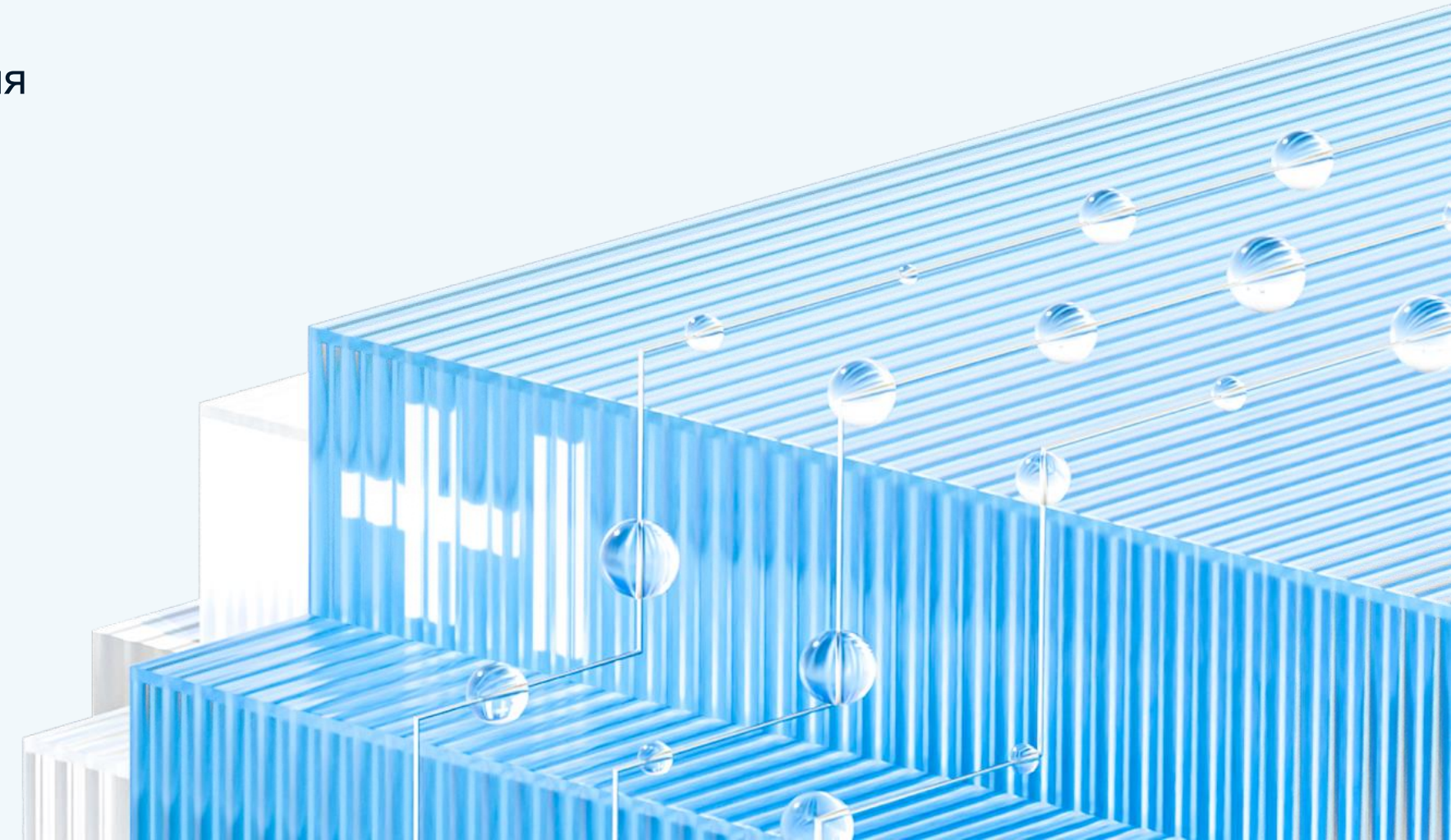
21.11.2025

REST API

Основы архитектурного стиля



ITI

Аня Баранова

Ведущий системный аналитик

- Участвовала в реализации первого в России сервиса по страхованию животных
- Работала в маленьких и больших нефтяных компаниях от инженера до аналитика
- Создавала процессы для системы «Честный знак»

Привет, коллеги!

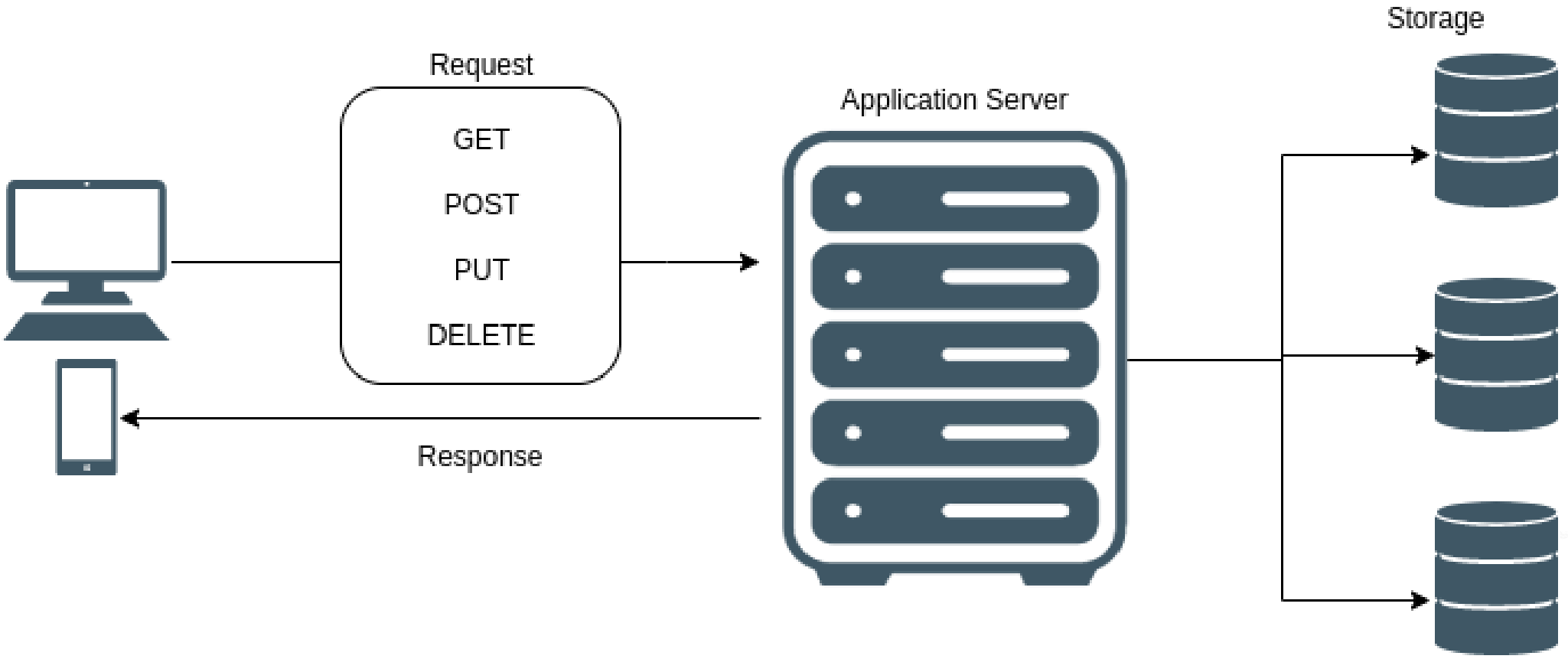
Содержание

REST API

REST API (Representational State Transfer Application Programming Interface) — это архитектурный стиль для создания веб-сервисов, который использует стандартные HTTP-методы и принципы для взаимодействия между клиентом и сервером через ресурсы (данные и функциональность).

Кто и когда создал?

1. REST был описан в 2000 году в докторской диссертации Роя Филдинга (Roy Fielding), одного из основных архитекторов протокола HTTP.
2. Изначально REST задумывался как универсальный архитектурный стиль для распределённых систем, основанный на принципах, выявленных при разработке HTTP.


Storage
Request
GET
POST
PUT
DELETE
Application Server
Response

ТОЛЬКО ЛИ HTTP?

Примеры альтернативных протоколов для RESTful API:

- **Пример REST-подобного взаимодействия через WebSockets:**

```
// Клиент отправляет запрос в формате JSON
{
  "method": "GET",
  "resource": "/users/123",
  "requestId": "abc-123"
}

// Сервер обрабатывает и возвращает ответ
{
  "status": 200,
  "body": {"id": 123, "name": "Alice"},
  "inResponseTo": "abc-123"
}
```

ВИДЫ ВЗАИМОДЕЙСТВИЯ

СИНХРОННЫЙ ЗАПРОС

Пример процесса синхронного запроса:

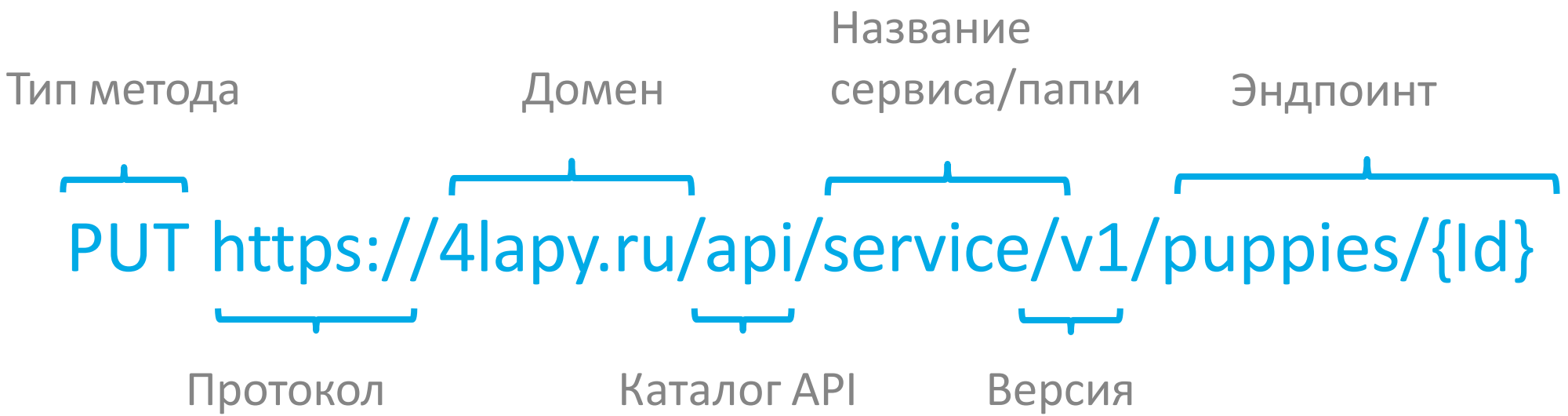
- Клиент отправляет запрос
- Сервер принимает запрос, обрабатывает его.
- Сервер возвращает полный ответ (например, данные ресурса или подтверждение операции).
- Клиент продолжает выполнение на основе полученного результата.

АСИНХРОННЫЙ ЗАПРОС

- Клиент отправляет запрос, инициирующий долгую операцию, например, загрузку файла, обработку данных, генерацию отчёта
- Сервер принимает запрос, создаёт задачу и возвращает клиенту идентификатор задачи и статус
- Клиент периодически опрашивает статус задачи
- Когда задача завершается, сервер возвращает статус и может предоставить результат
- Клиент запрашивает результат (при необходимости):

ПРИНЦИПЫ И ОГРАНИЧЕНИЯ

- Клиент-серверная архитектура (Client-Server)
- Отсутствие состояния (Stateless)
- Кэширование (Cacheable)
- Единый интерфейс (Uniform Interface)
  (Ресурсы и URI)
- Слои (Layered System)
- Код по требованию (Code on Demand)



{Id} – path параметр

СТРУКТУРА

ВЕРСИОНИРОВАНИЕ

ОПРЕДЕЛЕНИЕ И ИСПОЛЬЗОВАНИЕ

Версионирование в REST API — это практика управления изменениями и обновлениями интерфейса API таким образом, чтобы новые функции и улучшения могли внедряться, не нарушая работу существующих клиентов. Основная цель — обеспечить обратную совместимость и избежать конфликтов при эволюции API.

Основные способы:

- **Версионирование в URL:** /v1/users или /api/v2/orders
- **Версионирование через заголовки:** Accept: application/vnd.myapi.v1+json или кастомный заголовок X-API-Version: 2
- **Версионирование через параметры запроса:** /users?version=1 или /api/resource?api-version=2
- **Контентное :** application/vnd.example.v1+json)

ОБРАТНАЯ СОВМЕСТИМОСТЬ

ОПРЕДЕЛЕНИЕ И РЕКОМЕНДАЦИИ ПО ПОДДЕРЖАНИЮ

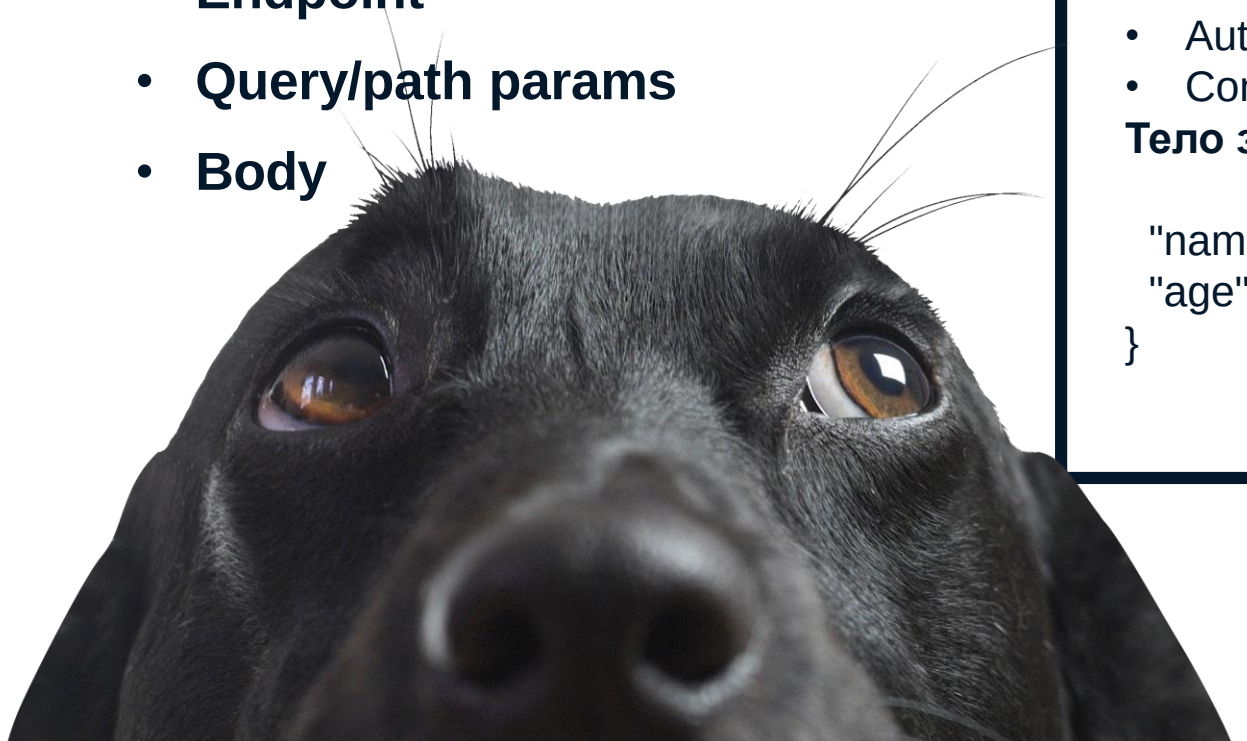
Обратная совместимость (backward compatibility) в REST API — это способность системы взаимодействовать с предыдущими версиями API, данных или программного обеспечения. Например, новое обновление сервиса должно работать с клиентами, которые используют старую версию API

Рекомендации по работе с версиями API

- Планируйте изменения так, чтобы минимизировать влияние на существующие интеграции.
- Поддерживайте обратную совместимость как можно дольше, добавляя новые поля или параметры без удаления существующих.
- Информируйте клиентов об изменениях заранее, указывая сроки перехода на новые версии.
- Создавайте документацию для каждой версии API, чтобы упростить интеграцию для разработчиков клиентских приложений.
- Используйте мониторинг, чтобы отслеживать использование устаревших версий и планировать их вывод из эксплуатации.

СТРУКТУРА ЗАПРОСА

- **Header**
- **Cookie**
- **URL**
- **Метод**
- **Endpoint**
- **Query/path params**
- **Body**



Стартовая строка (Request Line): GET /api/puppies HTTP/1.1

- Метод (GET, POST, PUT, DELETE и др.) — что мы хотим сделать
- Путь (/api/users) — адрес ресурса на сервере.
- Версия протокола (HTTP/1.1) — используемая версия HTTP

Заголовки (Headers)

- Host: example.com
- User-Agent: Mozilla/5.0
- Authorization: Bearer abc123...
- Content-Type: application/json

Тело запроса (Body)

```
  "name": “Бруно",
  "age": 14
}
```

СТРУКТУРА ОТВЕТА

- Header
- Cookie
- Body
- Код HTTP

Статусная строка (Status Line)
HTTP/1.1 200 OK
- Версия протокола (HTTP/1.1).
- Код статуса (200) — значение, которое говорит об успехе или ошибке
- Текст статуса (OK) — пояснение к коду.

Заголовки ответа (Headers)
- Content-Type — тип данных в теле ответа.
- Server — какое ПО работает на сервере.
- Set-Cookie — куки, которые сервер просит сохранить

Тело ответа (Body)

```
{
  "id": 1,
  "name": “Бруно",
  “age": 14
}
```

QUERY PARAMETRS

ОПРЕДЕЛЕНИЕ И ПРИМЕРЫ

Query Parameters (параметры запроса) — это часть URL, которая позволяет передавать дополнительные данные серверу для фильтрации, сортировки, пагинации и других операций. Они добавляются в конец URL после знака вопроса ? и разделяются амперсандами &.

Пагинация

GET /puppies?page=2&per_page=20

Фильтрация данных

GET /puppies?breed=yorkshire

Сортировка

GET / puppies?sort=age_asc

Поиск

GET / puppies?search=query

Авторизация и аутентификация

GET / puppies?token=abc123

POST

PUT

GET

DELETE

HEAD

PATCH

OPTIONS

CRUD (Create, Read, Update, Delete) — это аббревиатура, которая отражает четыре основные функции, доступные для манипулирования данными в системе

МЕТОДЫ HTTP

СВОЙСТВА HTTP МЕТОДОВ

МЕТОД	БЕЗОПАСНОСТЬ	КЭШИРУЕМОСТЬ	ИДЕМПОТЕНТНОСТЬ	ТЕЛО ЗАПРОСА
GET	ДА	ДА	ДА	НЕТ
POST	НЕТ	ДА	НЕТ	ДА
PUT	НЕТ	НЕТ	ДА	ДА
PATCH	НЕТ	НЕТ	НЕТ	ДА
DELETE	НЕТ	НЕТ	ДА	НЕТ

КОДЫ HTTP ОТВЕТОВ

1xx: Информационные (100–199)
2xx: Успешные (200–299)
3xx: Перенаправления (300–399)
4xx: Ошибки клиента (400–499)
5xx: Ошибки сервера (500–599)

КОДЫ HTTP ОТВЕТОВ

САМЫЕ ПОПУЛЯРНЫЕ И ЧАСТО ИСПОЛЬЗУЕМЫЕ КОДЫ HTTP ОТВЕТОВ

- 200 OK
- 201 Создано
- 204 Нет контента
- 206 Частичное содержимое
- 400 Плохой запрос
- 401 Не авторизован
- 403 Доступ запрещен
- 404 Не найдено
- 409 Конфликт
- 429 Слишком много запросов
- 500 Внутренняя ошибка сервера
- 502 Плохой шлюз
- 503 Сервер не доступен
- 504 Таймаут шлюза

GET https://4lapy.ru/api/products/v1/puppies/{meet_id}

- Как 404?

JSON VS XML?

JSON

- Легче и компактнее
- Проще читать и разбирать
- Быстрее парсится
- Стандарт де-факто для REST

XML

- Сложные документные структуры
- Богатые схемы
- Наследие и стандарты

Bruno

Collection Edit View Window Help

bruno

Collections

Search requests ...

Sample API Collection

bruno

Opensource IDE for exploring and testing APIs

COLLECTIONS

Create Collection Open Collection Import Collection

https://www.usebruno.com/downloads

LINKS

Documentation

Report Issues

GitHub

Press ⌘ K (mac) or Ctrl K (windows) anytime to quickly search collections, folders, and requests

Search Cookies Dev Tools v2.14.2

Основные угрозы безопасности API

ИСТОЧНИКИ УГРОЗ

1. Неавторизованный доступ к API

Если API не имеет надежной системы аутентификации и авторизации, злоумышленники могут получить доступ к конфиденциальным данным и критически важным функциям.

2. Утечки данных из-за недостаточной валидации запросов

Некорректно обработанные входные данные могут привести к утечкам информации, SQL-инъекциям и другим атакам, направленным на получение конфиденциальных сведений.

3. Атаки типа (MITM)

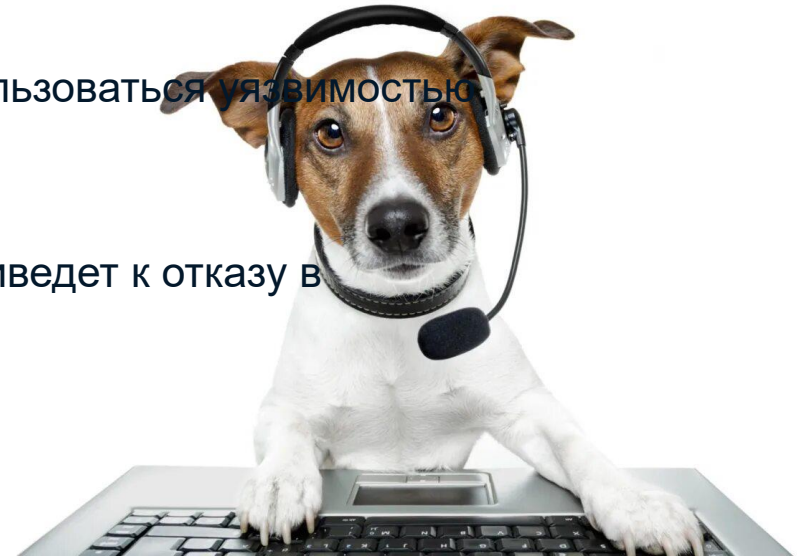
Без надежного шифрования злоумышленники могут перехватить данные, передаваемые между клиентом и сервером API, включая учетные данные пользователей и другие критически важные данные.

4. Атаки с повышением привилегий

Если API не проверяет уровень доступа пользователей, атакующий может воспользоваться уязвимостью для получения административных прав или доступа к чужим данным.

5. DDoS-атаки и чрезмерные запросы

Злоумышленники могут перегрузить API большим количеством запросов, что приведет к отказу в обслуживании и недоступности сервисов.



СПАСИБО ЗА ВНИМАНИЕ!