Раздел 0×07: REST и промисы

План действий

загрузка информации с сервера, её обработка
и использование в приложении

План действий

загрузка информации с сервера, её обработка
и использование в приложении

- узнаем что такое REST

План действий

загрузка информации с сервера, её обработка
и использование в приложении

- узнаем что такое REST
- скачаем информацию с сервера используя REST

План действий

загрузка информации с сервера, её обработка
и использование в приложении

- узнаем что такое REST
- скачаем информацию с сервера используя REST
- узнаем новый способ загрузки данных и заодно познакомимся с технологией Promise

План действий

загрузка информации с сервера, её обработка
и использование в приложении

- узнаем что такое REST

- скачаем информацию с сервера используя REST

- узнаем новый способ загрузки данных и заодно познакомимся с технологией Promise

- разберемся, зачем все-таки нужна модель

План действий

загрузка информации с сервера, её обработка
и использование в приложении

- узнаем что такое REST
- скачаем информацию с сервера используя REST
- узнаем новый способ загрузки данных и заодно познакомимся с технологией Promise
- разберемся, зачем все-таки нужна модель
- познакомимся с паттерном «Адаптер»

План действий

загрузка информации с сервера, её обработка
и использование в приложении

- узнаем что такое REST
- скачаем информацию с сервера используя REST
- узнаем новый способ загрузки данных и заодно познакомимся с технологией Promise
- разберемся, зачем все-таки нужна модель
- познакомимся с паттерном «Адаптер»
- отправим информацию по REST

HTTP 1.1

протокол передачи данных. Описывает в каком виде обмениваются информацией клиент и сервер

HTTP 1.1

HTTP 1.1

- строка запроса / строка статуса
  `GET localhost:8080 HTTP/1.1`
  `HTTP/1.1 200 OK`

HTTP 1.1

- строка запроса / строка статуса
  `GET localhost:8080 HTTP/1.1`
  `HTTP/1.1 200 OK`

- строки заголовков в виде пар ключ-значение
  `Content-Type: application/json`

HTTP 1.1

- строка запроса / строка статуса
  `GET localhost:8080 HTTP/1.1`
  `HTTP/1.1 200 OK`

- строки заголовков в виде пар ключ-значение
  `Content-Type: application/json`

- пустая строка

HTTP 1.1

- строка запроса / строка статуса
  `GET localhost:8080 HTTP/1.1`
  `HTTP/1.1 200 OK`

- строки заголовков в виде пар ключ-значение
  `Content-Type: application/json`

- пустая строка

- тело

HTTP методы чтения

HTTP методы чтения

- GET — запрос на получение информации с сервера

HTTP методы чтения

- `GET` — запрос на получение информации с сервера
- `HEAD` — запрос для проверки, обновилась
  ли информация на сервере и стоит ли заново ее скачать
  или можно оставить закешированную версию

HTTP методы чтения

- `GET` — запрос на получение информации с сервера
- `HEAD` — запрос для проверки, обновилась ли информация на сервере и стоит ли заново ее скачать или можно оставить закешированную версию
- `OPTIONS` — запрос для проверки, какие запросы можно делать на этот ресурс

HTTP методы записи

HTTP методы записи

- POST — запрос на создание новой записи на сервере

HTTP методы записи

- POST — запрос на создание новой записи на сервере
- PUT — запрос на перезапись существующей информации на сервере

HTTP методы записи

- POST — запрос на создание новой записи на сервере
- PUT — запрос на перезапись существующей информации на сервере
- PATCH — запрос на частичную перезапись существующей информации на сервере

REST. Начало

REST —

Representational **S**tate **T**ransfer *(крымскотат. — передача состояния приложения)*. Стиль взаимодействия компонент распределенного приложения в сети

REST

общение с сервером

REST

Каждой сущности на сервере в соответствие ставится URI адрес. Управление этим ресурсом осуществляется отправкой заголовка

REST

Каждой сущности на сервере в соответствие ставится URI адрес. Управление этим ресурсом осуществляется отправкой заголовка

- ресурсы

REST

Каждой сущности на сервере в соответствие ставится URI адрес. Управление этим ресурсом осуществляется отправкой заголовка

- ресурсы

- управляющие глаголы

REST

Каждой сущности на сервере в соответствие ставится URI адрес. Управление этим ресурсом осуществляется отправкой заголовка

- ресурсы

- управляющие глаголы
  - GET — *получение*

REST

Каждой сущности на сервере в соответствие ставится URI адрес. Управление этим ресурсом осуществляется отправкой заголовка

- ресурсы

- управляющие глаголы
  - GET — *получение*
  - POST — *создание*

REST

Каждой сущности на сервере в соответствие ставится URI адрес. Управление этим ресурсом осуществляется отправкой заголовка

- ресурсы

- управляющие глаголы
  - GET — *получение*
  - POST — *создание*
  - PUT — *обновление*

REST

Каждой сущности на сервере в соответствие ставится URI адрес. Управление этим ресурсом осуществляется отправкой заголовка

- ресурсы

- управляющие глаголы
  - GET — *получение*
  - POST — *создание*
  - PUT — *обновление*
  - DELETE — *удаление*

REST

Сайт

Мобильное приложение

клиент

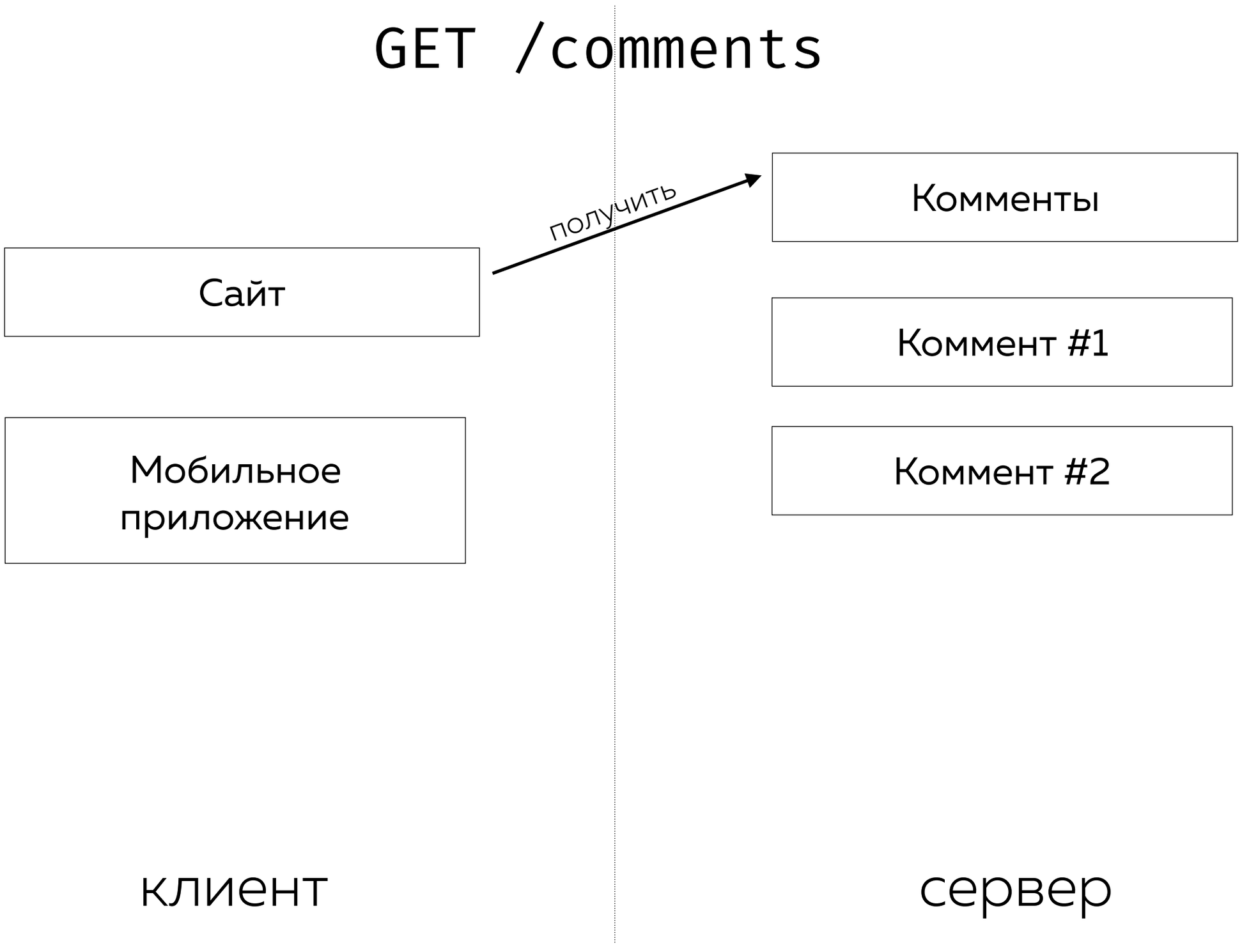
Комменты

Коммент #1

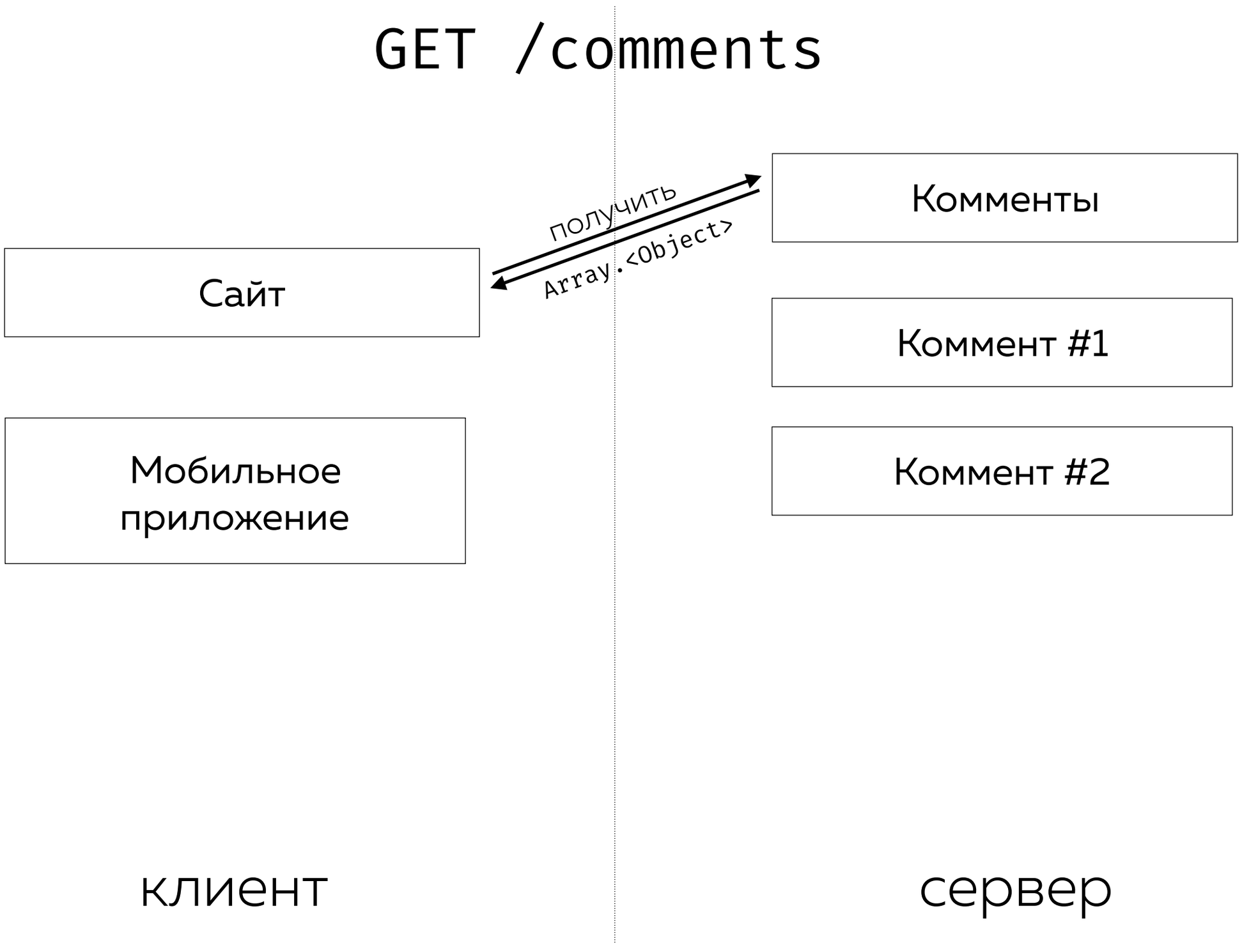
Коммент #2

сервер

REST


GET /comments
Сайт
получить
Комментары
Коммент #1
Коммент #2
Мобильное приложение
клиент
сервер

REST


GET /comments
Сайт
получить
Array.<Object>
Комментары
Коммент #1
Коммент #2
Мобильное приложение
клиент
сервер

REST

Сайт

Мобильное приложение

клиент

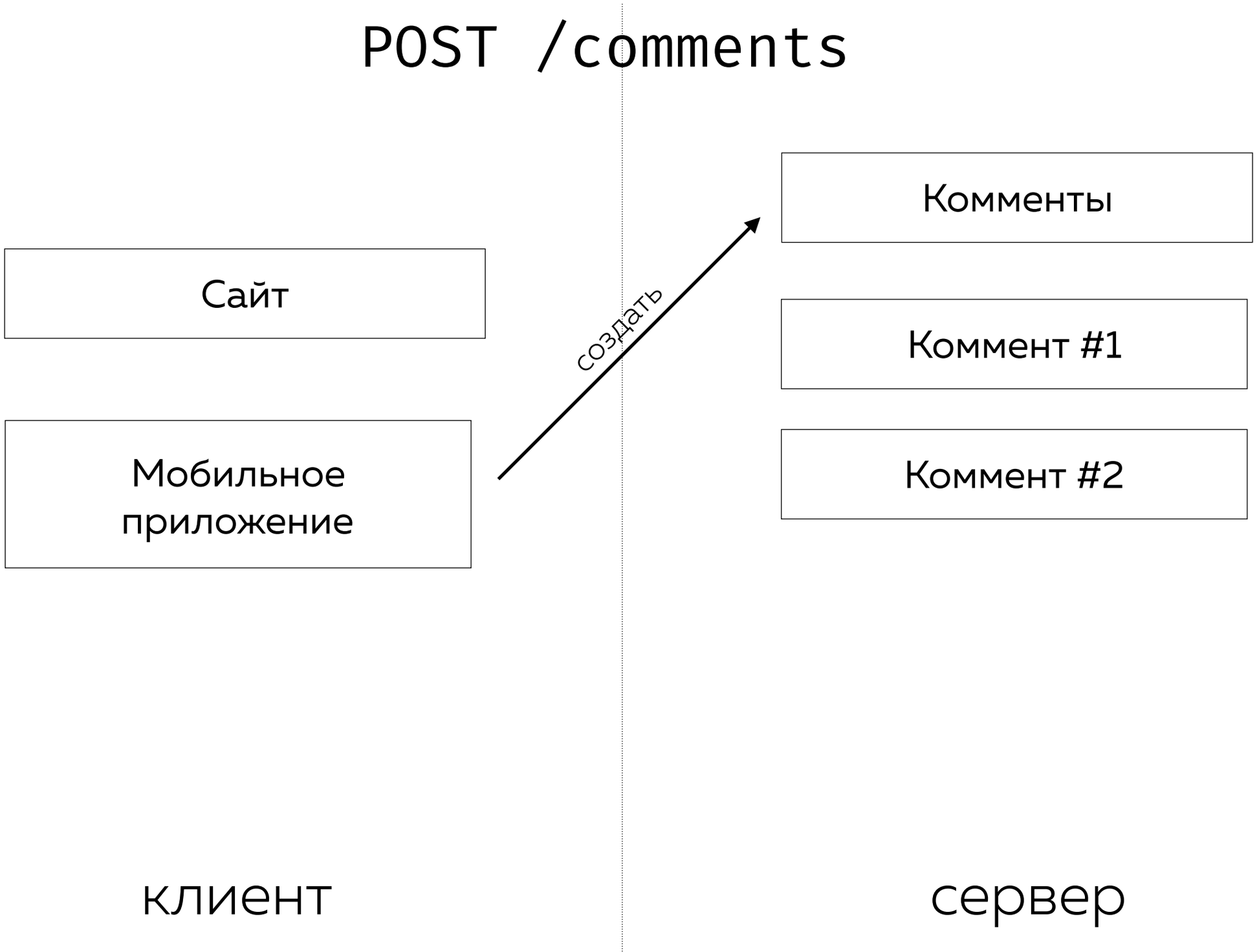
Комменты

Коммент #1

Коммент #2

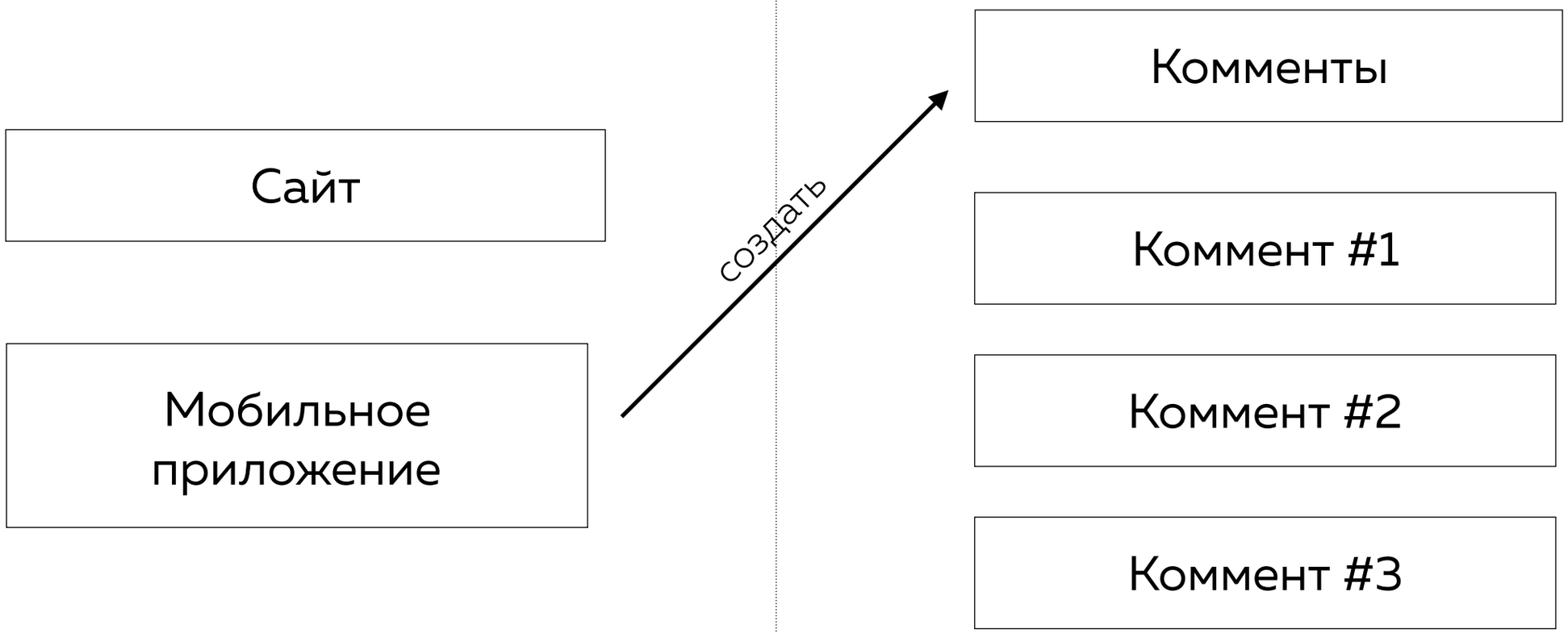
сервер

REST


POST /comments
Сайт
Мобильное приложение
создать
Комменты
Коммент #1
Коммент #2
клиент
сервер

REST

POST /comments

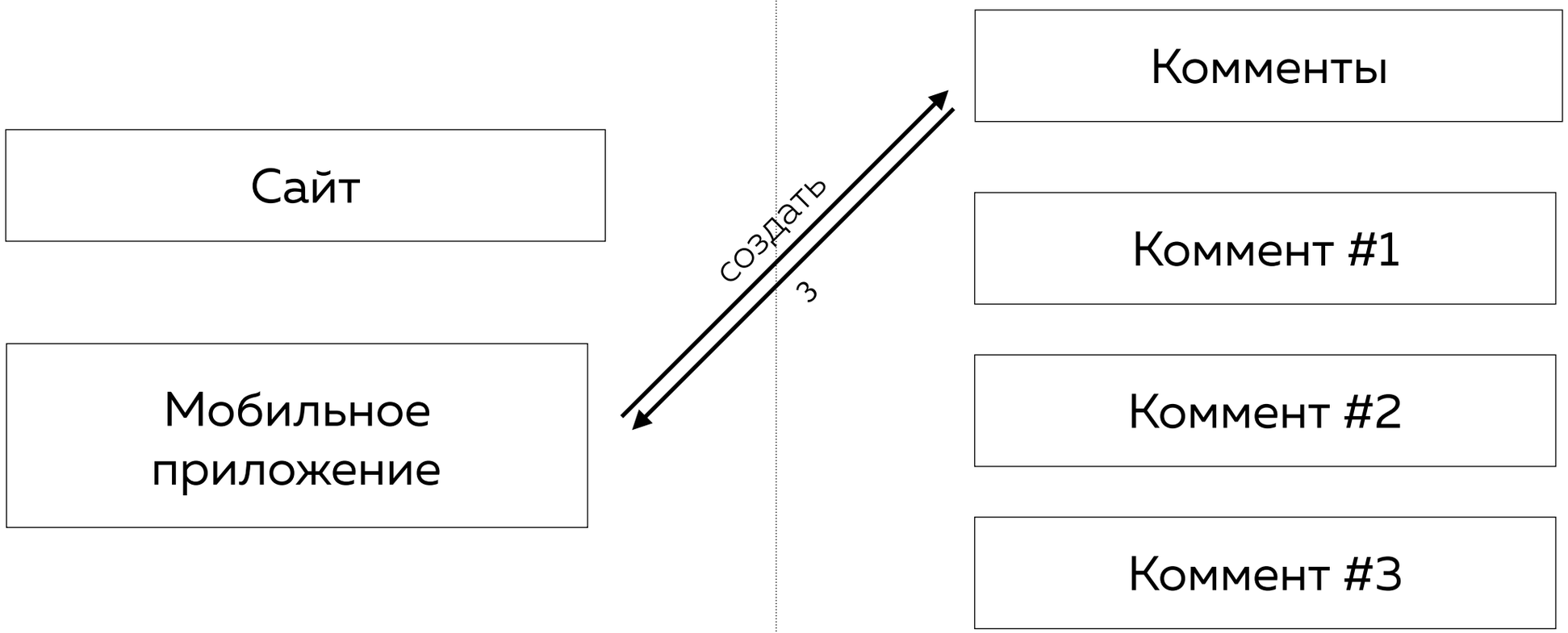


клиент

сервер

REST

POST /comments



клиент

сервер

REST

Сайт

Мобильное приложение

клиент

Комменты

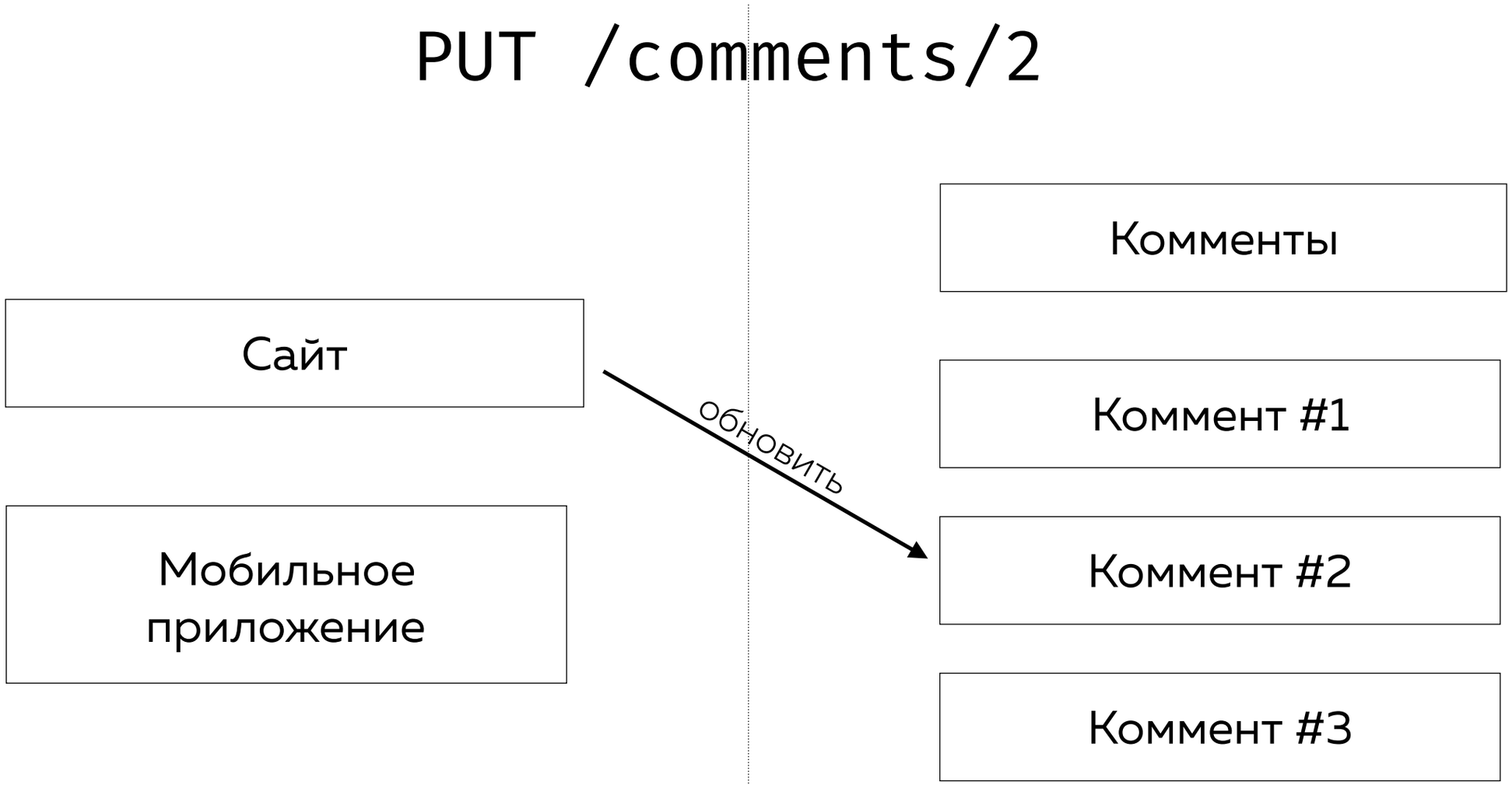
Коммент #1

Коммент #2

Коммент #3

сервер

REST



клиент

сервер

REST

Сайт

Мобильное приложение

клиент

Комменты

Коммент #1

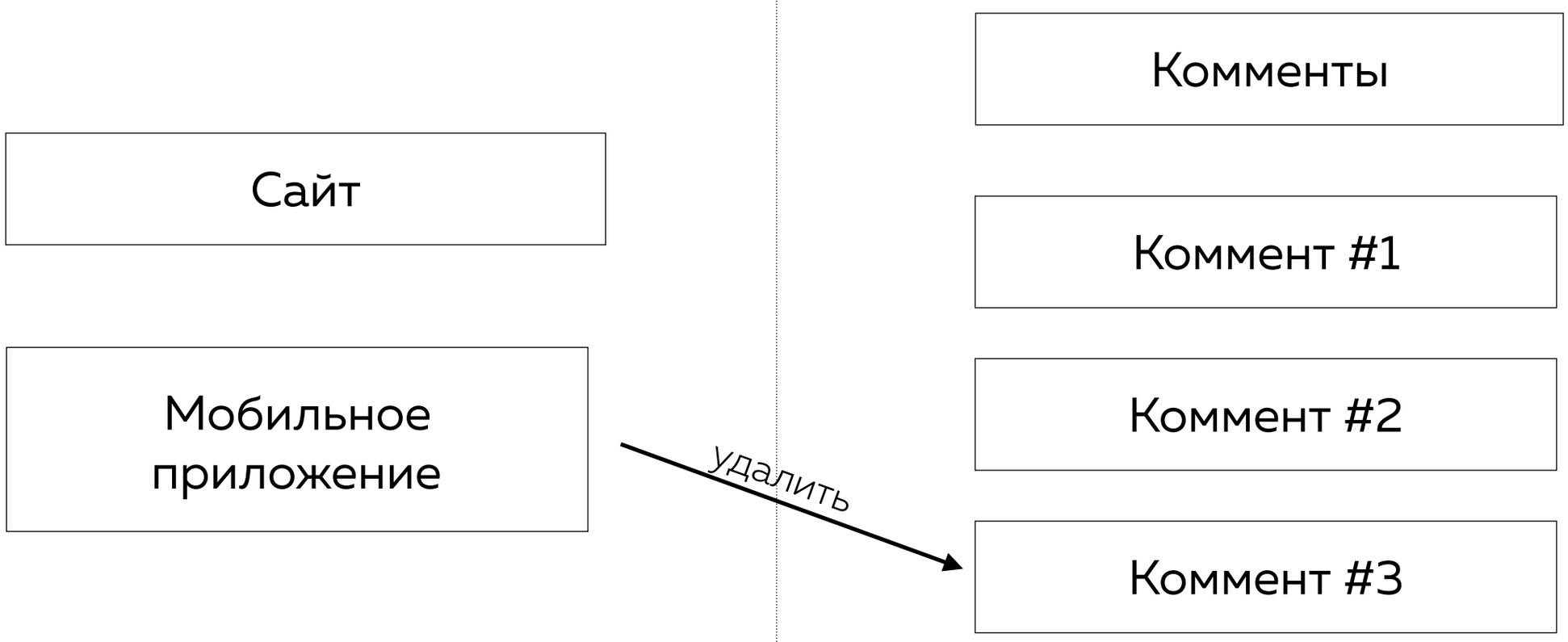
Коммент #2

Коммент #3

сервер

REST

```
DELETE /comments/3
```



клиент

сервер

REST

Сайт

Мобильное приложение

клиент

Комменты

Коммент #1

Коммент #2

сервер

fetch и Promise

немного об обработке асинхронного кода

Синхронная загрузка данных

```
const getResponse = (url) ⇒ {
  const xhr = new XMLHttpRequest();
  xhr.open('GET', url, false);
  xhr.send();
  return xhr.response;
};

const data = getResponse('data.json');
```

Асинхронная загрузка

```
const getResponse = (url, onload) ⇒ {
  const xhr = new XMLHttpRequest();
  xhr.open('GET', url);
  xhr.onload = (evt) ⇒
      onload(evt.target.response);
  xhr.send();
};

getResponse('data.json', (response) ⇒ {
  console.log(response);
});
```

Асинхронная обработка ошибок

```
const getResponse = (url, onload, onerror) ⇒ {
  const xhr = new XMLHttpRequest();
  xhr.open('GET', url);
  xhr.onload = (evt) ⇒
      onload(evt.target.response);
  xhr.onerror = (evt) ⇒
      onerror(evt.target.status);
  xhr.send();
};

getResponse('data.json', (response) ⇒ {
  console.log(response);
});
```

Асинхронная обработка ошибок

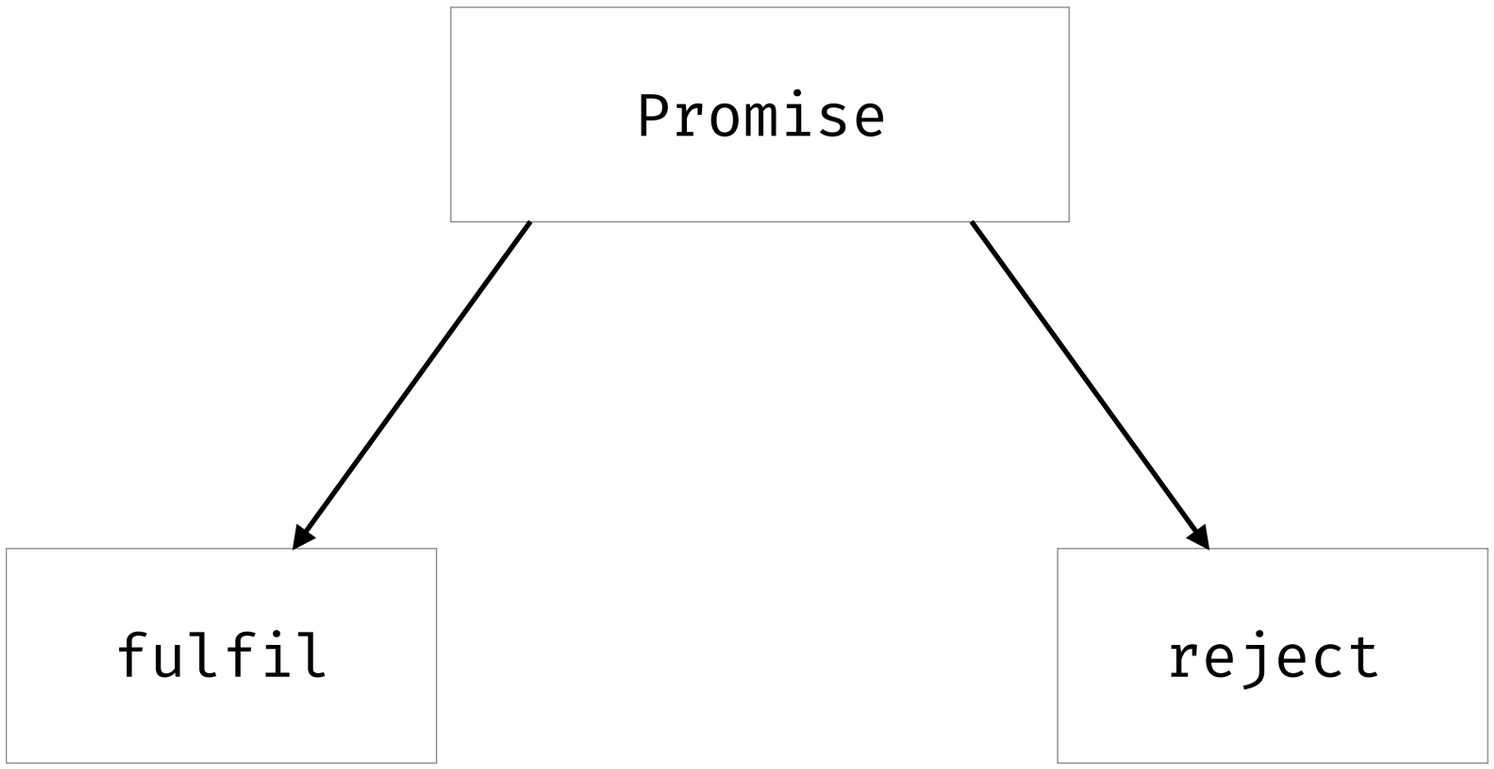
```
const getResponse = (url, onload, onerror) ⇒ {
  const xhr = new XMLHttpRequest();
  xhr.open('GET', url);
  xhr.onload = (evt) ⇒
      onload(evt.target.response);
  xhr.onerror = (evt) ⇒
      onerror(evt.target.status);
  xhr.send();
};

getResponse('data.json', (response) ⇒ {
  console.log(response);
}, (errorStatus) ⇒ {
  console.warn(`Error #${errorStatus} happened`);
});
```

Promise

способ написания асинхронного кода. Promise возвращает объект, замещающий возвращаемое значение, которое на момент завершения функции еще не известно

Promise



Promise. Успех

```
const getResponse = (url) ⇒ new Promise();

getResponse('data.json');
```

Promise. Успех

```
const getResponse = (url) ⇒ new Promise(
  (resolve, reject) ⇒ {

  }
);
```

```
getResponse('data.json');
```

Promise

создается с помощью конструктора `Promise`.
Принимает на вход функцию с двумя коллбэками — обработчик успеха и обработчик неудачи

Promise. Успех

```
const getResponse = (url) ⇒ new Promise(
  (resolve, reject) ⇒ {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url);

    xhr.onload = (evt) ⇒
      resolve(evt.target.response);
    xhr.send();
  }
);

getResponse('data.json');
```

Promise

в случае успешного завершения асинхронного действия вызывается функция `resolve`, которая передается как первый коллбэк

Promise. Успех

```
const getResponse = (url) ⇒ new Promise(
  (resolve, reject) ⇒ {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url);

    xhr.onload = (evt) ⇒
        resolve(evt.target.response);
    xhr.send();
  }
);

getResponse('data.json').
    then((data) ⇒ console.log(data));
```

Promise

возвращает объект, с двумя методами, принимающими на вход коллбэки. Метод `then` принимает на вход коллбэк успеха

Promise

в случае неудачного завершения асинхронного действия вызывается функция `reject`, которая передается как второй коллбэк в конструктор `Promise`

Promise. Обработка ошибки

```
const getResponse = (url) ⇒ new Promise(
  (resolve, reject) ⇒ {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url);

    xhr.onload = (evt) ⇒
        resolve(evt.target.response);
    xhr.onerror = () ⇒ reject('Error');
    xhr.send();
  }
);

getResponse('data.json').
    then((data) ⇒ console.log(data));
```

Promise

Обработка ошибок может производиться двумя способами. Первый — через второй коллбэк метода `then`

Promise. Обработка ошибки

```
const getResponse = (url) ⇒ new Promise(
  (resolve, reject) ⇒ {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url);

    xhr.onload = (evt) ⇒
        resolve(evt.target.response);
    xhr.onerror = () ⇒ reject('Error');
    xhr.send();
  }
);

getResponse('data.json').then(
    (data) ⇒ console.log(data),
    (error) ⇒ console.warn(error));
```

Promise

Второй способ — использование метода `catch`, объекта, созданного через `Promise`

Promise. Обработка ошибки

```
const getResponse = (url) ⇒ new Promise(
  (resolve, reject) ⇒ {
    const xhr = new XMLHttpRequest();
    xhr.open('GET', url);

    xhr.onload = (evt) ⇒
        resolve(evt.target.response);
    xhr.onerror = () ⇒ reject('Error');
    xhr.send();
  }
);

getResponse('data.json').
    then((data) ⇒ console.log(data)).
    catch((error) ⇒ console.warn(error));
```

Promise

Обработка может производиться цепочками методов. Любой из вызовов `then` возвращает Promise, у которого так же можно вызвать `then`

Цепочки обработки Promise

Цепочки обработки Promise

```
Promise.resolve('a').
```

```
"a"
```

Цепочки обработки Promise

```
Promise.resolve('a').
    then((val) ⇒ val.concat('b')).
```

```
"a"
"ab"
```

Цепочки обработки Promise

```
Promise.resolve('a').
    then((val) ⇒ val.concat('b')).
    then((val) ⇒ val.concat('c')).
```

```
"a"
"ab"
"abc"
```

Цепочки обработки Promise

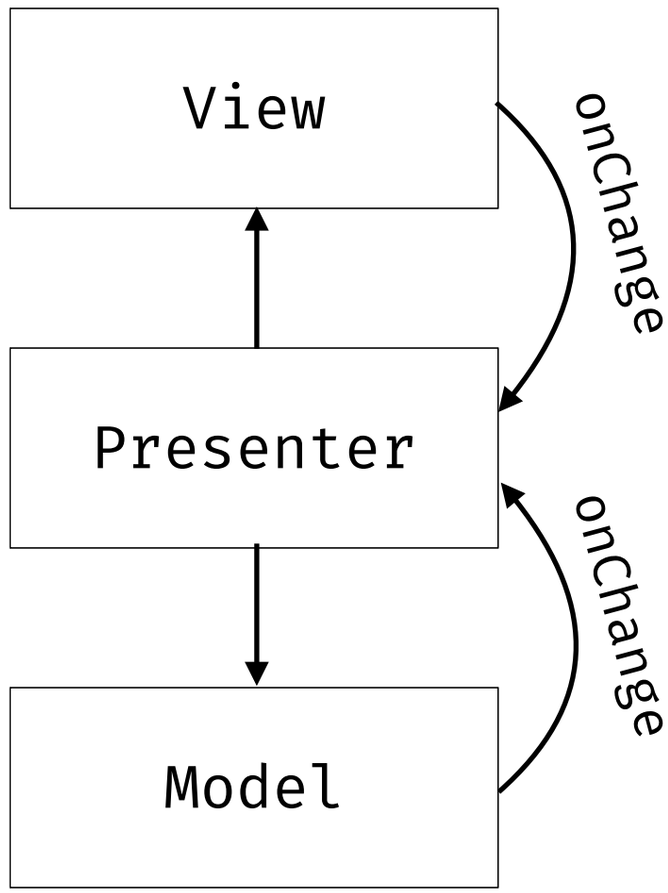
```
Promise.resolve('a').                    "a"
    then((val) ⇒ val.concat('b')).       "ab"
    then((val) ⇒ val.concat('c')).       "abc"
    then((val) ⇒ val.concat('d'));       "abcd"
```

MVP

- **Model** — работает только с данными. Слабо связанный компонент

- **View** — отвечает строго за отображение. Слабо связанный компонент

- **Presenter** — «связывает» между собой View и Model. Все взаимодействия производятся только через него



fetch

отправка данных

fetch

fetch

```
fetch(`http://localhost:8080`, {
```

fetch

```
fetch(`http://localhost:8080`, {
  method: `POST`,
```

fetch

```
fetch(`http://localhost:8080`, {
  method: `POST`,
  body: JSON.stringify({
    `date`: Date.now(),
    `time`: 402,
    `lives`: 3
  }),
```

fetch

```
fetch(`http://localhost:8080`, {
  method: `POST`,
  body: JSON.stringify({
    `date`: Date.now(),
    `time`: 402,
    `lives`: 3
  }),
  headers: {
    `Content-Type`: `application/json`
  }
});
```

body

body

- Строка — JSON в виде строки

body

- Строка — JSON в виде строки
- `FormData` — специальный объект, который позволяет конструировать запросы в виде данных из формы

body

- Строка — JSON в виде строки
- `FormData` — специальный объект, который позволяет конструировать запросы в виде данных из формы
- `URLSearchParams` — объект, представляющий собой строку поиска HTTP запроса

`body`

- Строка — JSON в виде строки
- `FormData` — специальный объект, который позволяет конструировать запросы в виде данных из формы
- `URLSearchParams` — объект, представляющий собой строку поиска HTTP запроса
- `Blob` — бинарные данные, например содержимое файлов

body

- Строка — JSON в виде строки
- `FormData` — специальный объект, который позволяет конструировать запросы в виде данных из формы
- `URLSearchParams` — объект, представляющий собой строку поиска HTTP запроса
- `Blob` — бинарные данные, например содержимое файлов
- `BufferSource` — бинарные данные, создающиеся с помощью ArrayBuffer’ов

Обратная связь в интерфейсе

При синхронизации данных с сервером, нельзя менять состояние контрола мгновенно — нужно обновить состояние только в том случае, если запрос удался

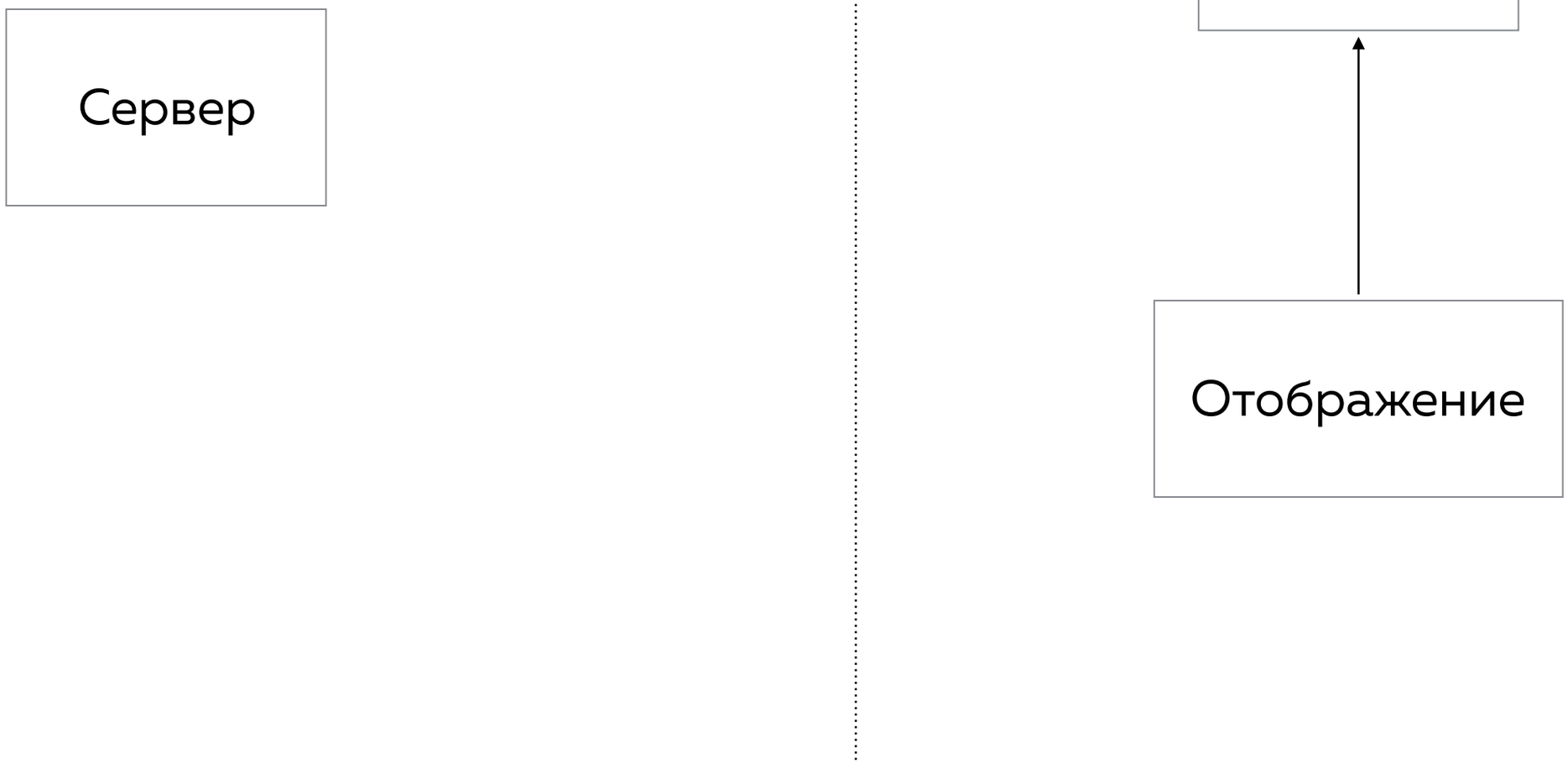
Обратная связь в интерфейсе

Сервер

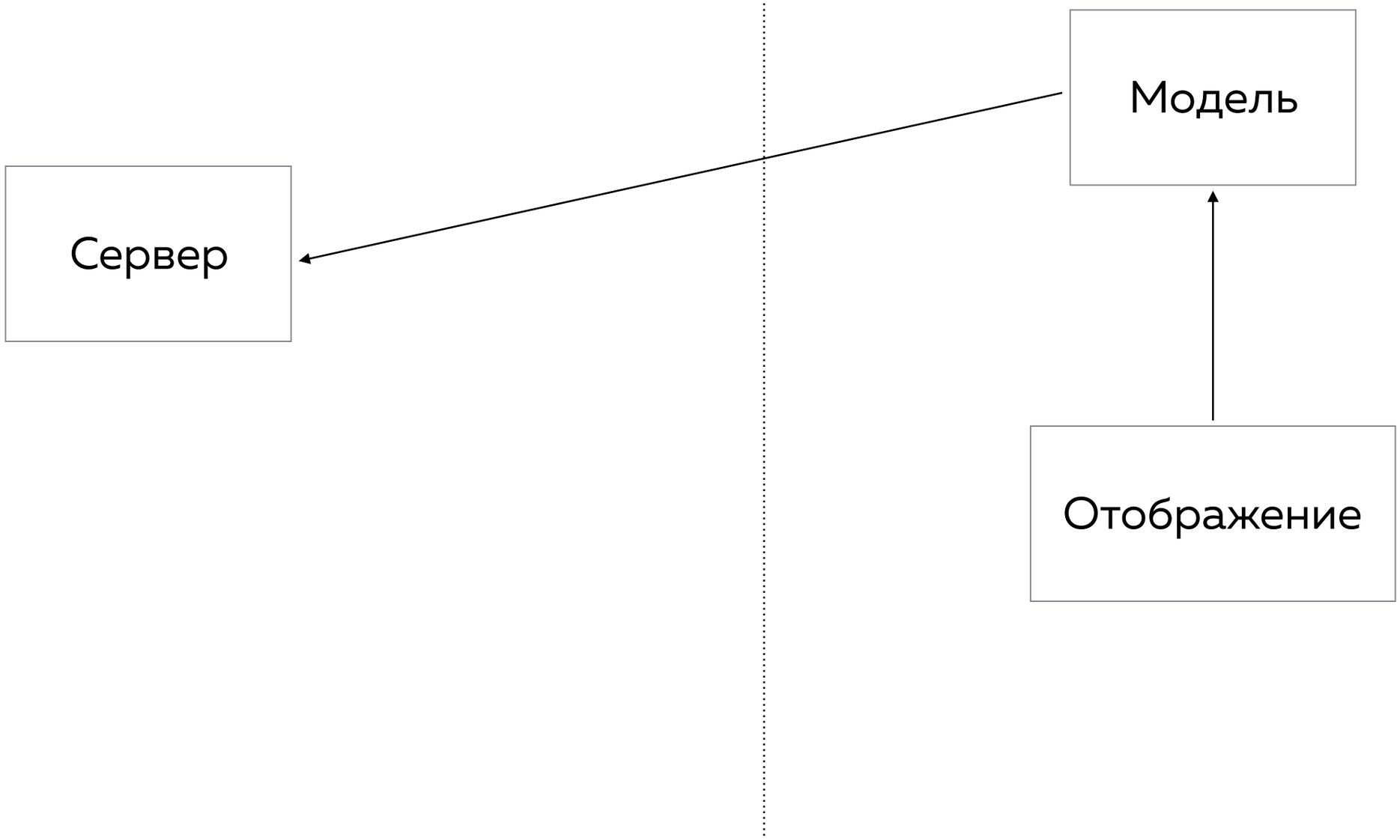
Модель

Отображение

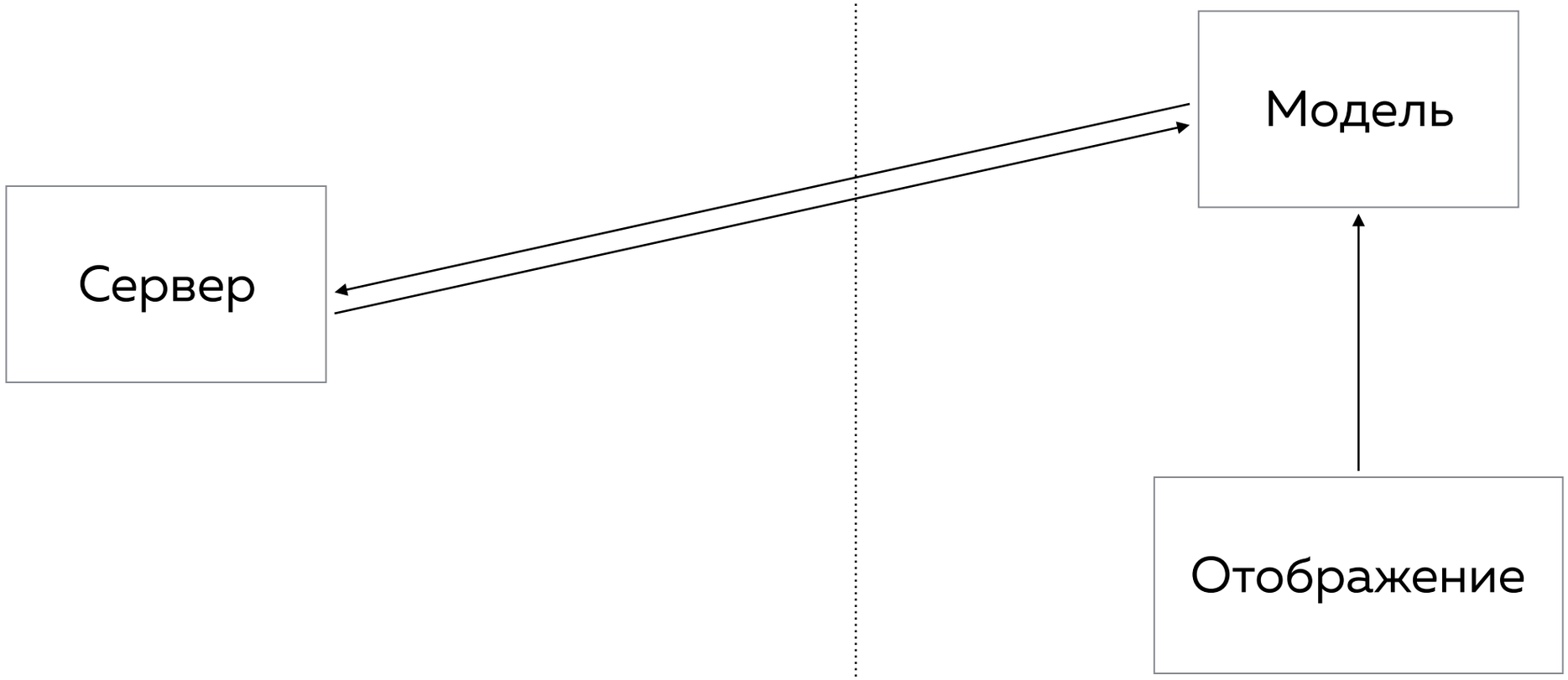
Обратная связь в интерфейсе



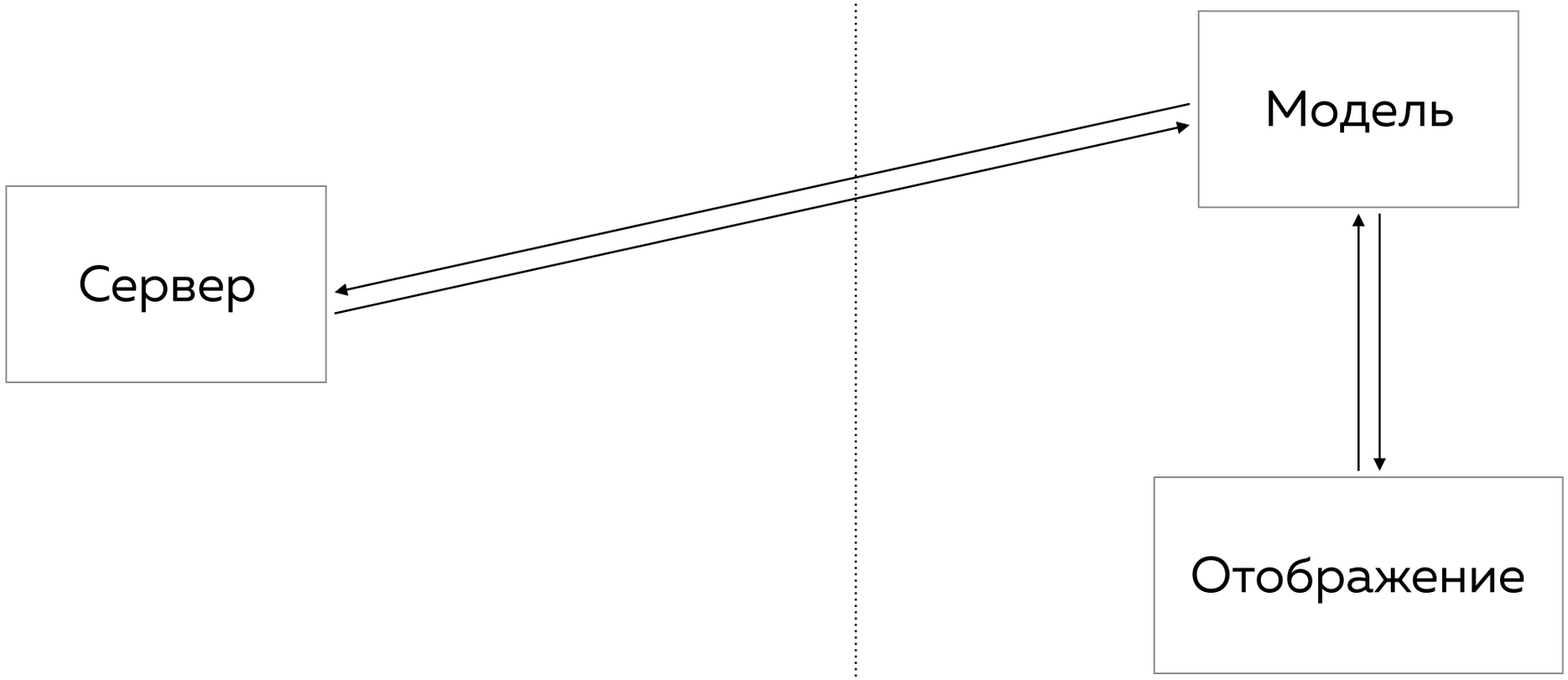
Обратная связь в интерфейсе



Обратная связь в интерфейсе



Обратная связь в интерфейсе



Обратная связь курильщика

100

Обратная связь курильщика

100

Обратная связь курильщика

101

Обратная связь курильщика

101

GET http://localhost/ 500 Internal…

Обратная связь курильщика

101

GET http://localhost/ 500 Internal…

Some error happened

Обратная связь здорового человека

100

Обратная связь здорового человека

100

Обратная связь здорового человека

100

Обратная связь здорового человека

100

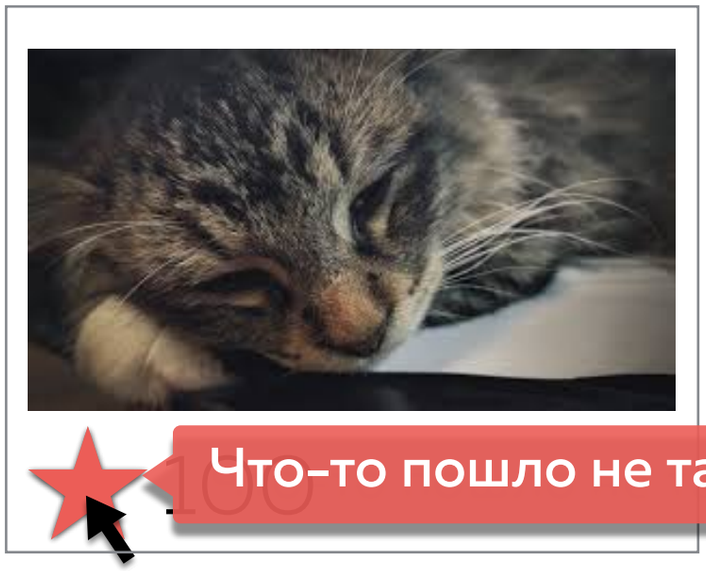
GET http://localhost/ 500 Internal…

Обратная связь здорового человека

100

GET http://localhost/ 500 Internal…

Обратная связь здорового человека



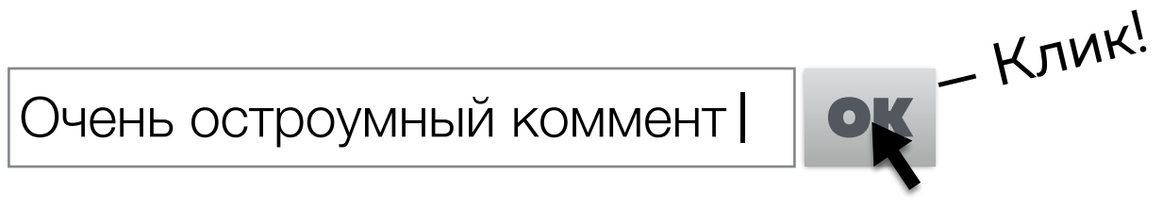
GET http://localhost/ 500 Internal…

Обратная связь курильщика — 2

Месть обратной связи

Обратная связь курильщика — 2

Месть обратной связи



Обратная связь курильщика — 2

Месть обратной связи

Обратная связь курильщика — 2

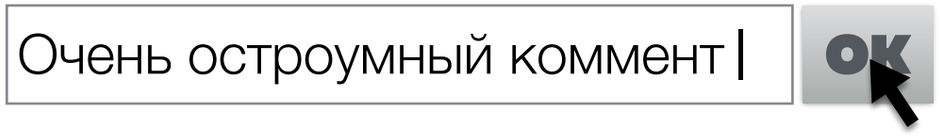
Месть обратной связи

Обратная связь курильщика — 2

Месть обратной связи

Обратная связь курильщика — 2

Месть обратной связи



Остроумный коммент

```
GET http://localhost/ 204
```

Обратная связь курильщика — 2

Месть обратной связи

Остроумный коммент

Остроумный коммент

```
GET http://localhost/ 204
GET http://localhost/ 204
```

Обратная связь курильщика — 2

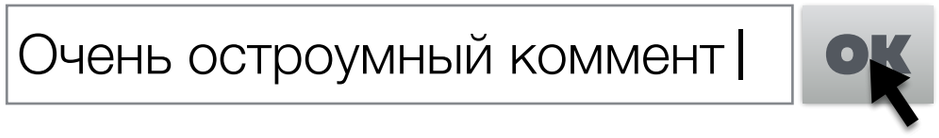
Месть обратной связи

Остроумный коммент 👎

Остроумный коммент 👎

```
GET http://localhost/ 204
GET http://localhost/ 204
```

Обратная связь здорового человека: новая надежда



Обратная связь здорового человека: новая надежда

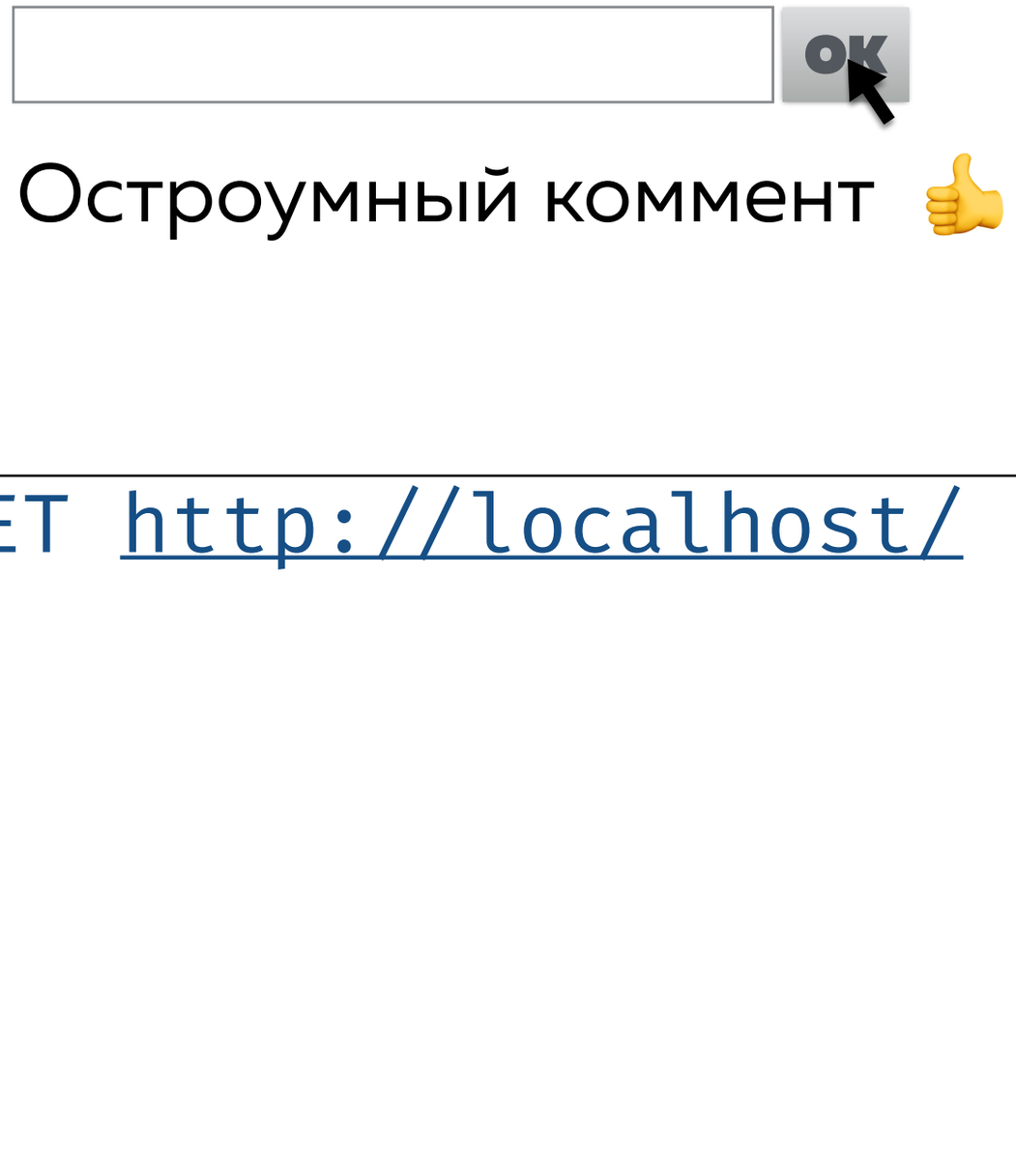
Обратная связь здорового человека: новая надежда

Обратная связь здорового человека: новая надежда

Остроумный коммент

GET http://localhost/ 204

Обратная связь здорового человека: новая надежда



GET http://localhost/ 204

Обратная связь здорового человека: новая надежда

Остроумный коммент 👍

Очень остроумно!

```
GET http://localhost/ 204
```


echo "— Adios!" > farewell.txt