ОНЛАЙН-ОБРАЗОВАНИЕ

OTUS

Самый простой back-end на NodeJS

Занятие № 36

Евгений Аристов

Дворжецкий Юрий

Меня хорошо слышно && видно?

Напишите в чат, если есть проблемы!

Ставьте + если все хорошо
Ставьте - если есть проблемы

Аушев Мустафа

- Backend разработчик
- Пишу на node.js/python/golang
- Архитектор (иногда)

05

Express

OTUS

Упражнение:
Установим Express?

```
// package.json
{
  "private": true,
  "dependencies": {
    "express": "4.16.3"
  }
}
```

Или

```
npm init
npm i express -save
```

OTUS

Упражнение:
Установим Express?

```
// server.js
var express = require('express');
var app = express();

app.get('/', function(req, res){
    res.send('Hello World');
});

app.listen(3000);
```

и npm start
и откроем localhost:3000

OTUS

Упражнение:
Установим Express?

// server.js и npm start localhost:3000/hello?name=Ivan

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

app.get('/hello', function(req, res){
    res.send('Hello ' + req.query.name);
});

app.listen(3000);
```

Request и response

- Могли уже заметить объекты request и response
- С помощью них можно получать параметры запроса
- И управлять ответом
- Да, и их можно обогащать – про это следующее занятие ☺️

Request

https://expressjs.com/ru/api.html

- req.app
- req.baseUrl
- req.body
- req.cookies
- req.fresh
- req.hostname
- req.ip
- req.ips
- req.method
- req.originalUrl
- req.params
- req.path
- req.protocol
- req.query
- req.route
- req.secure
- req.signedCookies
- req.stale
- req.subdomains
- req.xhr

- req.accepts()
- req.acceptsCharsets()
- req.acceptsEncodings()
- req.acceptsLanguages()
- req.get()
- req.is()
- req.param()
- req.range()

Обслуживание статического веб-контента

OTUS

- Статический контент таким способом не отправляют
- Обычно используют middleware

```
app.use(express.static('public'));
app.use(express.static('files'));
```

```
app.use('/static', express.static('public'));
```

```
http://localhost:3000/images/kitten.jpg
http://localhost:3000/css/style.css
http://localhost:3000/js/app.js
http://localhost:3000/images/bg.png
http://localhost:3000/hello.html
```

```
http://localhost:3000/static/images/kitten.jpg
http://localhost:3000/static/css/style.css
http://localhost:3000/static/js/app.js
http://localhost:3000/static/images/bg.png
http://localhost:3000/static/hello.html
```

Роутинг

- Обозначения маршрутов – не просто обозначения
- С помощью них можно получать параметры
- Крайне полезно в RESTful API
- Используется библиотека path-to-to-regexp

- https://www.npmjs.com/package/path-to-regexp
- https://expressjs.com/ru/guide/routing.html

OTUS

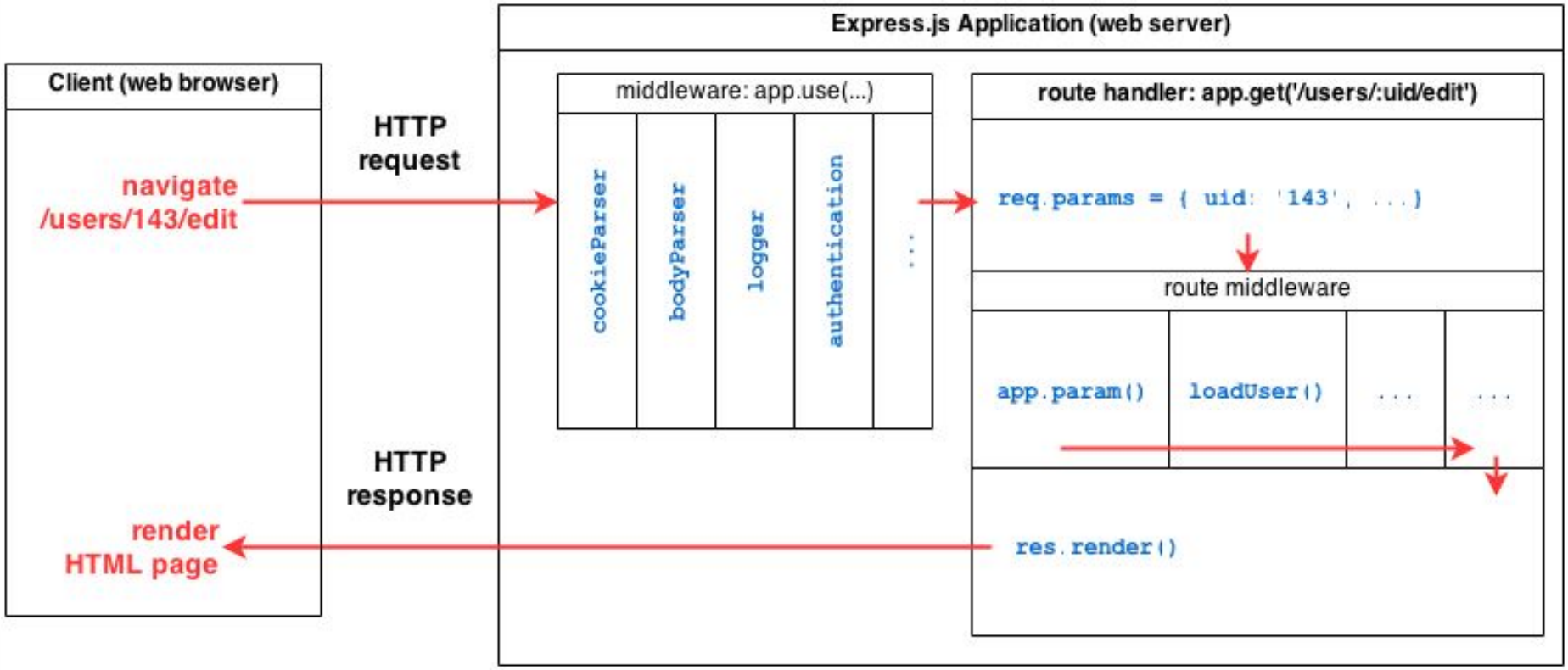
Express Middlewares

Пример middlewares

OTUS

```
app.use((req, res, next) => {
    console.log(new Date().toISOString());
    next();
});


app.get('/person', (req, res) => res.send({name: 'Ivan'}));


app.listen(3000);
```

Middlewares



Static web

OTUS

```
var express = require('express')
var path = require('path')
var serveStatic = require('serve-static')

var app = express()

app.use(serveStatic(path.join(__dirname, 'public-optimized')))
app.use(serveStatic(path.join(__dirname, 'public')))
app.listen(3000)
```

Ваши вопросы?

04

Шаблонизация

- Express может рендерить HTML-страницы
- Вообще это «классический Web»
- В SPA статика отдаётся nginx, рендеринг на стороне клиента
- Но «классический» тоже используется, когда принципиально SEO

- Ejs
- Pug (ex Jade)
- Nunjucks
- Mustache
- Angular SSR
- React SSR

И можете даже свой создать

https://expressjs.com/ru/guide/using-template-engines.html

Ejs (embedded javascript templates

OTUS

```
<% if (user) { %>
  <h2><%= user.name %></h2>
<% } %>
```

Mustache

```
<h1>{{header}}</h1>
{{#bug}}
{{/bug}}

{{#items}}
  {{#first}}
    <li><strong>{{name}}</strong></li>
  {{/first}}
  {{#link}}
    <li><a href="{{url}}">{{name}}</a></li>
  {{/link}}
{{/items}}

{{#empty}}
  <p>The list is empty.</p>
{{/empty}}
```

```
{
  "header": "Colors",
  "items": [
      {"name": "red", "first": true, "url": "#Red"},
      {"name": "green", "link": true, "url": "#Green"},
      {"name": "blue", "link": true, "url": "#Blue"}
  ],
  "empty": false
}
```

https://mustache.github.io/#demo

Pug (Jade)

OTUS

```
extends templates/site-page

block vars
    - pageClass = 'mvs-3a-page';
    - pageTitle = 'Модельно-восковой состав МВС-3А';
    - pageMetaDescription = 'МВС-3А - модельно-восковой состав';
    - pageMetaKeywords = 'спецхимпродукт,продукция,цена,фасовка,';
    - pageMetaKeywords += 'мвс-3а,ту 0255-003-23753216-2016';

block content
    .container
        .row: .col-xs-12
            h1
                | Модельно-восковой состав МВС-3А
                br
                small ТУ 0255-003-23753216-2016
        .row
            .product-content.col-xs-12.col-sm-12.col-lg-9
                p.
                    Модельно-восковой состав МВС-3А предназначен для изготовления моделей отливок изделий
                    сложной конфигурации и деталей высокой точности методом литья по выплавляемым
моделям.
                p.
                    Литье с использованием МВС-3А позволяет получать отливки
                    размерной точности 5&ndash;9 по ГОСТ 53464.
                table.table.table-bordered.product-content__characteristics
```

Pug syntax

OTUS

```
a(href='//google.com') Google
```

```
<a href="//google.com">Google</a>
```

```
h1= title
p Written with love by #{author}
```

```
<h1>On Dogs: Man's Best Friend</h1>
<p>Written with love by enlore</p>
```

Pug syntax

```
- for (var x = 0; x < 3; x++)
  li item
```

```
<li>item</li>
<li>item</li>
<li>item</li>
```

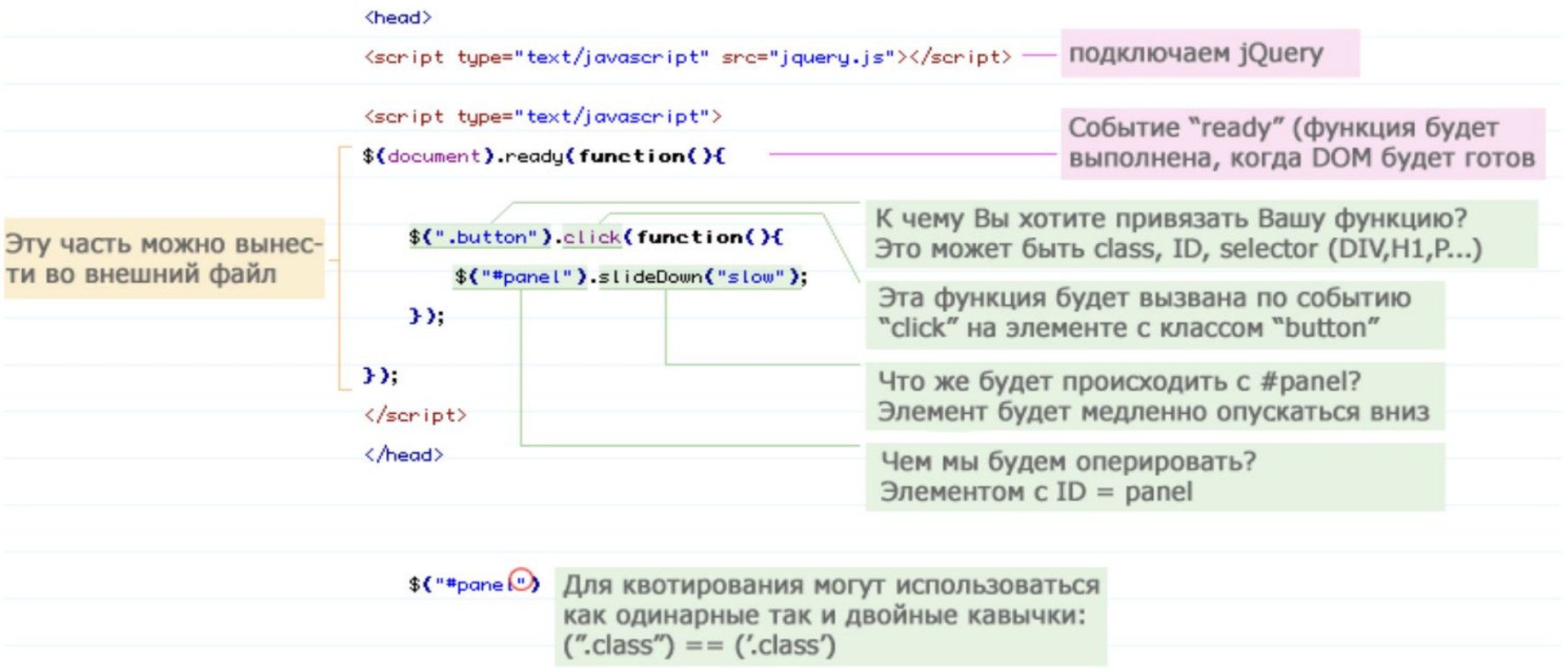
Pug syntax

OTUS

```
//- layout.pug
html
  head
    title My Site - #{title}
    block scripts
      script(src='/jquery.js')
  body
    block content
    block foot
      #footer
        p some footer content
```

```
//- page-a.pug
extends layout.pug

block scripts
  script(src='/jquery.js')
  script(src='/pets.js')

block content
  h1= title
  - var pets = ['cat', 'dog']
  each petName in pets
    include pet.pug
```

```
<head>
<script type="text/javascript" src="jquery.js"></script>
</head>
```



Ajax

OTUS

Фоновый обмен данными браузера с веб-сервером

- async — асинхронность запроса, по умолчанию true
- cache — вкл/выкл кэширование данных браузером, по умолчанию true
- contentType — по умолчанию «application/x-www-form-urlencoded»
- **data** — передаваемые данные — строка иль объект
- dataFilter — фильтр для входных данных
- **dataType** — тип данных возвращаемых в callback функцию (xml, html, script, json, text, _default)
- global — тригер — отвечает за использование глобальных AJAX Event'ов, по умолчанию true
- ifModified — тригер — проверяет были ли изменения в ответе сервера, дабы не слать еще запрос, по умолчанию false
- jsonp — переустановить имя callback функции для работы с JSONP (по умолчанию генерируется на лету)
- processData — по умолчанию отправляемые данный заворачиваются в объект, и отправляются как «application/x-www-form-urlencoded», если надо иначе — отключаем
- scriptCharset — кодировочка — актуально для JSONP и подгрузки JavaScript'ов
- timeout — время таймаут в миллисекундах
- **type** — GET либо POST
- **url** — url запрашиваемой страницы

Упражнение

Создать приложение

На странице index вывести массив 1 2 3

```
npx express-generator --view=pug express-example
```

Ваши вопросы?

05

Promise Интеграция с PostgreSQL

Упражнение

OTUS

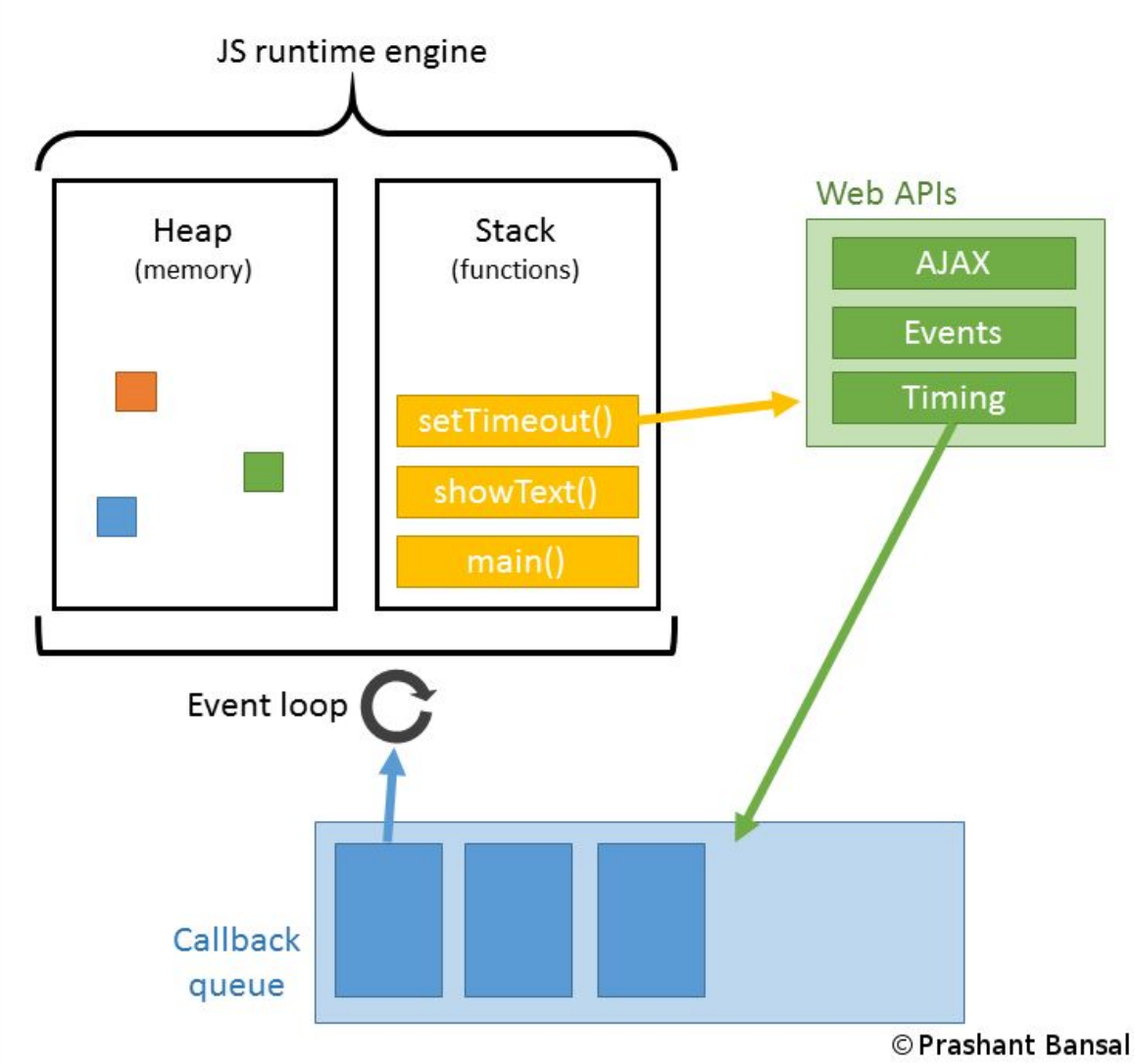
```
// Что будет выведено?

console.log('1');

setTimeout(() => console.log('2'), 1000)

console.log(3);
```

Упражнение

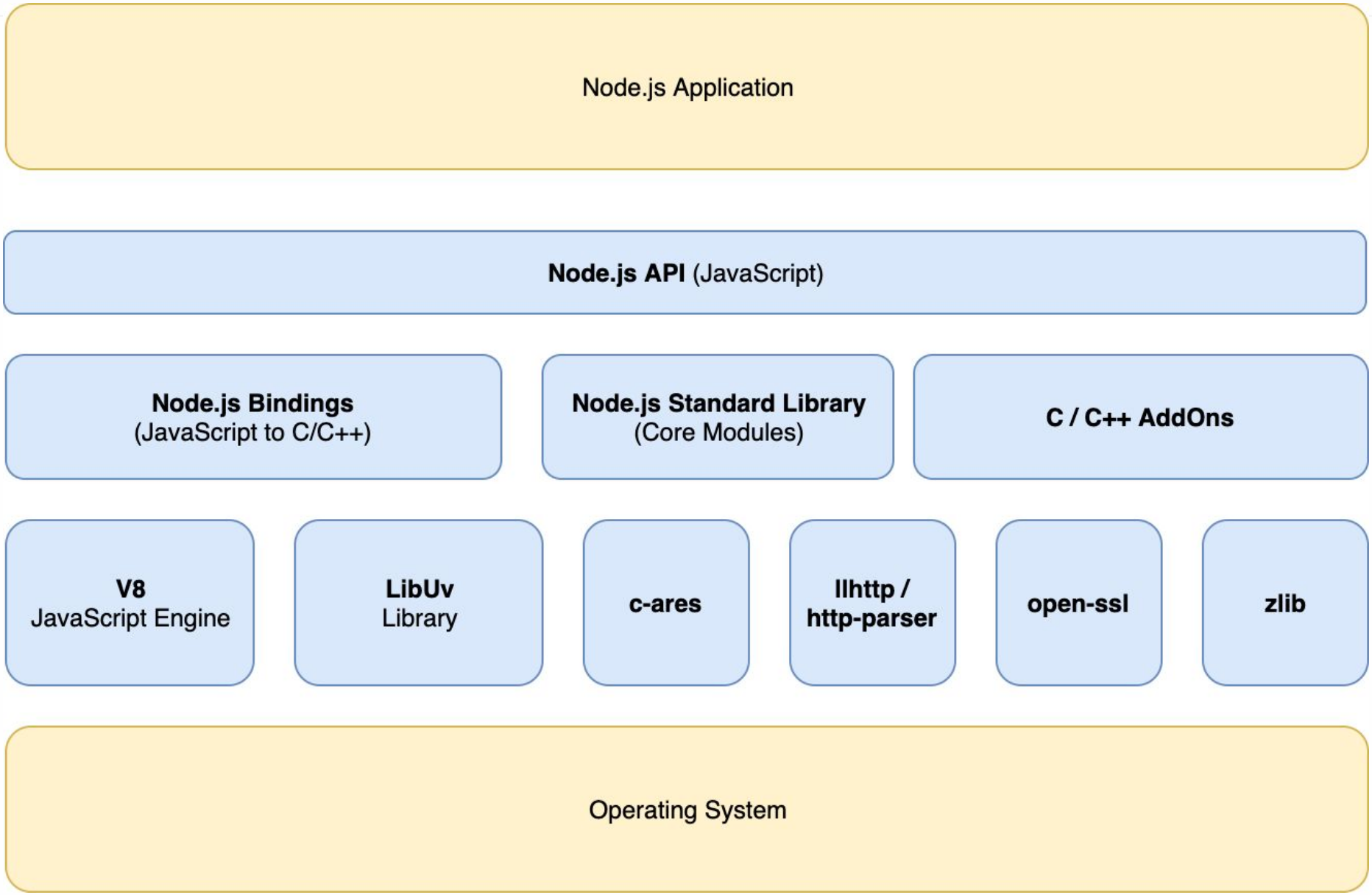
OTUS

```
// Что будет выведено?

console.log('1');

setTimeout(() => console.log('2'), 0)

console.log(3);
```

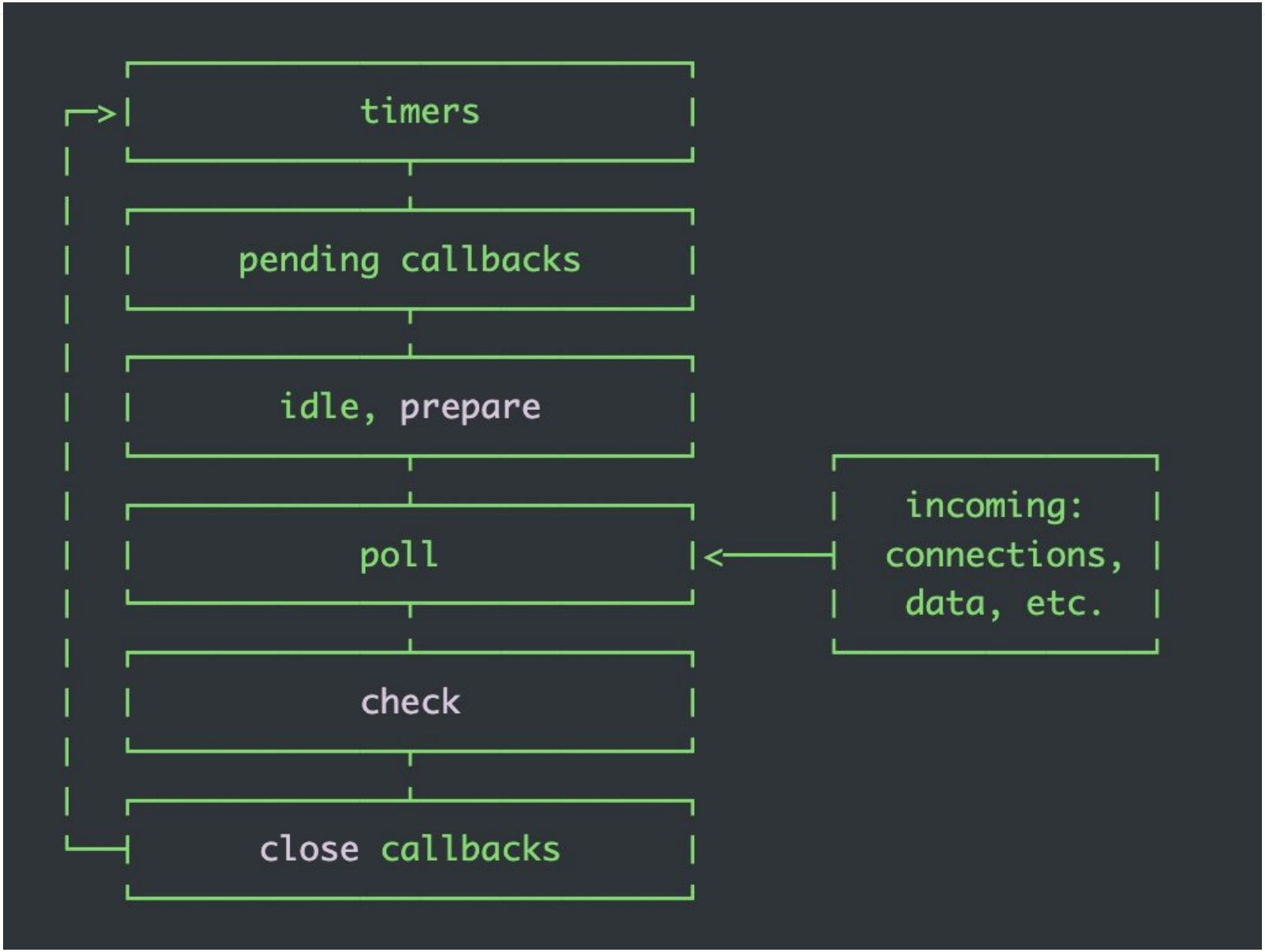
Модель памяти и Event Loop



Node.js architecture



Node.js architecture



Модель памяти

- Есть три области памяти - стек, куча и очередь событий
- Очередь событий инициирует выполнение кода
- JS однопоточен - основной поток один (но есть и другие)
- Вся иллюзия многопоточности делается на событиях
- Как следствие - обработчики должны быть быстрыми

```
let promise = new Promise((resolve, reject) => {

    setTimeout(() => {
        resolve("result");
    }, 1000);

});
promise.then(res => console.log(res)); // ?
```

```
httpGet(...)
    .then(() => ...)
    .then(() => ...)
    .catch(() => ...)
```

```
async function f() {
    return Promise.resolve(1);
}

f().then(alert); // ?
```

```
async function f() {

    let promise = httpGet();
    let result = await promise; // будет
ждать

    alert(result); // ?
}

f();
```

pg-promise

OTUS

```
var pgp = require("pg-promise")(/*options*/);
var db = pgp("postgres://username:password@host:port/database");

db.one("SELECT $1 AS value", 123)
    .then(function (data) {
        console.log("DATA:", data.value);
    })
    .catch(function (error) {
        console.log("ERROR:", error);
    });
```

https://github.com/vitaly-t/pg-promise

pg-promise

```
function createPuppy(req, res, next) {
  req.body.age = parseInt(req.body.age);
  db.none('insert into pups(name, breed, age, sex)' +
      'values(${name}, ${breed}, ${age}, ${sex})',
    req.body)
    .then(function () {
      res.status(200)
        .json({
          status: 'success',
          message: 'Inserted one puppy'
        });
    })
    .catch(function (err) {
      return next(err);
    });
}
```

https://mherman.org/blog/designing-a-restful-api-with-node-and-postgres/

Ваши вопросы?

Домашнее задание

Разработка собственного приложения для работы с кластером

Написать API для работы с физ.лицами.
Простая обёртка
Формат JSON физ.лица `{id:1, name:  'Ivan', age: 18`}
`GET /person` - получить всех/поиск
`GET /person/:id` - получение фил.лица по id
`POST /persons` - создание физ.лица
`DELETE /person` - удаление физ.лица
Нагрузить приложение через яндекс танк

Пожалуйста, пройдите опрос

https://otus.ru/polls/43436/

За опросы - плюшки

Спасибо за внимание!

Не бойтесь JS!