Новый вопрос для определения недостающих целей

- откуда поступают входные данные?
- как используются выходные данные?

Идентификация различных типов пользователей является обязательной при построении исчерпывающего списка целей API. Таким образом, мы должны добавить еще один запрос к нашей линии вопросов, чтобы явно идентифицировать их все:

- кто такие пользователи?

Нарисуйте эту

 на обычной или магнитной доске, листе бумаги или в электронном виде и начните опрос



Чего необходимо избегать

Проектирования API, находящегося под сильным влиянием точки зрения поставщика.

Может влиять на проектирование ее API:

данные(1),

код и бизнес-логика(2),

архитектура программного обеспечения(3) ,

общественная организация, коммуникационная структура компании(4)

1. Если список целей и данные вашего API слишком совпадают с вашей базой данных – будь то по структуре или названию, – возможно, вы проектируете свой API с точки зрения поставщика. В этом случае не стесняйтесь перепроверить, действительно ли пользователям API нужно иметь доступ к таким деталям.

2. При определении целей API всегда следует проверять, чтобы вы случайно не предоставили доступ к внутренней бизнес-логике, которая не касается потребителя и которая может быть опасной для поставщика.

3. Сопоставление дизайна API с базовой программной архитектурой может затруднить понимание и использование API потребителями. Поэтому, определяя цели API, вы всегда должны проверять, что ваши «что» и «как» не являются результатом вашей базовой архитектуры программного обеспечения.

4. Поэтому, когда вы определяете цели API, всегда следует проверять, действительно ли ваши «что» и «как» имеют отношение к потребителю.

Перенос целей в REST API

- Цели переносятся в пары типа «ресурс и действие». Ресурсы идентифицируются путями, а действия представлены методами HTTP





Каждый ресурс товара – это элемент в коллекции каталога, идентифицируемый как /catalog, поэтому можно выбрать путь /catalog/{productId} для обозначения товара.

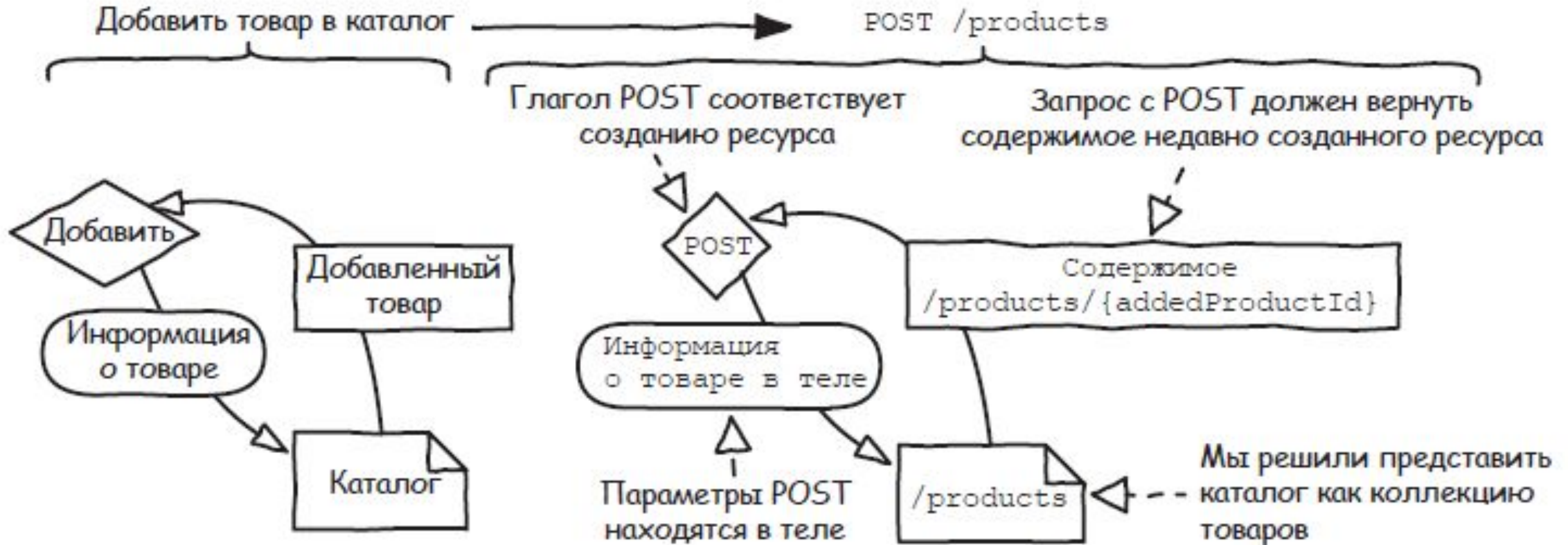
Также можно явно указать, что каталог – это коллекция ресурсов товара, используя путь /products, при этом товар из этой коллекции представлен путем /products/{productId}.

Хотя официальных правил REST, касающихся проектирования путей к ресурсам (кроме уникальности), не существует, наиболее распространенным форматом является

/ {имя, отражающее тип элемента коллекции во множественном числе} / {идентификатор элемента}.

Эту структуру можно расширить до нескольких уровней, например: /resources/{resourceId}/sub-resources/{subResourceId}.

Представление действий с помощью протокола HTTP



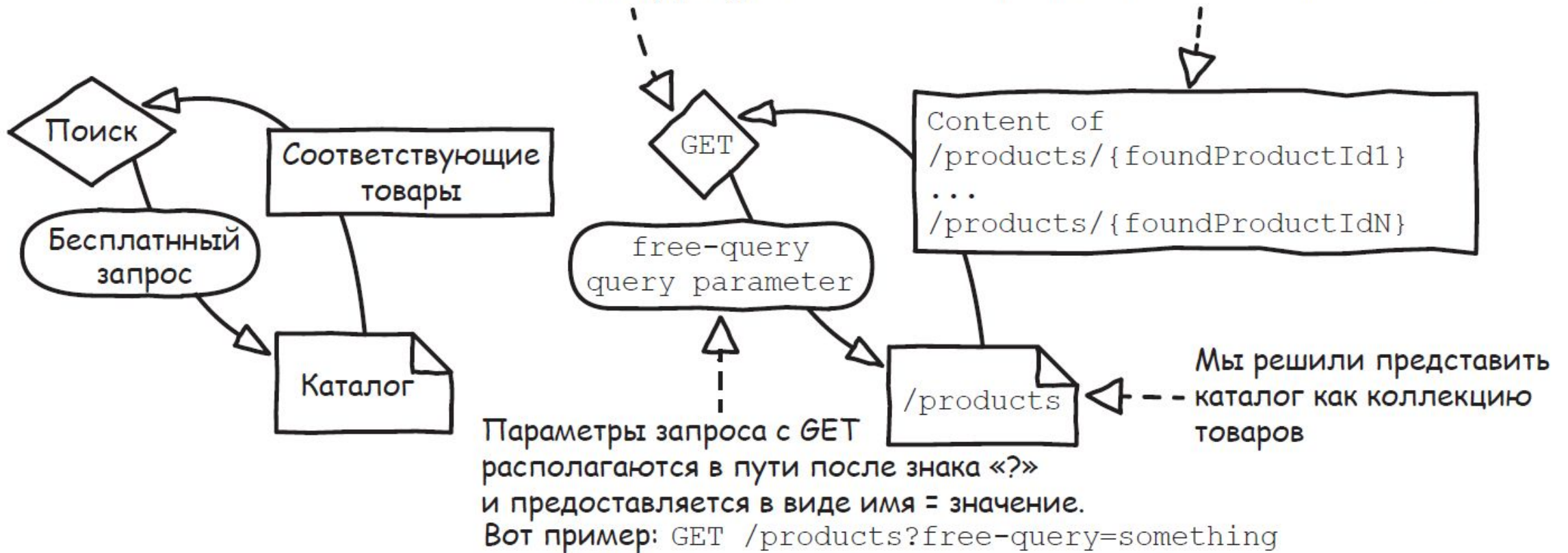
- HTTP-метод, соответствующий созданию ресурса, – это POST

Поиск товаров в каталоге
с помощью бесплатного запроса

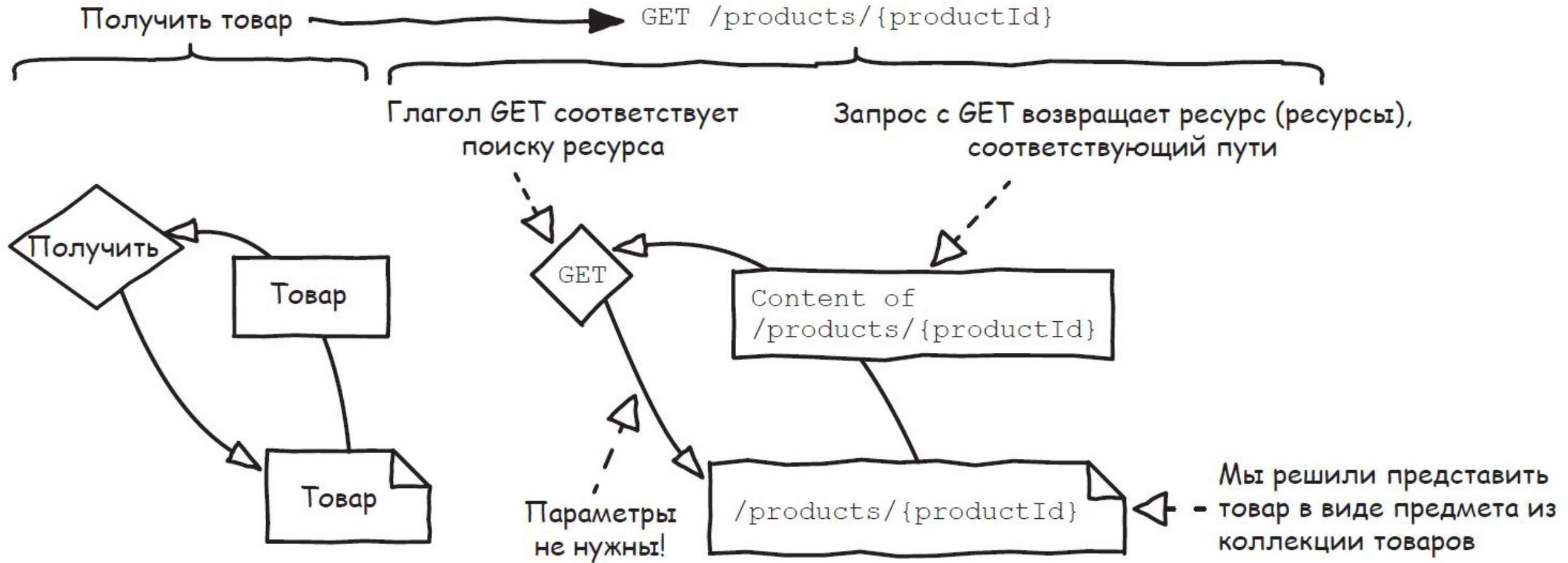
GET /products?free-query={freeQuery}

Глагол GET соответствует
поиску ресурса

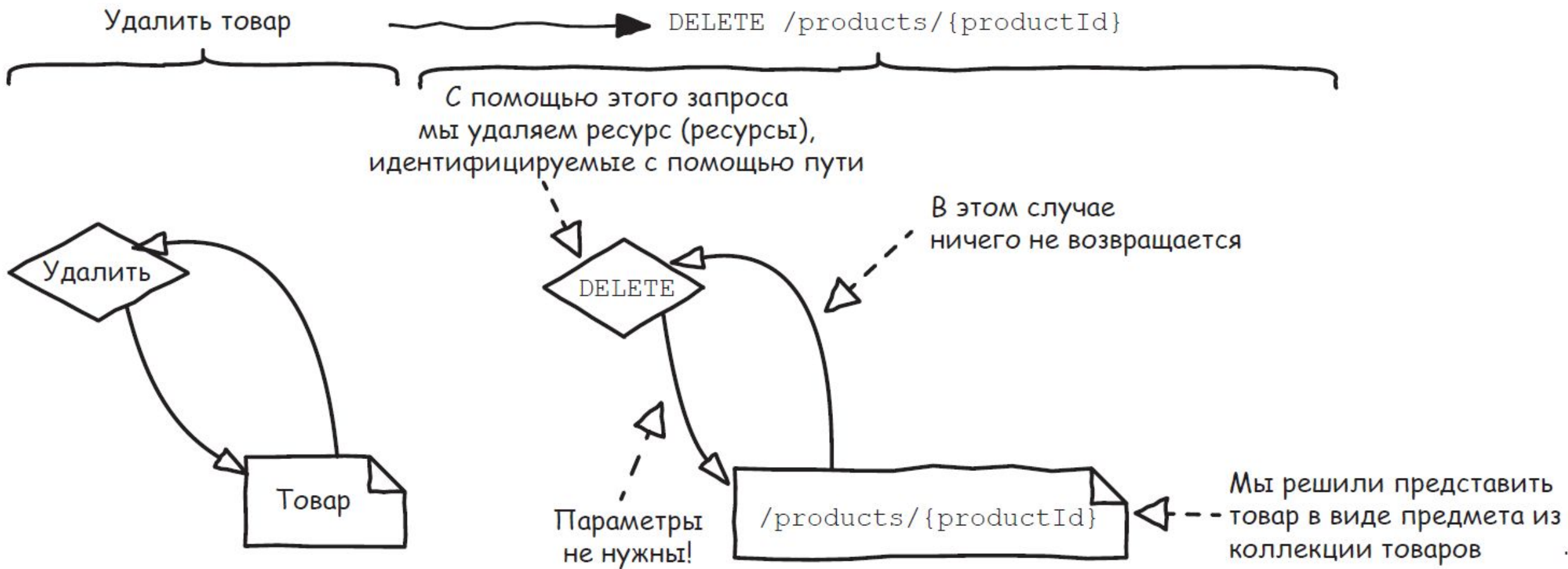
Запрос с GET возвращает ресурс
(ресурсы), соответствующий пути



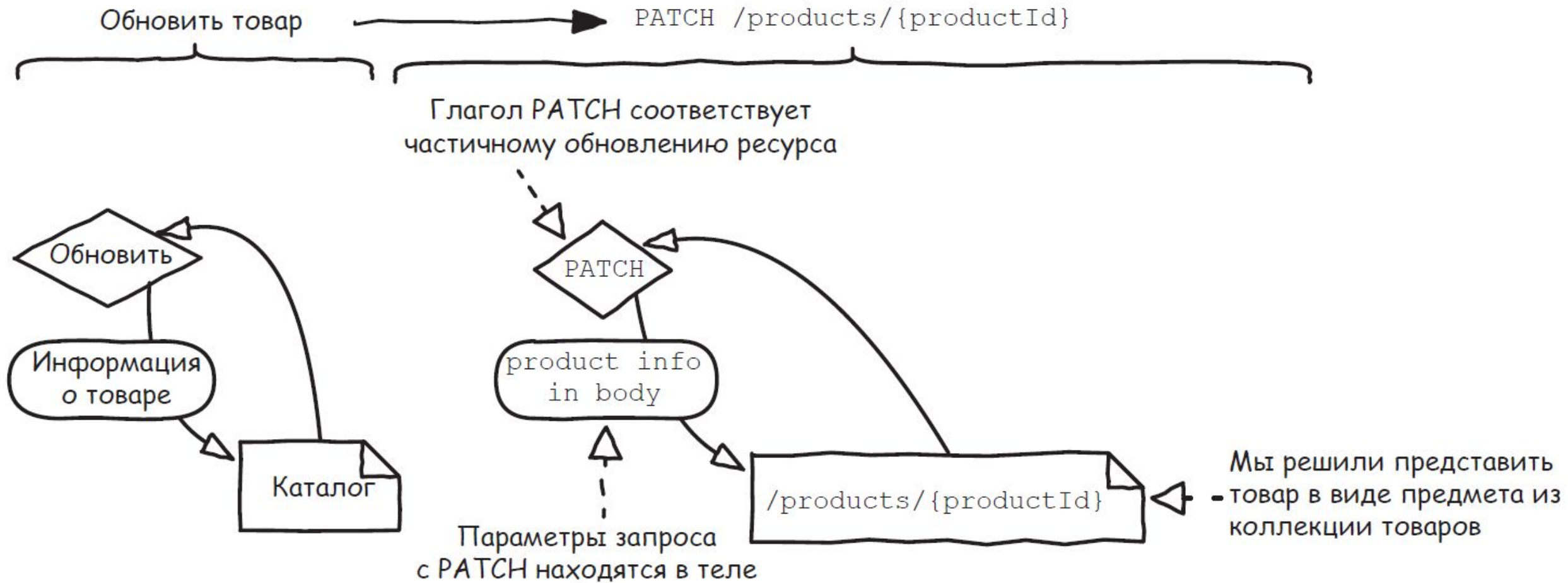
- Поиск HTTP-метод GET в пути /products.
- Чтобы получить только те товары, которые соответствуют некоему запросу, например названию товара, нужно передать параметр в этот запрос.



- Чтобы получить ресурс – один товара, обозначенный как `/products/{productId}`, также используем HTTP-метод GET.

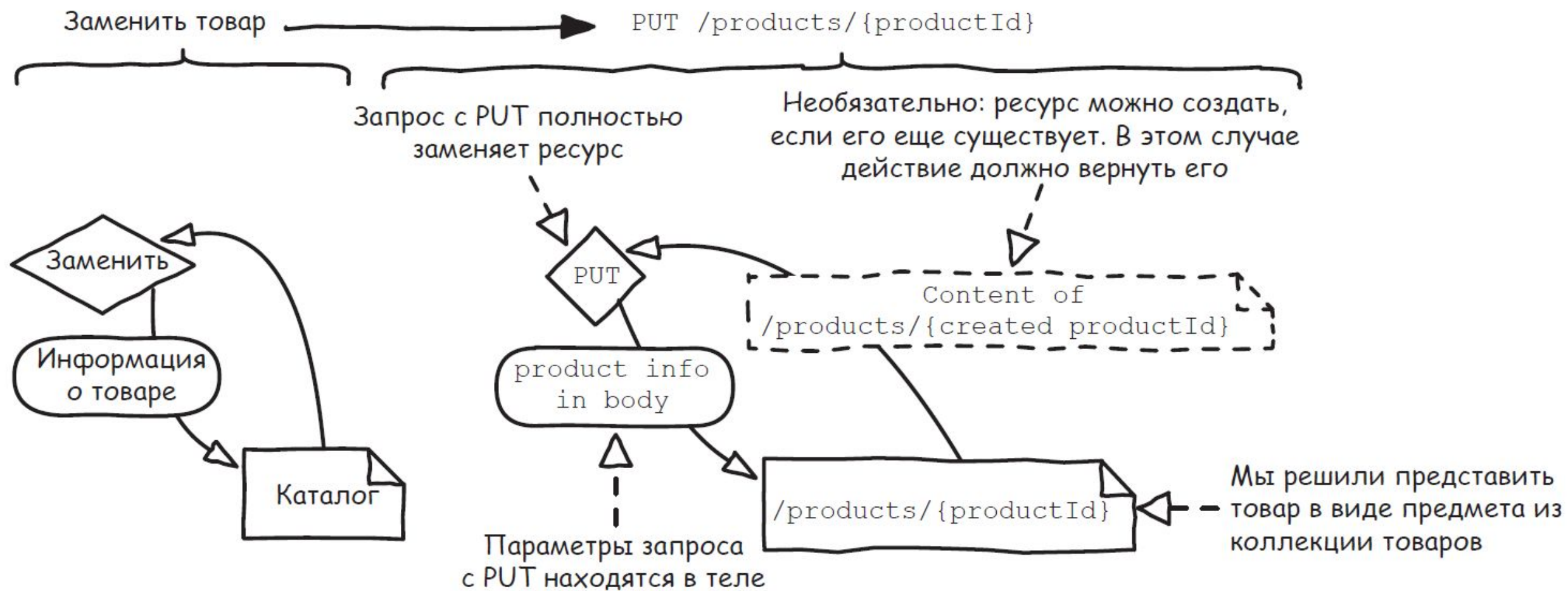


- Удаление товара с помощью протокола HTTP - DELETE /products/{productId}

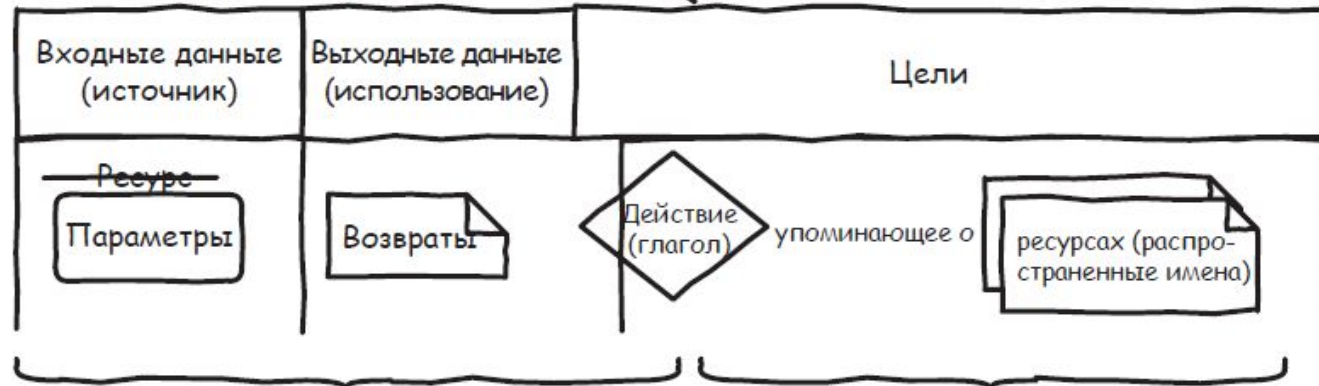


- HTTP-представление обновления товара – это PATCH /products/{productId}
- HTTP-метод PATCH можно использовать для частичного обновления

- HTTP-метод, имеющий две цели.
- HTTP-представление замены товара – PUT /products/{productId}
- HTTP-метод PUT можно использовать для полной замены существующего ресурса или для создания несуществующего и предоставления его идентификатора.
- В последнем случае эффект будет тот же, что и при добавлении товара в каталог. Как и в случае с POST, параметры запроса передаются в теле запроса.



После создания таблицы целей API (см. Главу 2)



3 Определите действия для каждого ресурса и их параметры и верните их, используя входные и выходные данные и цели




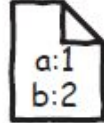
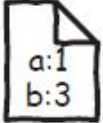
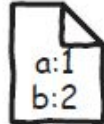









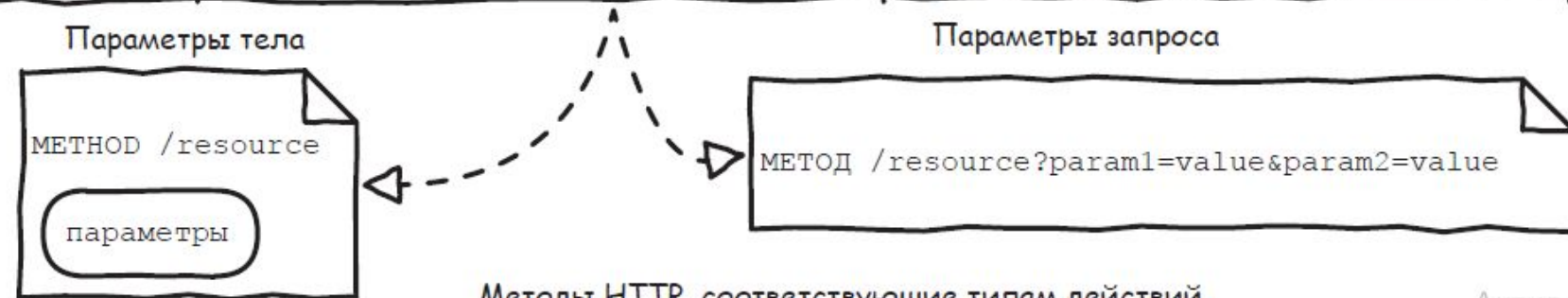
2 Определите ресурсы и их отношения, используя цели

Существует пять HTTP-методов для представления целей действия: GET, POST, PATCH, PUT, DELETE



4 Проектирование путей к ресурсам и выбор HTTP-методы соответствующие действиям

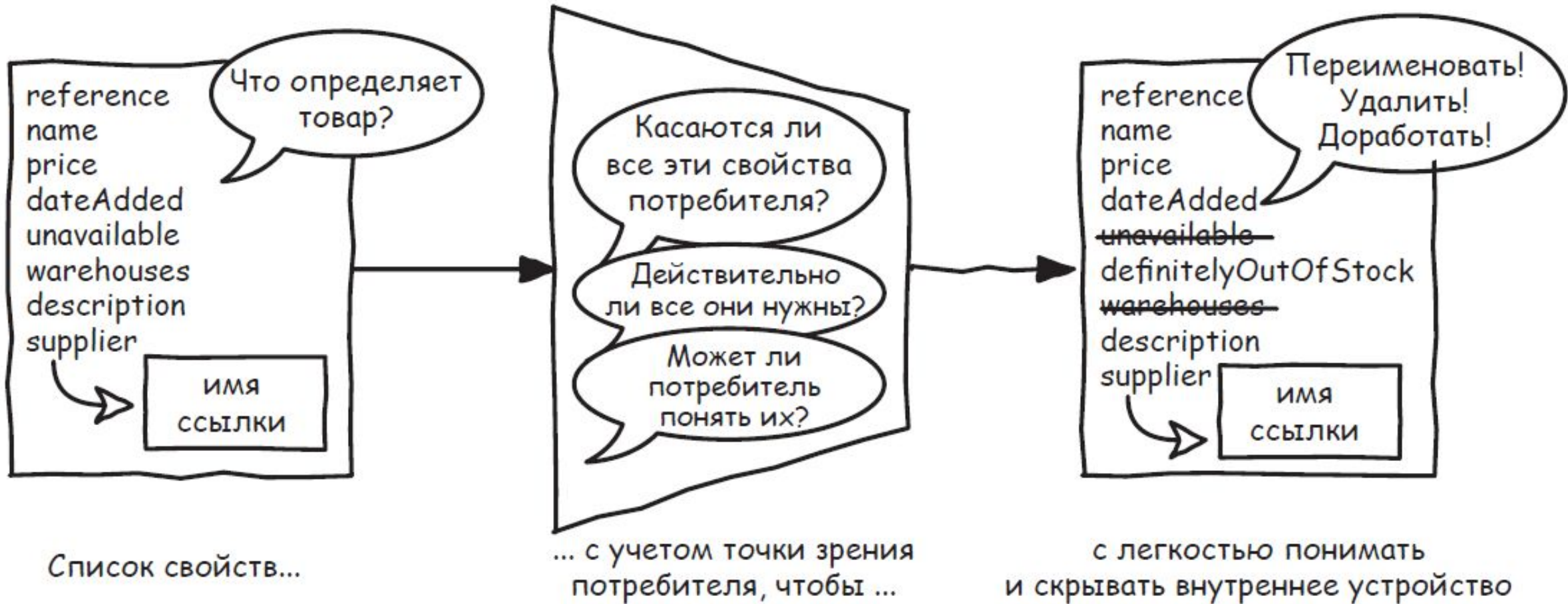
Действие	HTTP -метод	Параметры	До запроса	После запроса
Создать, добавить в	POST	Тело		 
Читать, получать, искать	GET		Без изменений	
Изменить, частично обновить	PATCH	Тело		
Заменить	PUT	Тело		
Создать с предопределенным идентификатором	PUT	Тело		 
Переместить, удалить	DELETE	Запрос	 	



Методы HTTP, соответствующие типам действий

Проектирование данных API

- Перечисление свойств структуры данных и назначение каждому свойству имя.
- Надо проанализировать каждую из них, чтобы убедиться, что наш дизайн ориентирован на точку зрения потребителя и не отражает точку зрения поставщика.
- Это можно сделать, спросив себя, можно ли понять каждое свойство, действительно ли это касается потребителя и действительно ли оно полезно.



- Для каждого свойства необходимо собрать следующие характеристики:

- его имя;

- его тип;

- является ли оно обязательным;

- необязательное описание при необходимости

- СЕ

МЭ

Обязательные характеристики

Обязательные характеристики			
Имя	Тип	Обязательно	Описание
Ссылка	Строка	Да	Уникальный идентификатор товара



Чем больше
ясности, тем
лучше!



Переносимые типы
(строка, номер, логи-
ческий тип данных,
дата, массив, объект)



Существенное
это свойство
или нет



Необязательная дополнительная
информация, не показанная ни
по имени, ни по типу

Имя	Тип	Обязательно	
reference	string	да	Уникальный идентификатор товара
name	string	да	
price	number	да	Цена в долларах США
dateAdded	date	да	Когда товар был добавлен в каталог
	boolean	нет	Определенно нет в наличии при наличии значения true
	string	нет	
supplier	object	да	Поставщик товара

Свойства ресурса товара

Пример товара

```
{
  "reference": "R123-456",
  "name": "a product",
  "price": 9.99,
  "dateAdded": "2018-01-02",
  "definitelyOutOfStock": false,
  "supplier": {
    "reference": "S789",
    "name": "a supplier"
  }
}
```

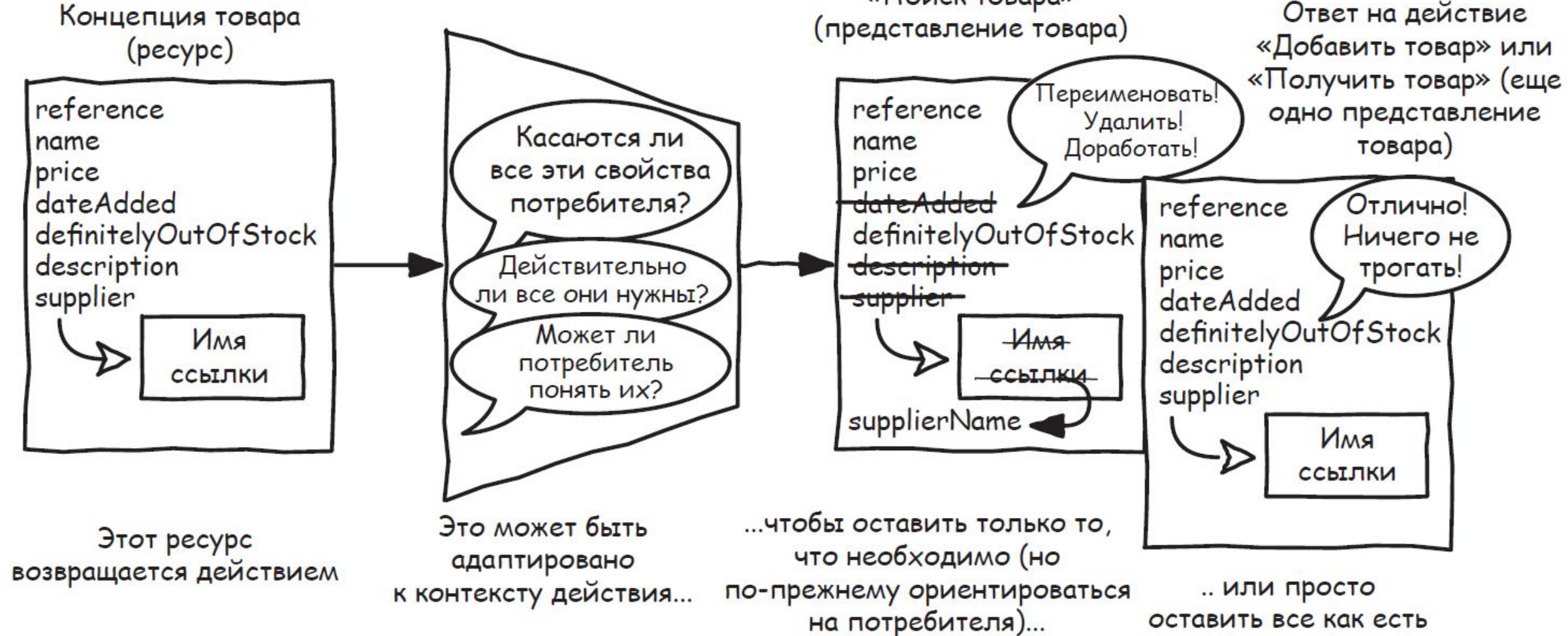
Имя	Тип	Обязательно	
reference	string	да	Уникальный идентификатор поставщика
name	string	да	

Свойства могут быть объектами

Проектирование ответов от концепций

- При проектировании ответов не должны слепо сопоставлять манипулируемый ресурс.
- Необходимо адаптировать их к контексту, удаляя или переименовывая свойства, а также корректируя структуру данных.

Два представления одного и того же понятия
в качестве ответа в трех разных контекстах



Resource	POST	GET	PUT	DELETE
/customers	Create a new customer	Retrieve all customers	Bulk update of customers	Remove all customers
/customers/1	Error	Retrieve the details for customer 1	Update the details of customer 1 if it exists	Remove customer 1
/customers/1/orders	Create a new order for customer 1	Retrieve all orders for customer 1	Bulk update of orders for customer 1	Remove all orders for customer 1

A POST request can also be used to submit data for processing to an existing resource, without any new resource being created.

Идемпотентный метод

- Метод HTTP является идемпотентным, если повторный идентичный запрос, сделанный один или несколько раз подряд, имеет один и тот же эффект, не изменяющий состояние сервера.
- Другими словами, идемпотентный метод не должен иметь никаких побочных эффектов (side-effects), кроме сбора статистики или подобных операций. Корректно реализованные методы GET, HEAD, PUT и DELETE идемпотентны, но не метод POST.
- Также все безопасные методы являются идемпотентными.

- Для идемпотентности нужно рассматривать только изменение фактического внутреннего состояния сервера, а возвращаемые запросами коды статуса могут отличаться: первый вызов DELETE вернёт код 200, в то время как последующие вызовы вернут код 404.
- Из идемпотентности DELETE неявно следует, что разработчики не должны использовать метод DELETE при реализации RESTful API с функциональностью удалить последнюю запись.
- Обратите внимание, что идемпотентность метода не гарантируется сервером, и некоторые приложения могут нарушать ограничение идемпотентности.

Безопасный метод

- Метод HTTP является безопасным, если он не меняет состояние сервера.
- Другими словами, безопасный метод проводит операции "только чтение" (read-only).
- Несколько следующих методов HTTP безопасные: GET, HEAD или OPTIONS.
- Все безопасные методы являются также идемпотентными, как и некоторые другие, но при этом небезопасные, такие как PUT или DELETE.
- Даже если безопасные методы являются по существу "только для чтения", сервер всё равно может сменить своё состояние: например, он может сохранять статистику.

- Что существенно, так то, когда клиент вызывает безопасный метод, то он не запрашивает никаких изменений на сервере, и поэтому не создаёт дополнительную нагрузку на сервер.
- Браузеры могут вызывать безопасные методы, не опасаясь причинить вред серверу: это позволяет им выполнять некоторые действия, например, предварительная загрузка без риска.
- Поисковые роботы также полагаются на вызовы безопасных методов.

- Безопасные методы не обязательно должны обрабатывать только статичные файлы; сервер может генерировать ответ "на-лету", пока скрипт, генерирующий ответ, гарантирует безопасность: он не должен вызывать внешних эффектов, таких как формирование заказов, отправка писем и др..
- Правильная реализация безопасного метода - это ответственность серверного приложения, потому что сам веб-сервер, будь то Apache, nginx, IIS это соблюсти не сможет.
- В частности, приложение не должно разрешать изменение состояния сервера запросами GET.

Кешируемые методы

- Кешируемые ответы - это HTTP-ответы, которые могут быть закешированы, то есть сохранены для дальнейшего восстановления и использования позже, тем самым снижая число запросов к серверу.
- Не все HTTP-ответы могут быть закешированы. Вот несколько ограничений:
 - Метод, используемый в запросе, кешируемый, если это GET или HEAD. Ответ для POST или PATCH запросов может также быть закеширован, если указан признак “свежести” данных и установлен заголовок Content-Location (en-US), но это редко реализуется. Например, Firefox не поддерживает это согласно. Другие методы, такие как PUT и DELETE не кешируемые, и результат их выполнения не кешируется
 - Коды ответа, известные системе кеширования, которые рассматриваются как кешируемые: 200, 203, 204, 206, 300, 301, 404, 405, 410, 414, 501.
 - Отсутствуют специальные заголовки в ответе, которые предотвращают кеширование: например, Cache-Control.

- Обратите внимание, что некоторые некешируемые запросы-ответы к определённым URI могут сделать недействительным (инвалидируют) предыдущие закешированные ответы на тех же URI. Например, PUT к странице pageX.html инвалидируют все закешированные ответы GET или HEAD запросов к этой странице.

OAS 3.0+

Объект "openapi"

- В объекте [openapi](#) указываем версию спецификации OpenAPI для проверки. Последняя версия 3.0.2

Объект "info"

- Объект `info` содержит основную информацию о вашем API, включая заголовок, описание, версию, ссылку на лицензию, ссылку на условия обслуживания и контактную информацию.
- Многие из свойств являются необязательными

title: "OpenWeatherMap API"

description: "Get the current weather."

version: "2.5"

termsOfService: "https://openweathermap.org/terms"

contact:

name: "OpenWeatherMap API"

url: "https://openweathermap.org/api"

email: "some_email@gmail.com"

license:

name: "CC Attribution-ShareAlike4.0 (CC BY-SA 4.0)"

url: "https://openweathermap.org/price"

Объект "servers"

- В объекте `servers` указывается базовый путь, используемый в ваших запросах API.
- Базовый путь - это часть URL, которая находится перед конечной точкой.

`servers:`

- `url: https://api.openweathermap.org/data/2.5/`
`description: Production server`
- `url: http://beta.api.openweathermap.org/data/2.5/`
`description: Beta server`
- `url: http://some-`
`other.api.openweathermap.org/data/2.5/`
`description: Some other server`

Объект "security"

- Объект `security` указывает протокол безопасности или авторизации, используемый при отправке запросов.
- **Выбор схемы безопасности**
- В API REST могут использоваться различные подходы безопасности для авторизации запросов. Рассмотрим наиболее распространенные методы авторизации в Требованиях аутентификации и авторизации. Swagger UI поддерживает четыре схемы авторизации:
 - API ключ;
 - HTTP;
 - OAuth 2.0;
 - Open ID Connect.
- Если ваш API использует OAuth 2.0 или другой метод, надо прочитать Security Scheme information, чтобы узнать, как ее настроить. Тем не менее, все методы безопасности в основном следуют одному и тому же шаблону

Объект "tags"

- Объект `tags` позволяет группировать пути (конечные точки)
- На корневом уровне объект `tags` перечисляет все теги, которые используются в объектах `operations` (которые появляются в объекте `paths`)

`tags:`

- `name: Current Weather Data`

- `description: "Get current weather details"`

- Здесь только один тег, но их может быть столько, сколько вы хотите (если у вас много конечных точек, имеет смысл создать несколько тегов для их группировки)


```
paths:
```

```
  /weather:
```

```
    get:
```

```
      tags:
```

```
        - Current Weather Data
```

- Порядок тегов в объекте `tags` на корневом уровне определяет их порядок в интерфейсе Swagger.
- Кроме того, `descriptions` появляются справа от имени тега.

Объект "externalDocs"

- Объект `externalDocs` позволяет добавлять ссылки на внешнюю документацию.
- Можно добавлять ссылки на внешние документы в объекте `paths`.

`externalDocs:`

`description: API Documentation`

`url: https://openweathermap.org/api`

- Эта документация должна относиться в целом к API.
- Чтобы связать определенный параметр с дополнительной документацией, можно добавить объект `externalDocs` к объекту операции.

Описание API с помощью формата

ОПИСАНИЯ

Создание документа OAS

Минимальный, но допустимый документ OAS:

```
openapi: «3.0.0» ①
```

```
info: ②
```

```
  title: Shopping API
```

```
  version: «1.0»
```

```
paths: {} ③
```

① Версия OAS.

② Общая информация API.

③ Пустые пути

Описание ресурса

openapi: «3.0.0»

info:

title: Shopping API

version: «1.0»

paths: ①

/products: ②

description: The products catalog ③

① Ресурсы API.

② Путь к ресурсу.

③ Описание ресурса.

- **Описание операций в ресурсе**
- Ресурс, описанный в документе OAS, должен содержать какие-то операции.

openapi: «3.0.0»

info:

title: Shopping API

version: «1.0»

paths:

/products: ①

description: The products catalog

get: ②

summary: Search for products ③

description: | ④

Search for products in catalog
using a free query parameter

① Ресурс.

② HTTP-метод действия.

③ Краткое описание действия.

④ Длинное описание действия.

- *свойство `description` является многострочным. Это особенность YAML: чтобы быть многострочным, свойство `string` должно начинаться с символа вертикальной черты (`|`).*
- В документе OAS операция должна описывать по крайней мере один ответ в свойстве `responses`

...

description: | Search for products in catalog
using a free query parameter

responses: ①

"200": ②

description: | ③

Products matching free query parameter

- ① СПИСОК ОТВЕТОВ.
- ② Код ответа 200 ОК.
- ③ Описание ответ.

Описание данных API с помощью OpenAPI и JSON Schema

Описание параметров запроса

- Для поиска товаров с использованием API потребитель должен выполнить запрос `GET /products?Free-query={free query}` (например, `GET /products?free-query=book`)
- Чтобы описать этот параметр, мы добавляем свойство `parameters` внутри операции `get` ресурса `/products`
- Свойство `parameters` представляет собой список или массив.
- В YAML каждый элемент списка или массива начинается с дефиса (-).
- Чтобы описать параметр, нам нужно как минимум три свойства: `name`, `in` и `schema`.
- Описание этого параметра также содержит два необязательных свойства: `required` и `description`.

parameters:

- name: free-query ①

description: | ②

A product's name, reference, or partial description

in: query ③

required: false ④

schema: ⑤

type: string ⑥

① Имя параметра.

② Описание параметра.

③ Расположение параметра.

④ Является ли параметр обязательным.

⑤ Описание структуры данных параметра.

⑥ Тип параметра (строка).

- Чтобы описать объект, мы должны взять тип object и перечислить его свойства. Каждое свойство идентифицируется по имени и типу (обязательные и необязательные свойства)

```
type: object ①
required: ⑥
  - reference
  - name
  - price
properties: ②
  reference: ③
    type: string ④
  name: ③
    type: string ④
  price: ③
    type: number ④
  description: ⑤
    type: string
```

① Эта схема описывает объект.

② Здесь содержатся свойства.

③ Имя свойства.

④ Тип свойства.

⑤ Необязательное свойство.

⑥ Обязательные свойства.

- Документирование схемы JSON

```
properties:  
  reference:  
    type: string  
    description: Product's unique identifier ②  
    example: ISBN-9781617295102 ③
```

- ① Описание объекта.
- ② Описание свойства.
- ③ Пример значения свойства.

- **Описание комплексного свойства с помощью JSON Schema**

supplier: ②

type: object

description: Product's supplier

required: ③

- reference

- name

properties: ④

reference:

type: string

description: Supplier's unique identifier

example: MANPUB

name:

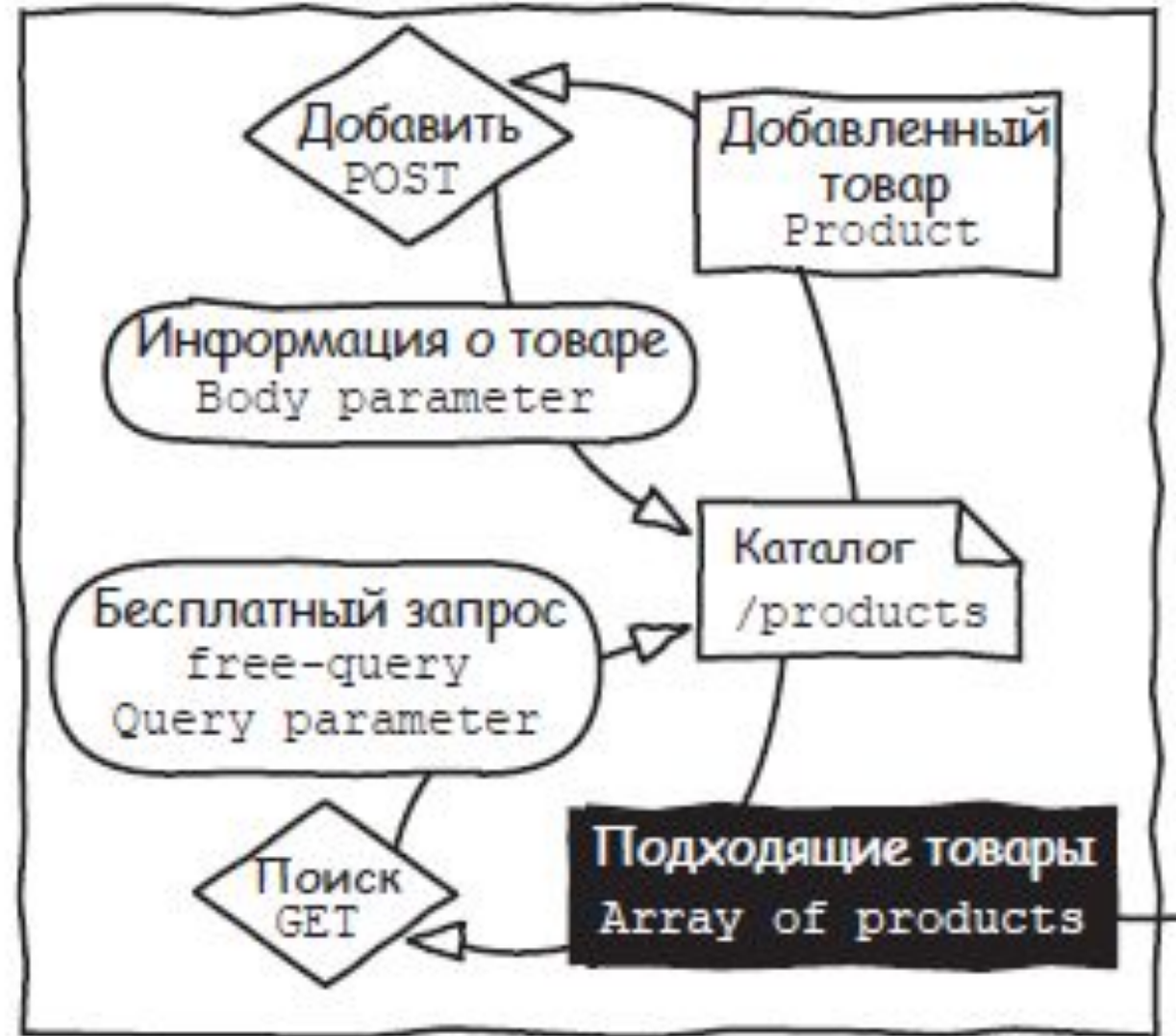
type: string

example: Manning Publications

- Чтобы добавить свойство `supplier` в JSON-схему товара, включаем его в список `properties` и устанавливаем для его типа значение `object`.
- Мы предоставляем описание для свойства и список обязательных свойств (`reference` и `name`); затем описываем эти свойства.
- Для свойства `reference` мы предоставляем `type`, `description` и `example`.
- Для свойства `name` устанавливаем значения только для `type` и `example`.
- Наконец, добавляем свойство `supplier` в список товара `required`.

Описание ответов

- Когда пользователи ищут товары, они должны получать товары, соответствующие предоставленному бесплатному запросу.
- Соответствующий API-запрос `GET /products?free-query={free query}` возвращает HTTP-ответ 200 OK, тело которого будет содержать массив товаров.



- В документе OAS данные, возвращаемые операцией в теле HTTP-ответа, определяются в свойстве `content`, как показано в примере.
- При описании содержимого ответа вы должны указать медиа тип документа, содержащегося в теле ответа.
- Медиа тип указан в HTTP-ответе.
- На данный момент, как было сказано ранее, мы будем считать само собой разумеющимся, что наш API возвращает документы в формате JSON, поэтому мы указываем в ответе `application/json`.

responses:

"200":

description: Products matching free query

content: ①

application/json: ②

schema: ③

[...]

① Определение тела ответа.

② Медиа тип тела ответа.

③ JSON-схема тела ответа.

- Массив описывается с использованием типа `array`.
- Свойство `items` содержит схему элементов массива.

`content:`

`application/json:` ①

`schema:` ②

`type:` `array` ③

`description:` Array of products

`items:` ④

`type:` `object`

`description:` A product

`required:`

`- reference`

`...`

`properties:`

`reference:`

`description:` Unique ID identifying a product

`type:` `string`

- ① Медиа тип тела ответа.
- ② JSON-схема тела ответа.
- ③ Тип ответа – массив.
- ④ Схема элементов массива.

Описание параметров тела

- Давайте посмотрим на действие «Добавить товар».
- Чтобы добавить товар в каталог, пользователь API должен предоставить какую-то информацию о товаре в теле своего запроса, как показано в примере.



post:

summary: Add product

description: Add product to catalog

requestBody: ①

description: Product's information ②

application/json: ③

schema: ④

[...]

responses:

- ① Определение параметра тела.
- ② Описание параметра тела.
- ③ Медиа тип параметра тела.
- ④ Схема параметра тела.

- Параметр тела HTTP-запроса описан в его свойстве `requestBody`.
- Как и тело ответа, параметр `body` имеет медиа тип `(application/json)`, а его содержимое описывается схемой JSON.
- Описание тела запроса и ответа выполняется одинаково.

```
requestBody:
  content:
    multipart/form-data: # Media type
      schema:             # Request payload
        type: object
        properties:       # Request parts
          id:              # Part 1 (string value)
            type: string
            format: uuid
          address:         # Part2 (object)
            type: object
            properties:
              street:
                type: string
              city:
                type: string
          profileImage:    # Part 3 (an image)
            type: string
            format: binary
```

Повторное использование компонентов

- Чтобы избежать двукратного описания JSON-схемы товара, нам нужно всего лишь объявить ее *схемой многократного использования*.
- Многократно используемые компоненты описаны в разделе `components`, который находится в корне документа OAS.
- Повторно используемые схемы определены в `schemas`; имя каждой такой схемы определяется как свойство `schemas`.
- Это свойство содержит повторно используемый компонент, схему JSON.


```
openapi: "3.0.0"
[...]
components: ①
  schemas: ②
    product: ③
      type: object ④
      description: A product
      required:
        - reference
        - name
        - price
        - supplier
      properties:
        reference: [...]
```

- Теперь вместо переопределения JSON-схемы товара мы можем использовать ссылку JSON для доступа к этой предопределенной схеме, когда она нам понадобится.
- *Ссылка JSON* – это свойство, имя которого – `$ref`, а содержимое – URL-адрес.
- Этот URL-адрес может указывать на любой компонент внутри документа или даже в других документах.
- Поскольку мы только ссылаемся на локальные компоненты, мы используем локальный URL-адрес, содержащий только фрагмент, описывающий путь к нужному элементу, как показано в примере.
- Здесь `product` находится в `schemas`, которые расположены в компонентах в корне документа.

Использование предопределенного компонента с ссылкой

```
post:
  summary: Add product
  description: Add product to catalog
  [...]
  responses:
    "200":
      description: Product added to catalog
      content:
        application/json:
          schema: ①
          $ref: "#/components/schemas/product" ②
```

① Схема ответа.

② Ссылка на предопределенную схему.

Использование предопределенного компонента в массиве

```
responses:
```

```
  "200":
```

```
    description: Products matching free query
```

```
    content:
```

```
      application/json:
```

```
        schema: ①
```

```
          type: array ②
```

```
          items: ③
```

```
            $ref:
```

```
              "#/components/schemas/product" ④
```

Описание параметров пути

- Ресурс-товар, который можно удалить, обновить или заменить, идентифицируется переменным путем.
- Обратите внимание, что действия, обозначенные глаголами *get*, *update* и *replace*, возвращают один и тот же товар, а также что действия *update* и *replace* используют один и тот же параметр.
- Также заметьте, что путь `/products/{productId}` содержит переменную `productId`, которая называется *параметром пути*.
- В приведенном ниже листинге показано, как можно определить это в нашем документе OAS для действия удаления.

```
paths:
  /products:
    [...]
  /products/{productId}: ①
    description: A product
    delete: ②
      summary: Delete a product
    parameters: ③
      - name: productId ④
        in: path ⑤
        required: true ⑥
        description: Product's reference
        schema:
          type: string
```

- ① Путь ресурса товара с параметром.
- ② Действие "Удалить товар".
- ③ Параметры действия "Удалить товар".
- ④ Имя параметра пути.
- ⑤ Параметр находится в пути.
- ⑥ Параметр является обязательным.

- Добавлен новый путь `/products/{productId}`, чтобы определить ресурс товара.
- Параметр пути обозначен фигурными скобками (`{productId}`) в `paths`.
- Затем определяем этот параметр пути в списке действия удаления `parameters`.
- Он определяется почти так же, как и любой другой параметр, который идет в разделе `parameters`: нам нужно установить значения для `name`, `location` и `schema`. `name` должно соответствовать имени внутри фигурных скобок в пути, поэтому установлен для него значение `productId`.
- Местоположение (`in`), очевидно, является путем, а тип этого параметра, определенный в его схеме, – строкой.
- Поскольку это параметр пути, необходимо сделать этот параметр обязательным, установив для `required` значение `true`.
- Если этого не сделать, парсер выкинет ошибку.

Описание многократно используемого параметра

```
components: ①
  parameters: ②
    productId: ③
    name: productId
    in: path
    required: true
    description: Product's reference
    schema:
      type: string
```

① Компоненты многократного использования.

② Параметры многократного использования.

③ Имя параметра многократного использования.

- В разделе `components` в корне документа OAS каждый повторно используемый параметр определяется как свойство `parameters` и идентифицируется по имени.
- Свойство `productId` содержит определение параметра пути `productId`, как мы его определили для DELETE `/products/{productId}`.

Использование predeterminedного параметра

paths:

/products:

[...]

/products/{productId}: ②

delete:

parameters:

- \$ref: #/components/parameters/productid ③

[...]

- ① Определение параметра пути.
- ② Путь ресурса товара с параметром.
- ③ Ссылка на predeterminedный параметр.

- Строго говоря, параметр `productId` не является параметром действия; это параметр ресурса.
- В документе OAS параметры могут определяться не только на уровне действий, но и на уровне ресурсов, опять же в разделе `parameters`.
- Структура этого раздела точно такая же, как и на уровне действий.
- Все параметры, определенные на уровне ресурса, применимы ко всем действиям на ресурсе.
- Определить параметр пути `productId` в разделе `parameters` пути `/products/{productId}`.

Параметры уровня ресурса

```
components:
```

```
  parameters:
```

```
    productId: ①
```

```
    [...]
```

```
paths:
```

```
  /products:
```

```
  [...]
```

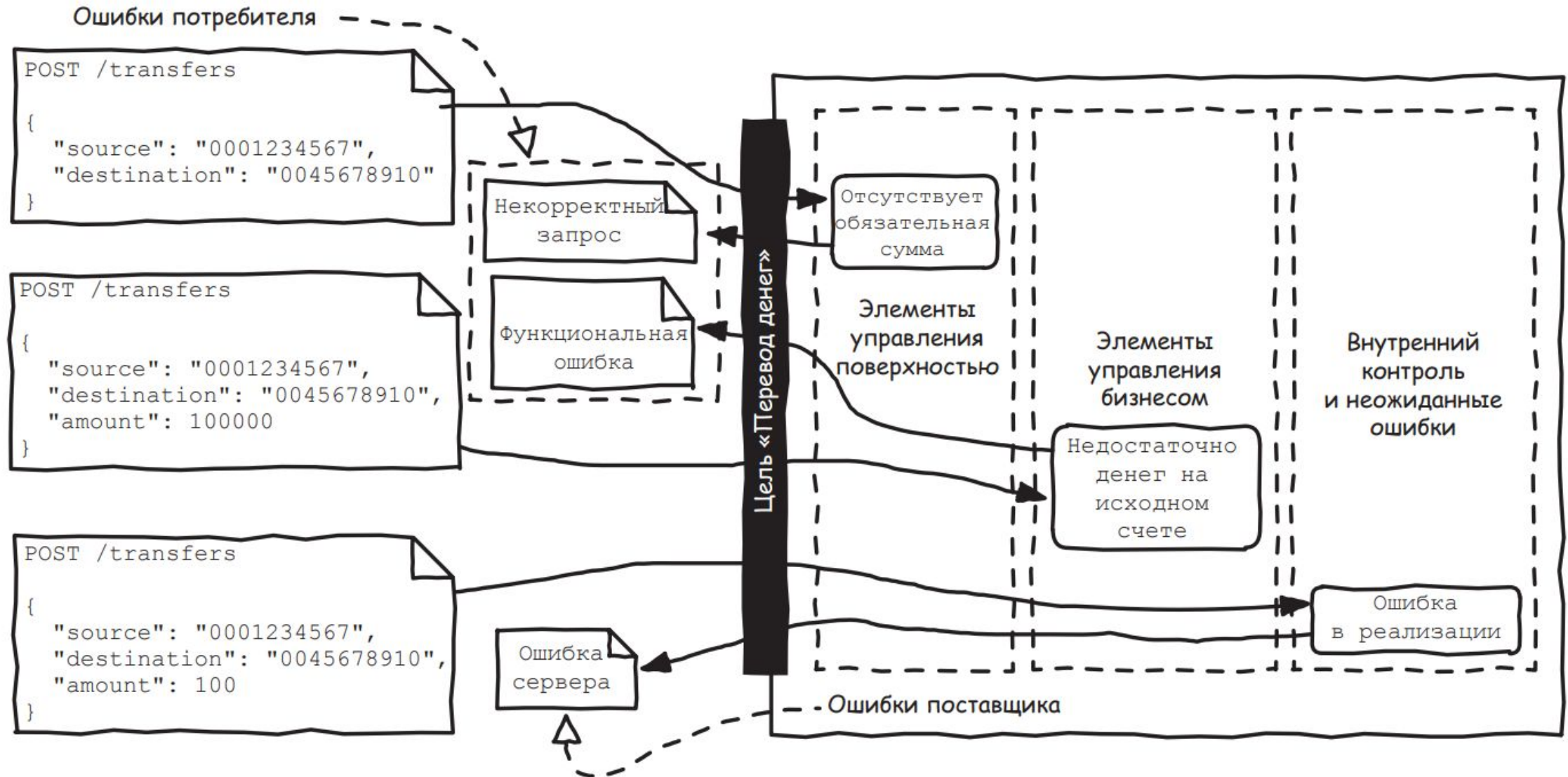
```
  /products/{productId}:
```

```
    parameters: ②
```

```
      - $ref: #/components/parameters/productId ③
```

- ① Определение параметра пути.
- ② Параметры уровня ресурса.
- ③ Ссылка на предопределенный параметр.
- ④ Определений параметров пути больше нета.

Немного об ошЫбках



Ошибки потребителя коды состояния 4XX

```
POST /transfers
{
  "source": "0001234567",
  "destination": "0045678910"
}
```

400 Bad Request

```
POST /transfers
{
  "source": "0001234567",
  "destination": "0045678910",
  "amount": 100000
}
```

403 Forbidden

```
POST /transfers
{
  "source": "0001234567",
  "destination": "0045678910",
  "amount": 100
}
```

500 Internal Server Error

```
POST /transfers
{
  "source": "0001234567",
  "destination": "0045678910",
  "amount": 100
}
```

200 OK

Цель «Перевод денег»

Отсутствует
обязательная
сумма

Элементы
управления
поверхностью

Элементы
управления
бизнесом

Недостаточно
денег на
исходном
счете

Внутренний
контроль
и неожиданные
ошибки

Неожиданная
ошибка
сервера

Коды состояния ошибок поставщика 5XX

Все прошло гладко

- Подробное сообщение об ошибке с использованием универсального типа

```
{  
  "source": "amount",  
  "type": "MISSING_MANDATORY_PROPERTY",  
  "message": "Amount is mandatory"  
}
```

- Возвращение нескольких ошибок

```
{  
  "message": "Invalid request",  
  "errors": [  
    {  
      "source": "source",  
      "type": "MISSING_MANDATORY_PROPERTY",  
      "message": "Source is mandatory" },  
    {  
      "source": "destination",  
      "type": "MISSING_MANDATORY_PROPERTY"},  
      "message": "Destination is mandatory"  
    }  
  ]  
}
```

Коды состояния HTTP – некорректные запросы и фун

Случай использования	Пример	Код состояния HTTP
Неправильный параметр пути	Чтение несуществующего счета с помощью запроса GET /accounts/123	404 Not Found
Отсутствует обязательное свойство	Не указана сумма	400 Bad Request
Неверный тип данных	"startDate":1423332060	400 Bad Request
Функциональная ошибка	Сумма превышает безопасный лимит	403 Forbidden
Функциональная ошибка	Перевод из source в destination запрещен	403 Forbidden
Функциональная ошибка	За последние пять минут уже был выполнен идентичный денежный перевод	409 Conflict
Неожиданная ошибка сервера	Ошибка в реализации	500 Internal Server Error