html academy
интерактивные
онлайн-курсы

Раздел 5:

REST. Express

В этой лекции

В этой лекции

– REST

В этой лекции

– REST

– Express.js

В этой лекции

– REST

– Express.js

– Тесты

REST

REST —

Representational **S**tate **T**ransfer *(передача представления состояния)*. 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

REST

– ресурсы

REST

– ресурсы

– управляющие глаголы

REST

- ресурсы
- управляющие глаголы
  - GET — *получение*

REST

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

REST

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

REST

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

REST

Сайт

Мобильное приложение

клиент

Комменты

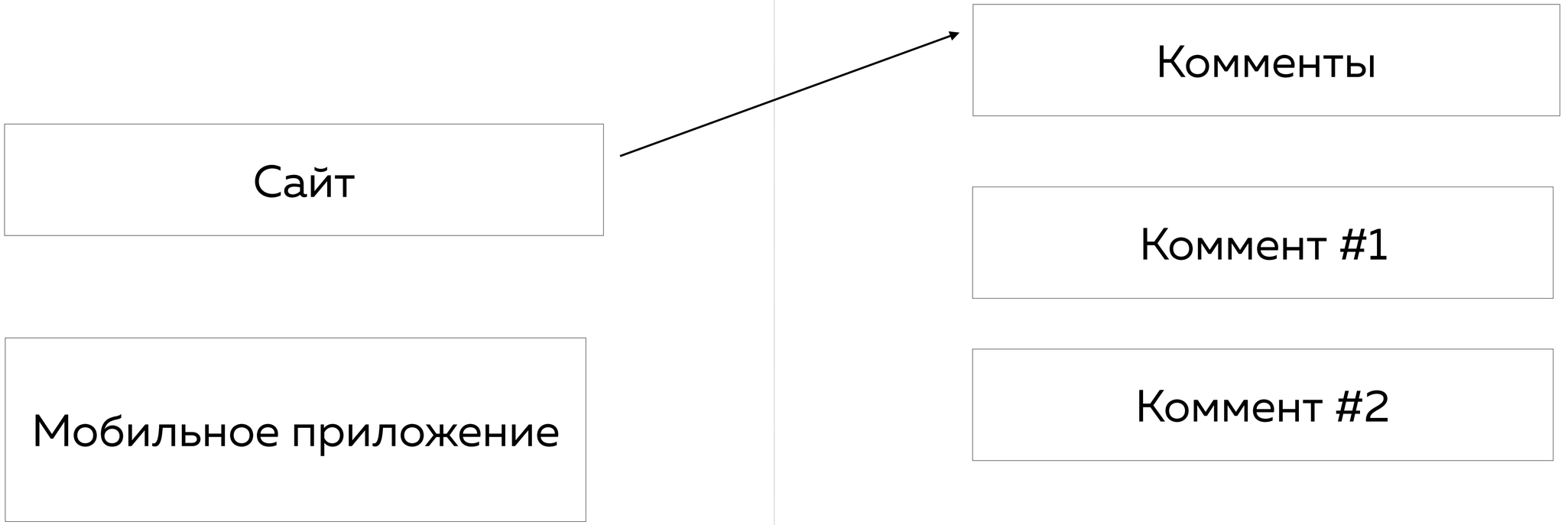
Коммент #1

Коммент #2

сервер

REST

GET /comments

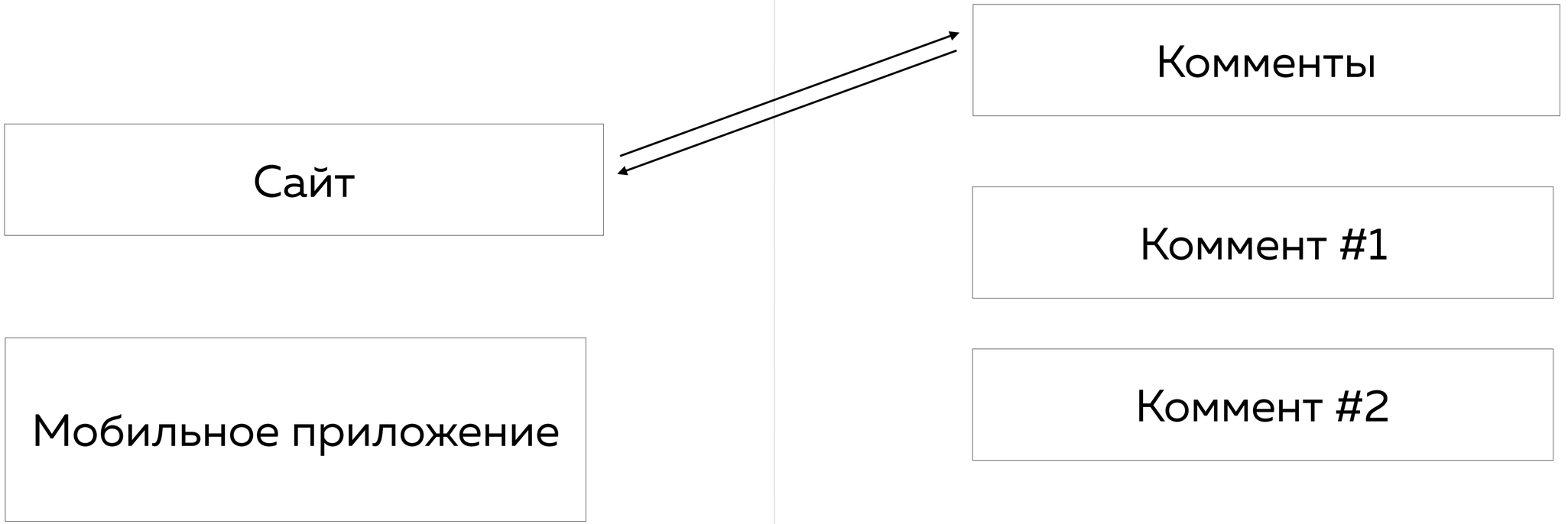


клиент

сервер

REST

GET /comments



клиент

сервер

REST

Сайт

Мобильное приложение

клиент

Комменты

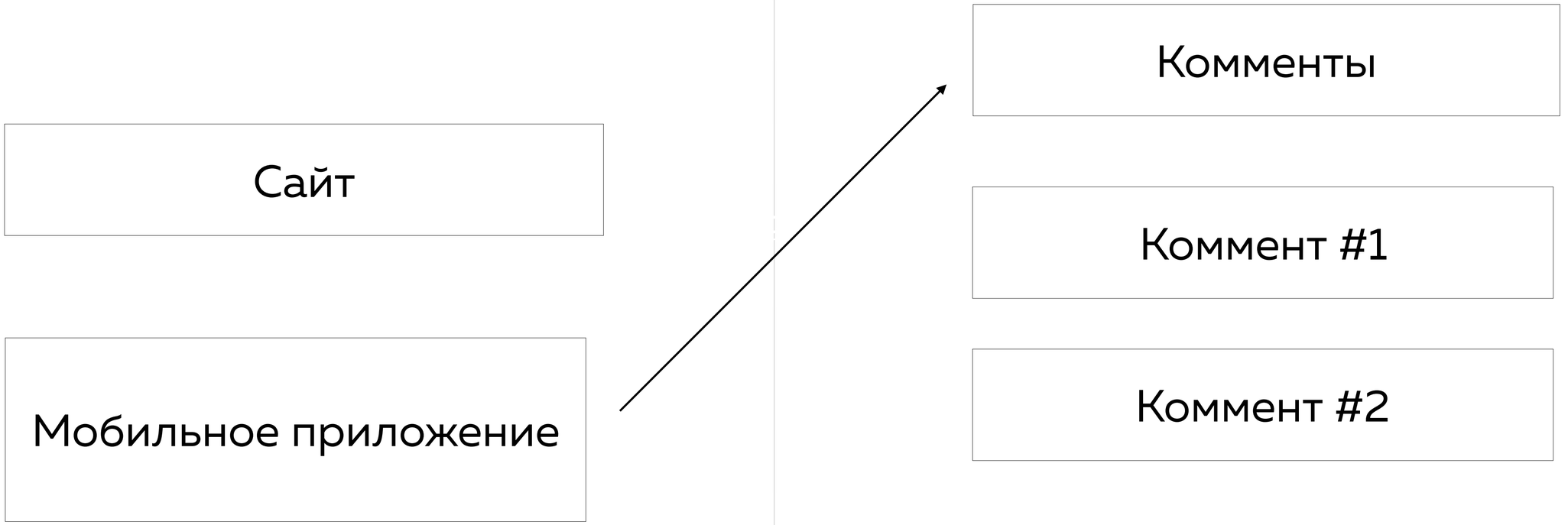
Коммент #1

Коммент #2

сервер

REST

POST /comments

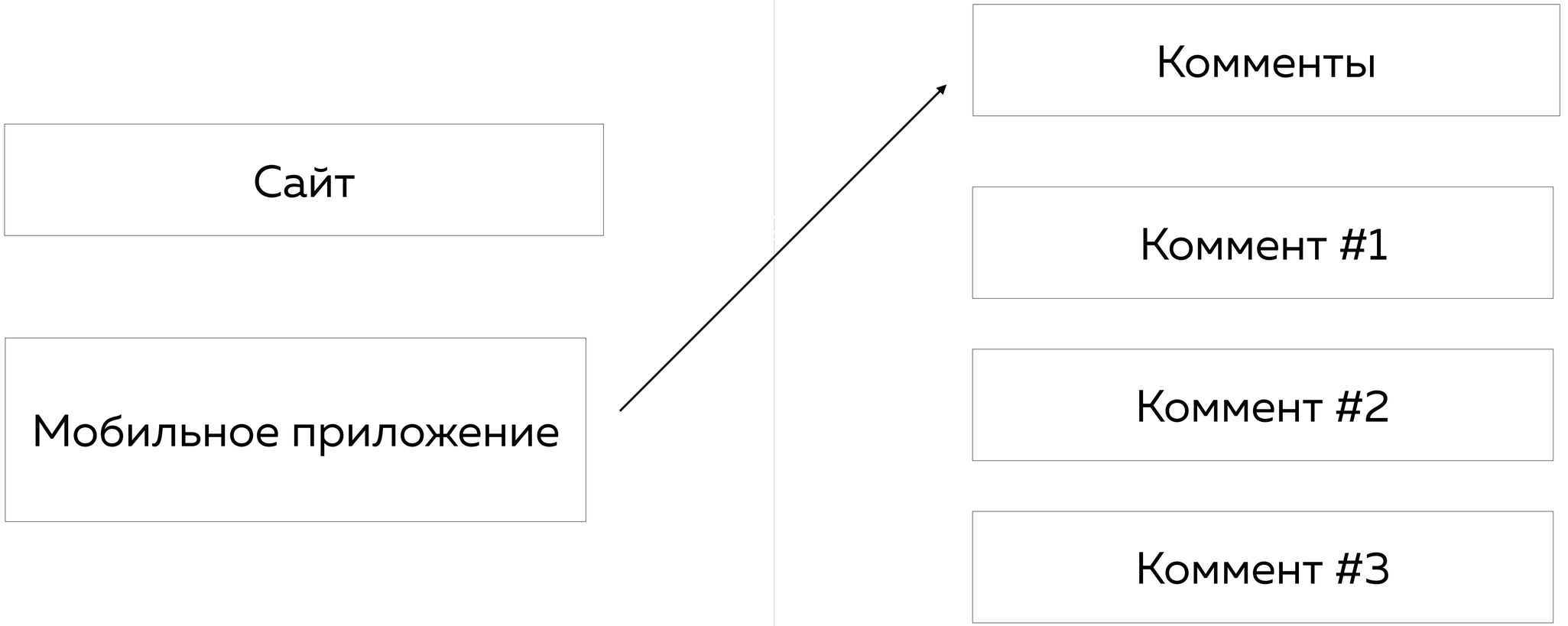


клиент

сервер

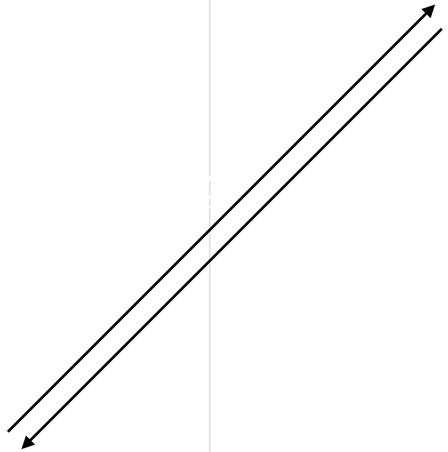
REST

POST /comments



REST

POST /comments



Сайт

Мобильное приложение

Комменты

Коммент #1

Коммент #2

Коммент #3

клиент

сервер

REST

Сайт

Мобильное приложение

клиент

Комменты

Коммент #1

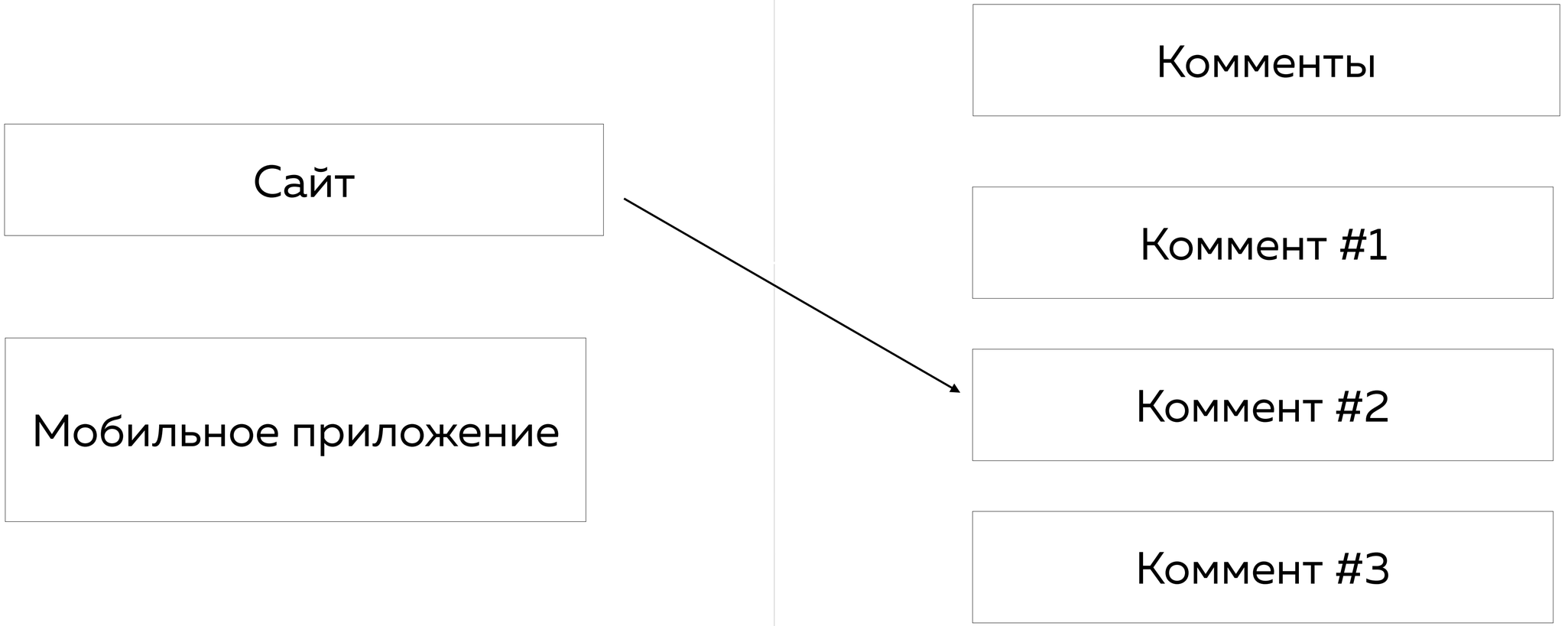
Коммент #2

Коммент #3

сервер

REST

PUT /comments/2



REST

Сайт

Мобильное приложение

клиент

Комменты

Коммент #1

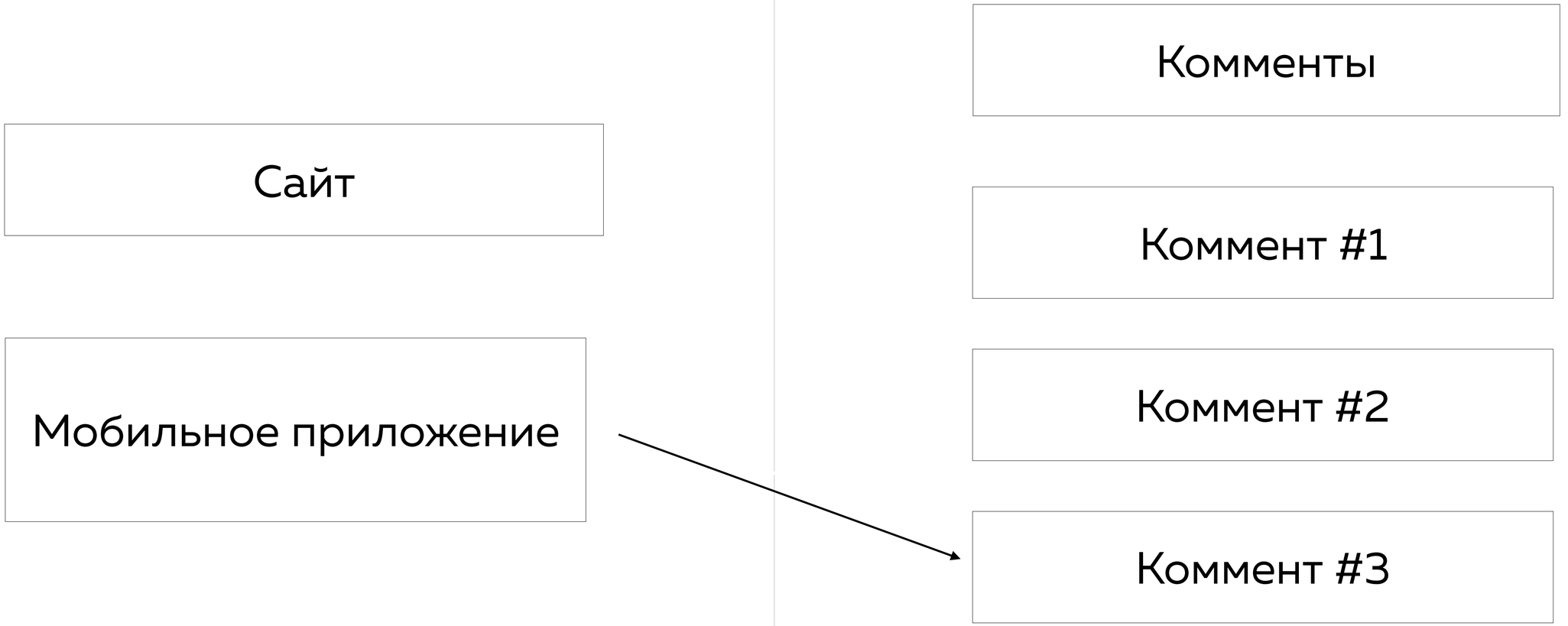
Коммент #2

Коммент #3

сервер

REST

```
DELETE /comments/3
```



REST

Сайт

Мобильное приложение

клиент

Комменты

Коммент #1

Коммент #2

сервер

Express

Фреймворк для написания веб-приложений для платформы *Node.js*


4
4

Hello Express

```
const express = require('express');
const app = express();

app.get('/', (req, res) => res.send('Hello World!'));

app.listen(3000, () =>
  console.log('Example app listening on port 3000!'));
```

expressjs.com

Какие проблемы решает

Какие проблемы решает

– Разделение обработчиков по адресам и методам

Какие проблемы решает

– Разделение обработчиков по адресам и методам

– Управление заголовками

Какие проблемы решает

– Разделение обработчиков по адресам и методам

– Управление заголовками

– Управление статикой

Какие проблемы решает

– Разделение обработчиков по адресам и методам

– Управление заголовками

– Управление статикой

– Управление кешированием

Какие проблемы решает

– Разделение обработчиков по адресам и методам

– Управление заголовками

– Управление статикой

– Управление кешированием

– Многое другое

Middleware (обработчик)

Обработчик (слой) — абстракция, которая позволяет обработать запрос и вернуть ответ, либо передать запрос дальше

Виды обработчиков (*middleware*)

Виды обработчиков (*middleware*)

– Фильтрующие:

  Слои, которые извлекают какую-то информацию из запроса и передают управление дальше

Виды обработчиков (*middleware*)

- Фильтрующие:
  Слои, которые извлекают какую-то информацию из запроса и передают управление дальше

- Обрабатывающие:
  Слои, которые обрабатывают конкретный запрос и возвращают ответ пользователю

Виды обработчиков (*middleware*)

- Фильтрующие:
  Слои, которые извлекают какую-то информацию из запроса и передают управление дальше

- Обрабатывающие:
  Слои, которые обрабатывают конкретный запрос и возвращают ответ пользователю

- Корректирующие:
  Слои, которые обрабатывают исключительные ситуации, ошибки и т.д. Нужны для обработки непредвиденных или ожидаемых ошибок

Router (маршрутизатор)

Частный случай обработчика (middleware), который отвечает за конкретный результат и обрабатывает конкретный запрос с конкретным методом

```
app.METHOD(PATH, HANDLER)
```

app.METHOD(PATH, HANDLER)

— *app* — экземпляр текущего приложения *express*

app.METHOD(PATH, HANDLER)

– *app* — экземпляр текущего приложения *express*

– *METHOD* — метод, обработываемого запроса (*GET*, *POST*, *PUT*, *DELETE*...)

app.METHOD(PATH, HANDLER)

– *app* — экземпляр текущего приложения *express*

– *METHOD* — метод, обработываемого запроса (*GET*, *POST*, *PUT*, *DELETE*...)

– *PATH* — путь (регистронезависим по умолчанию), который обрабатывает обработчик (*'/user'*, *'/profile/admin'*,...)

app.METHOD(PATH, HANDLER)

- *app* — экземпляр текущего приложения *express*
- *METHOD* — метод, обработываемого запроса (*GET*, *POST*, *PUT*, *DELETE*...)
- *PATH* — путь (регистронезависим по умолчанию), который обрабатывает обработчик (*'/user'*, *'/profile/admin'*,...)
- *HANDLER* — обработчик, который сработает при совпадении пути и метода запроса

Пример

```
const express = require(`express`);

const app = express();

app.get(`/`, (req, res) => res.send(`Hello World!`));

app.post(`/`, (req, res) => res.send(`Got a POST request!`));

app.put(`/user`, (req, res) => res.send(`Got a PUT request at /user`));

app.delete(`/user`, (req, res) => res.send(`Got a DELETE request at /user`));

app.listen(3000, () => console.log(`Example app listening on port 3000!`));
```

expressjs.com/en/starter/basic-routing.html

Методы

expressjs.com/en/4x/api.html#routing-methods

Методы

– Поддержка всех базовых методов — *GET*, *HEAD*, *POST*, *PUT*, *DELETE*...

expressjs.com/en/4x/api.html#routing-methods

Методы

– Поддержка всех базовых методов — *GET*, *HEAD*, *POST*, *PUT*, *DELETE*...

– Поддержка дополнительных методов — *OPTIONS*, *PATCH*...

expressjs.com/en/4x/api.html#routing-methods

Методы

– Поддержка всех базовых методов — *GET*, *HEAD*, *POST*, *PUT*, *DELETE*...

– Поддержка дополнительных методов — *OPTIONS*, *PATCH*...

– Все методы записываются с маленькой буквы — *get*, *post*, *put*, *delete*...

expressjs.com/en/4x/api.html#routing-methods

Методы

– Поддержка всех базовых методов — *GET*, *HEAD*, *POST*, *PUT*, *DELETE*...

– Поддержка дополнительных методов — *OPTIONS*, *PATCH*...

– Все методы записываются с маленькой буквы — *get*, *post*, *put*, *delete*...

– Специальный метод *all* позволяет обработать запрос вручную

expressjs.com/en/4x/api.html#routing-methods

Статические пути (*static path*)

```
// Корень сайта
app.get(`/`, (req, res) => res.send(`root`));

// Раздел /about
app.get(`/about`, (req, res) => res.send(`root`));

// Путь /random.text
app.get(`/random.text`, (req, res) => res.send(`random.text`));
```

https://expressjs.com/en/guide/routing.html

Динамические пути (*dynamic path*)

```
// Путь /keks или /kks
app.get(`/ke?ks`, (req, res) => res.send(`keks`));

// Путь /keeeeks или /keks, или /keeks и т.д.
app.get(`/ke+ks`, (req, res) => res.send(`keks`));

// Путь /keks или /keks, или /ks
app.get(`/k(ek)?s`, (req, res) => res.send(`keks`));
```

https://expressjs.com/en/guide/routing.html

Параметризованные пути (*dynamic path*)

```
// Для запроса GET /cat/keks — вернёт keks
// Для запроса GET /cat/something — вернёт something
app.get(`/cat/:name`, (req, res) => res.send(req.params.name));

// Для запроса GET /cat/keks/thegreat — вернёт keks thegreat
// Для запроса GET /cat/Кот/Матроскин — вернёт Кот Матроскин
app.get(`/cat/:name/:familyname`, (req, res) => res.send(`$
{req.params.name} ${req.params.familyname}`));
```

https://expressjs.com/en/guide/routing.html

Регулярные выражения (*regexp path*)

```
// Ловит всё что содержит keks — /keks-cat, /getkeksnow, ....
app.get(/keks/, (req, res) => res.send(`keks`));

// Ловит всё что заканчивается на keks — /mykeks, /catkeks, но
не /keks-cat, /getkeksnow...
app.get(/.*keks$/, (req, res) => res.send(`keks`));
```

https://expressjs.com/en/guide/routing.html

express.static

expressjs.com/en/4x/api.html#express.static

express.static

– Раздаёт статику с диска

expressjs.com/en/4x/api.html#express.static

express.static

– Раздаёт статику с диска

– Следит за правильным кешированием файлов

expressjs.com/en/4x/api.html#express.static

express.static

– Раздаёт статику с диска

– Следит за правильным кешированием файлов

– Предоставляет *index.html* как корневой ресурс

expressjs.com/en/4x/api.html#express.static

Асинхронная обработка запроса

Как не стоит делать

```
const app = require(`express`)();
const WIZARDS = [];
const NOT_FOUND_CODE = 404;

app.get(`/wizards/:name`, (req, res) => {
  const wizardName = req.params.name;

  (async () => {
    const found = WIZARDS.find((it) => it.name === wizardName);
    if (!found) {
      res.status(NOT_FOUND_CODE).send(`Wizard with name "${wizardName}" not found`);
    }
    res.send(found);
  })().catch((e) => res.status(500).send(e.message));
});
```

Правильный асинхронный обработчик

```
const app = require(`express`)();
const WIZARDS = [];
const NOT_FOUND_CODE = 404;

const async = (fn) => (req, res, next) => fn(req, res, next).catch(next);

app.get(`/wizards/:name`, async(async (req, res) => {
  const wizardName = req.params.name;

  const found = WIZARDS.find((it) => it.name === wizardName);
  if (!found) {
    res.status(NOT_FOUND_CODE).send(`Wizard with name "${wizardName}" not found`);
  }
  res.send(found);
}));
```

Request extends http.IncomingMessage

expressjs.com/en/4x/api.html#req

Request extends http.IncomingMessage

– Расширяет стандартные возможности *http.IncomingMessage*

expressjs.com/en/4x/api.html#req

Request extends http.IncomingMessage

– Расширяет стандартные возможности *http.IncomingMessage*

– Автоматически разбирает *query* в объект — *req.query*

expressjs.com/en/4x/api.html#req

Request extends http.IncomingMessage

– Расширяет стандартные возможности *http.IncomingMessage*

– Автоматически разбирает *query* в объект — *req.query*

– Дополнительные методы и поля для работы с кодировкой, параметрами путей и т.д.

expressjs.com/en/4x/api.html#req

Request.body

Request.body

– По умолчанию *req.body === undefined*

Request.body

- По умолчанию *req.body === undefined*
- Express не читает тело запроса по умолчанию, для этого ему нужно знать в каком формате находятся данные и как их читать

Request.body

- По умолчанию *req.body === undefined*
- Express не читает тело запроса по умолчанию, для этого ему нужно знать в каком формате находятся данные и как их читать
- Для управления автоматическим чтением данных из запроса используются сторонние библиотеки

Body parser

Body parser

– Библиотека *body-parser*, работает со следующими форматами данных

Body parser

- Библиотека *body-parser*, работает со следующими форматами данных
  - *application/json*

Body parser

- Библиотека *body-parser*, работает со следующими форматами данных
  - *application/json*
  - *text/plain*

Body parser

- Библиотека *body-parser*, работает со следующими форматами данных
  - *application/json*
  - *text/plain*
  - *application/x-www-form-urlencoded*

Body parser

- Библиотека *body-parser*, работает со следующими форматами данных
  - *application/json*
  - *text/plain*
  - *application/x-www-form-urlencoded*
- Библиотеки *multer*, *busboy*, *multiparty* работают с данными в формате формы:

Body parser

- Библиотека *body-parser*, работает со следующими форматами данных
  - *application/json*
  - *text/plain*
  - *application/x-www-form-urlencoded*
- Библиотеки *multer*, *busboy*, *multiparty* работают с данными в формате формы:
  - *multipart/form-data*

Response extends http.ServerResponse

expressjs.com/en/4x/api.html#res

Response extends http.ServerResponse

– Расширяет стандартные возможности *http.ServerResponse*

expressjs.com/en/4x/api.html#res

Response extends http.ServerResponse

- Расширяет стандартные возможности *http.ServerResponse*
- Автоматически подбирает правильный ответ в соответствии с содержимым в res.send

expressjs.com/en/4x/api.html#res

Response extends http.ServerResponse

- Расширяет стандартные возможности *http.ServerResponse*
- Автоматически подбирает правильный ответ в соответствии с содержимым в res.send
- Добавляет дополнительные методы для работы с файлами, частичной отправкой данных и т.д.

expressjs.com/en/4x/api.html#res

Router

Объект, который позволяет логически обобщать несколько роутов

Тестирование

Как проверить работоспособность *REST API*

Как проверить работоспособность *REST API*

– Вручную написать тестовую *html*-страницу

Как проверить работоспособность *REST API*

– Вручную написать тестовую *html*-страницу

– Использовать встроенные в систему утилиты (*curl*, *wget*...)

Как проверить работоспособность *REST API*

– Вручную написать тестовую *html*-страницу

– Использовать встроенные в систему утилиты (*curl*, *wget*...)

– Использовать внешние графические утилиты (*insomnia*, *poster*...)

Как проверить работоспособность *REST API*

– Вручную написать тестовую *html*-страницу

– Использовать встроенные в систему утилиты (*curl*, *wget*...)

– Использовать внешние графические утилиты (*insomnia*, *poster*...)

– Написать тесты, которые будут автоматически проверять работу *API*

Библиотека supertest

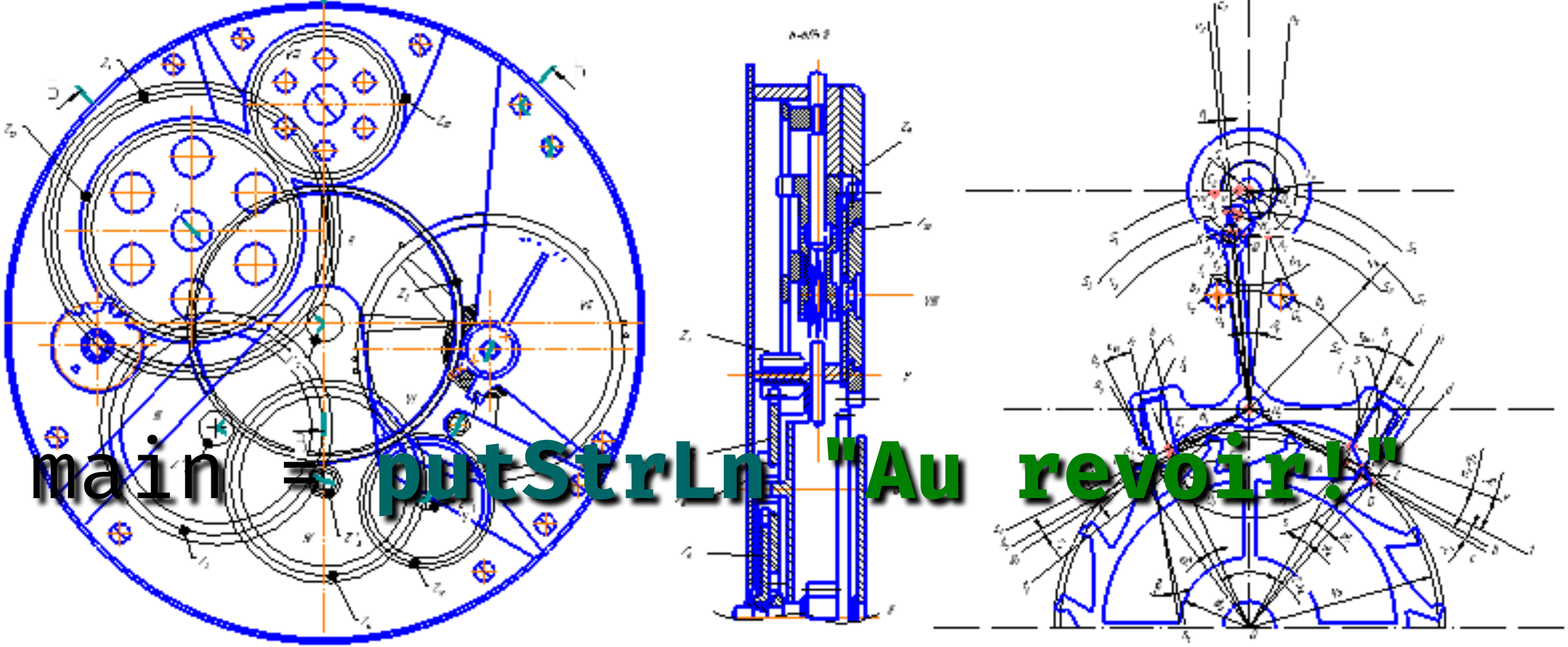
```
const request = require(`supertest`);
const express = require(`express`);
const assert = require(`assert`);

const app = express();

app.get(`/users`, function (req, res) {
  res.status(200).send([{name: `tobi`}]);
});

describe(`GET /users`, function () {
  it(`respond with json`, function () {
    return request(app).get(`/users`).set(`Accept`, `application/json`).
        expect(200).
        expect(`Content-Type`, /json/).
        then((response) => {
          const users = response.body;
          assert(users.length, 1);
          assert(users[0].name, `tobi`);
        });
  });
});
```

www.npmjs.com/package/supertest



```
main = putStrLn "Au revoir!"
```