matrix

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

Developing Web-Applications with Node.js

matrix

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company

Course Agenda

01 | Introduction to Node.js

02 | Asynchronous programming

03 | File System Operations

04 | Basic Web Server

05 | External Modules (NPM)

06 | Express web framework

Course Agenda

07 | express Routing

08 | express Input Validation

09 | express Views

10 | express Config

11 | Working with MySQL

12 | Working with MongoDB

Course Agenda

13 | Authentication

14 | Real Time apps

15 | Testing

16 | Deploying

17 | Microservices

18 | TypeScript

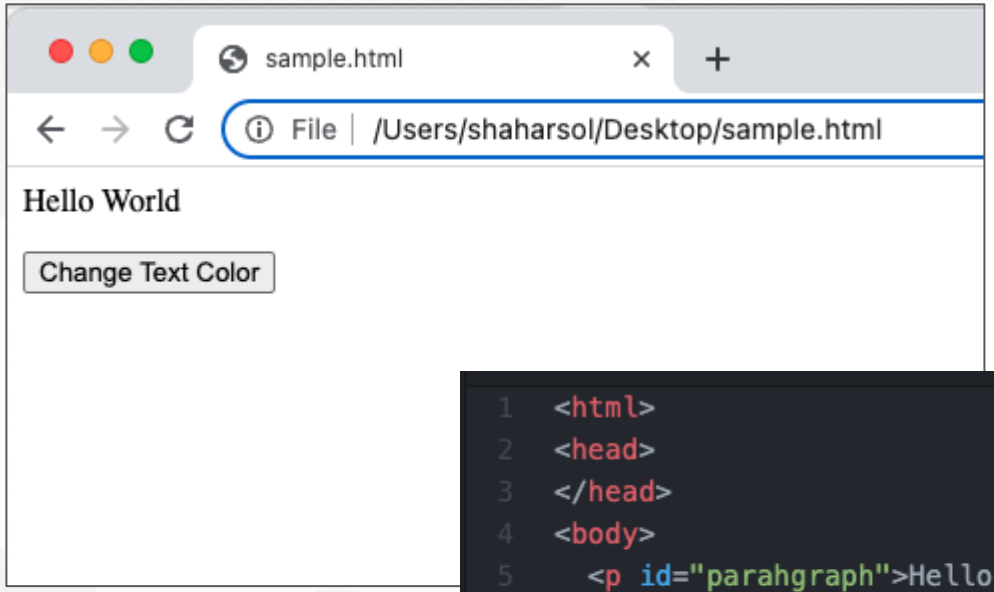
Samples Repository

https://github.com/shaharsol/jb-nodejs

01 | Introduction to Node.js

01 | Introduction to Node.js

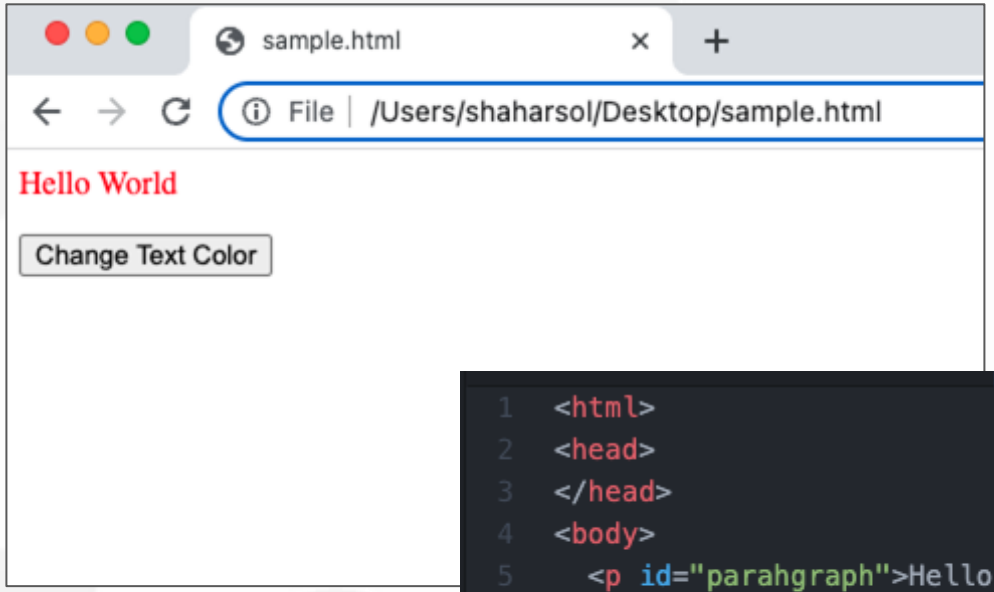
The Javascript origin - the browser




```html
<html>
<head>
</head>
<body>
  <p id="parahgraph">Hello World</p>
  <button onclick="document.getElementById('parahgraph').style.color='red';">
    Change Text Color
  </button>
</body>
```

01 | Introduction to Node.js

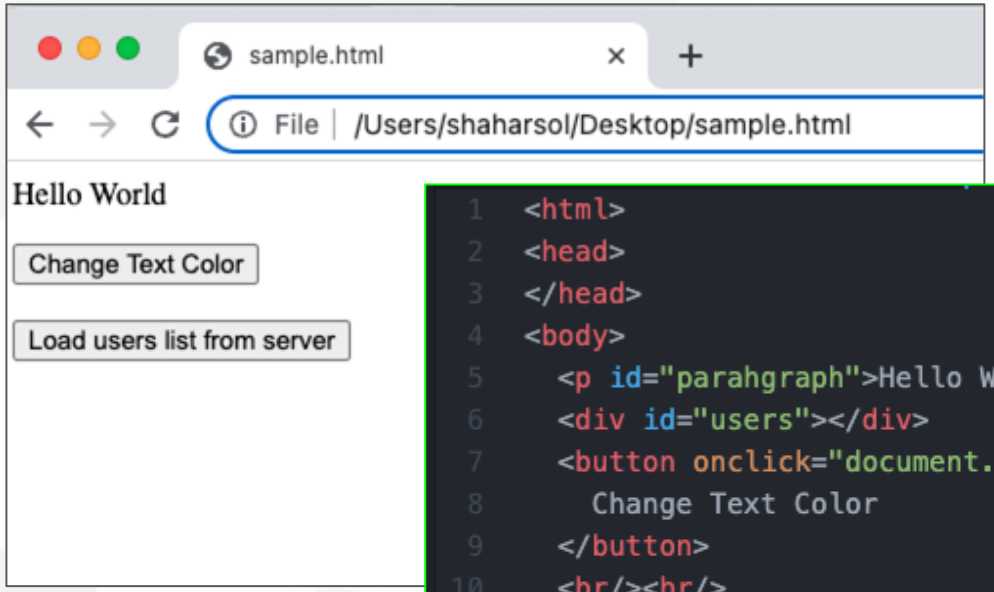
The Javascript origin - the browser




```html
<html>
<head>
</head>
<body>
  <p id="parahgraph">Hello World</p>
  <button onclick="document.getElementById('parahgraph').style.color='red';">
    Change Text Color
  </button>
</body>
```

01 | Introduction to Node.js

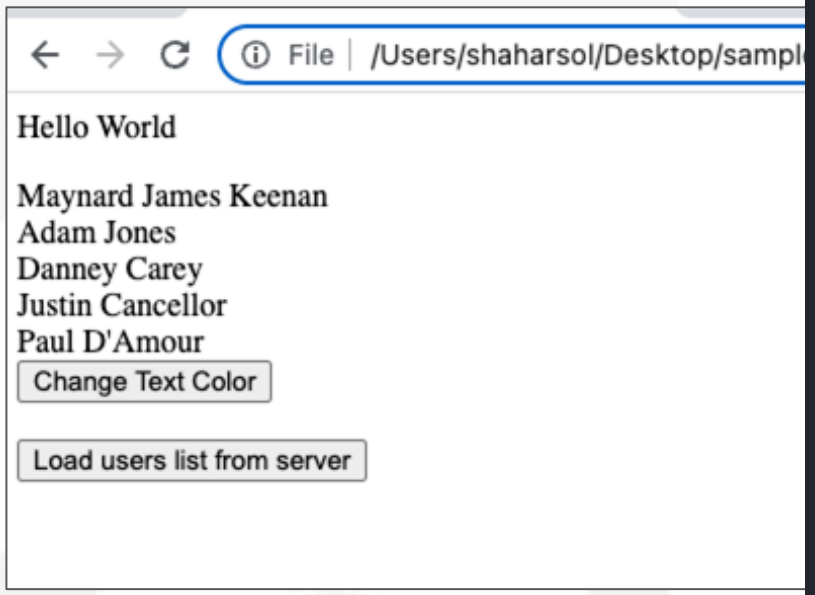
The Javascript origin - the browser




```
<html>
<head>
</head>
<body>
  <p id="parahgraph">Hello World</p>
  <div id="users"></div>
  <button onclick="document.getElementById('parahgraph').style.color='red';">
    Change Text Color
  </button>
  <br/><br/>
  <button onclick="loadUserListFromServer();">
    Load users list from server
  </button>
</body>
```

01 | Introduction to Node.js

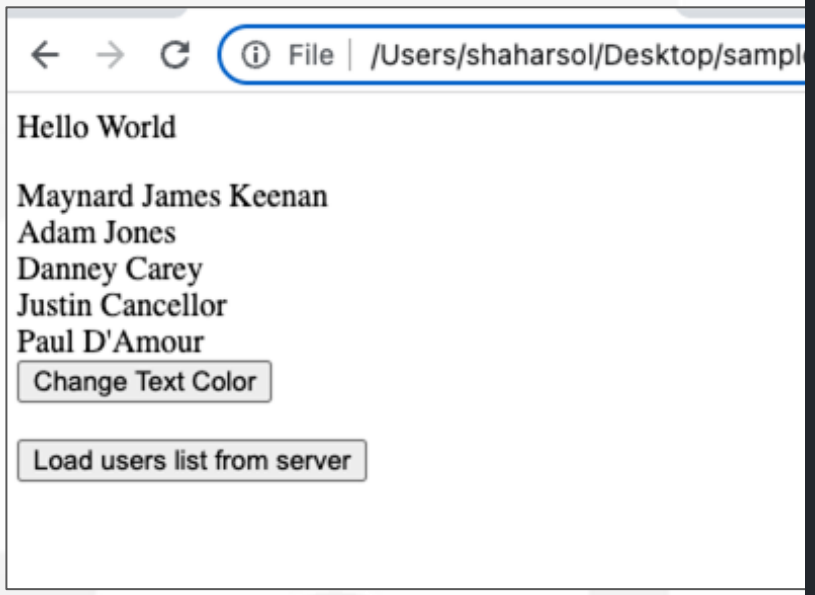
The Javascript origin - the browser




```html
<html>
<head>
  <script type="text/javascript">
    function loadUserListFromServer() {
      serverAPI.getUserList(function(err,users){
        if(err){
          alert(err)
        }else{
          document.getElementById('users').innerHTML = users;
        }
      })
    }
  </script>
</head>
<body>
  <p id="parahgraph">Hello World</p>
  <div id="users"></div>
  <button onclick="document.getElementById('parahgraph').style.color='red';">
    Change Text Color
  </button>
  <br/><br/>
  <button onclick="loadUserListFromServer();">
    Load users list from server
  </button>
</body>
```

01 | Introduction to Node.js

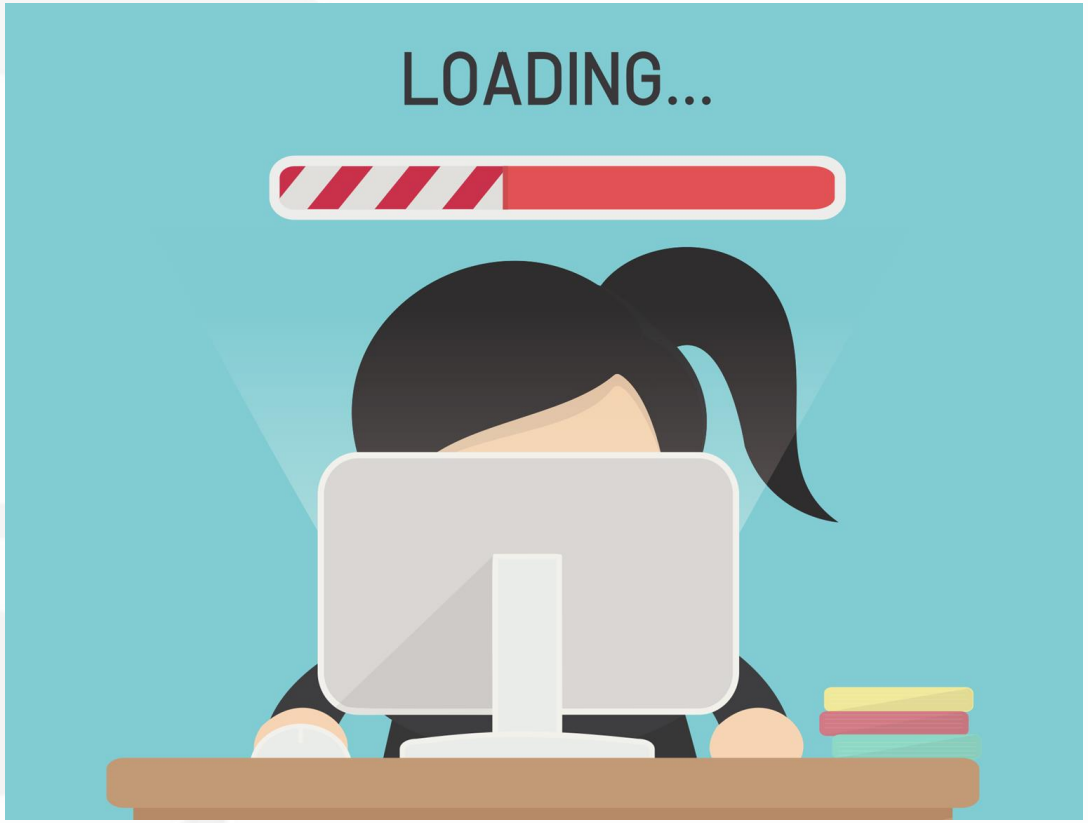
The Javascript origin - the browser




```html
<html>
<head>
  <script type="text/javascript">
    function loadUserListFromServer() {
      serverAPI.getUserList(function(err,users){
        if(err){
          alert(err)
        }else{
          document.getElementById('users').innerHTML = users;
        }
      })
    }
  </script>
</head>
<body>
  <p id="parahgraph">Hello World</p>
  <div id="users"></div>
  <button onclick="document.getElementById('parahgraph').style.color='red';">
    Change Text Color
  </button>
  <br/><br/>
  <button onclick="loadUserListFromServer();">
    Load users list from server
  </button>
</body>
```

01 | Introduction to Node.js

2009. Internet gets big. Apache servers get slow.

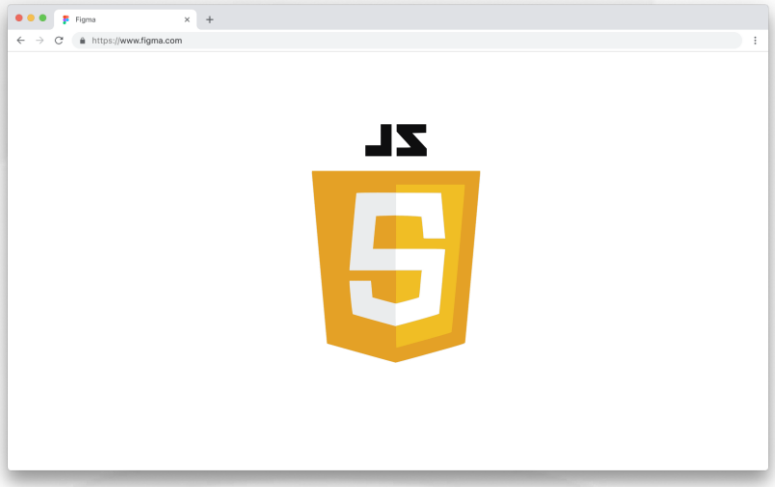


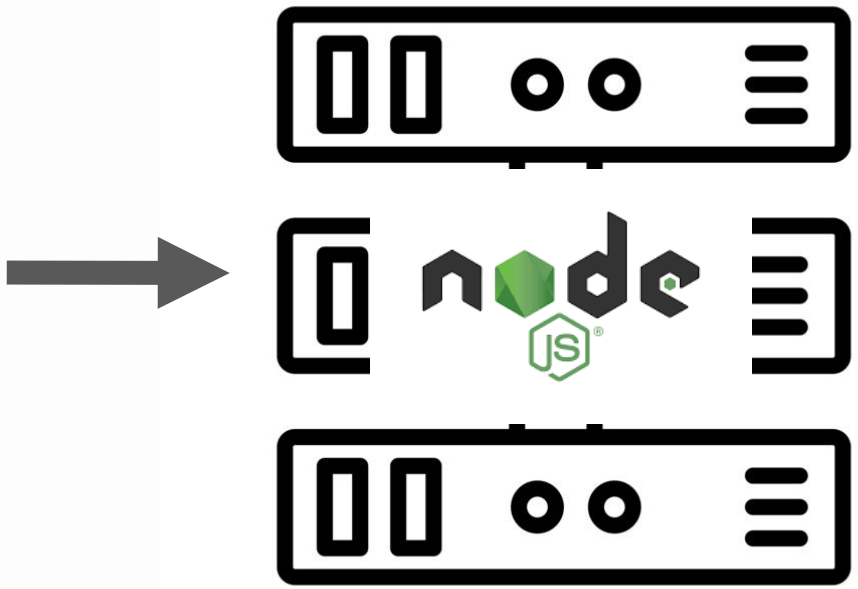
01 | Introduction to Node.js

Ryan Dahl’s idea: Node.js

01 | Introduction to Node.js

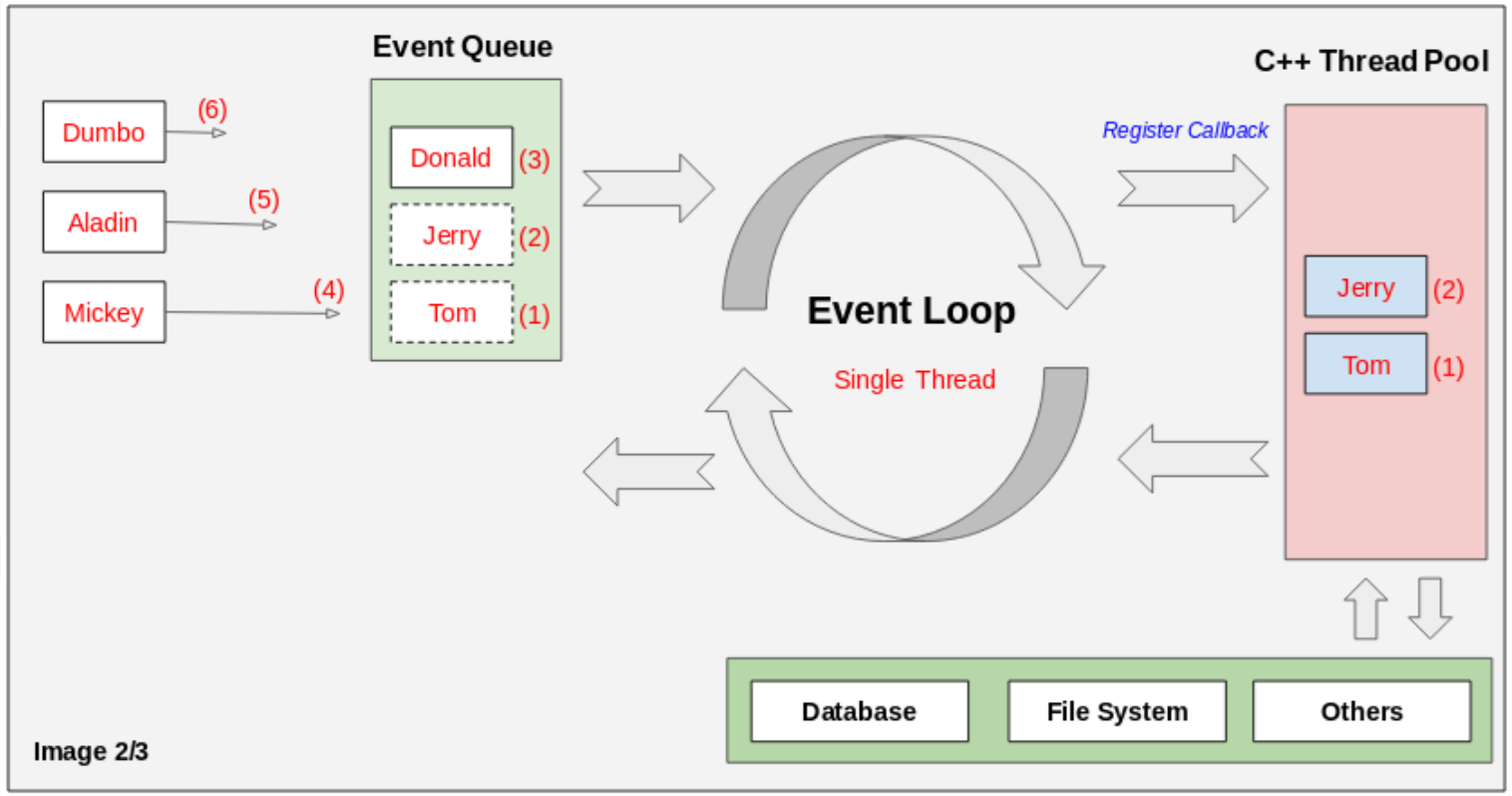
Ryan Dahl’s idea: Node.js





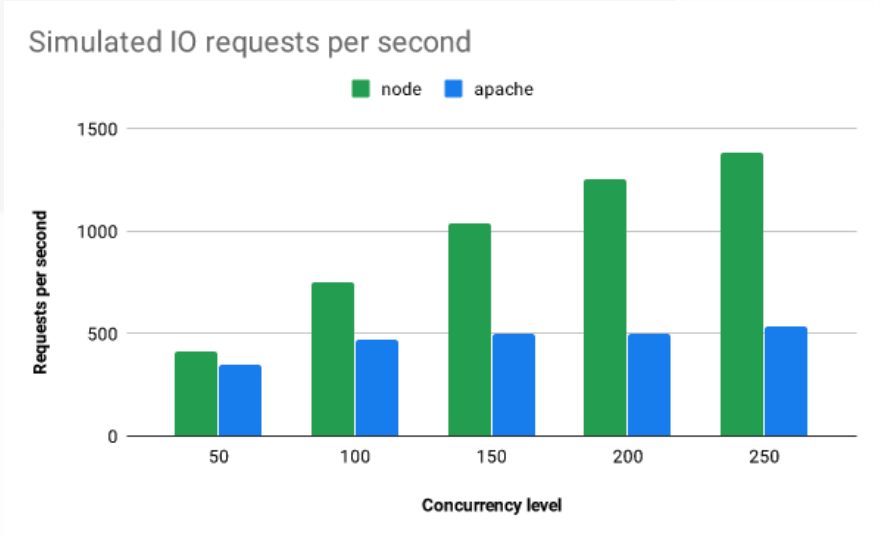
01 | Introduction to Node.js

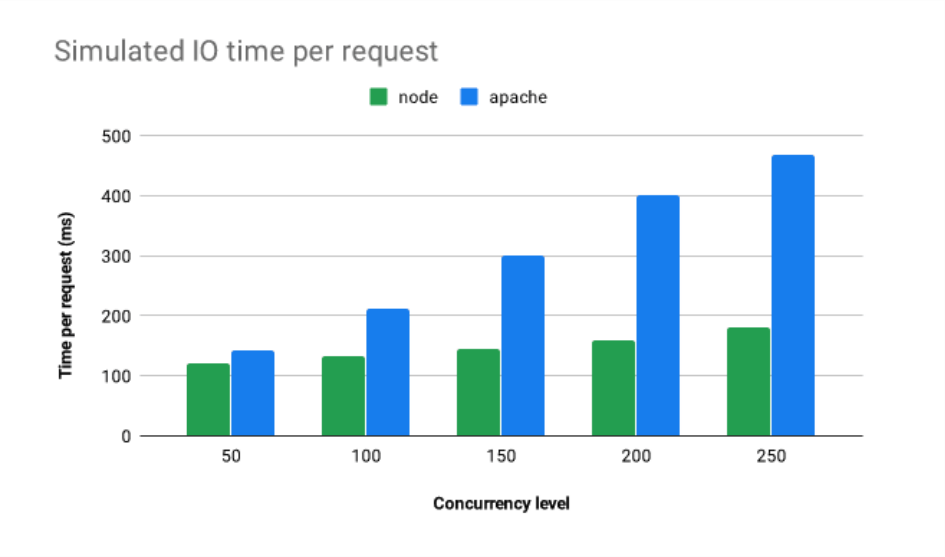
Node.js Event Loop



01 | Introduction to Node.js

Did it work? Hell yeah!





01 | Introduction to Node.js

Let’s play. What we need.

- **node**
- **npm**
- **nvm**
- **npx**

01 | Introduction to Node.js

Let’s play. What we need.

- node –version
- npm –version
- nvm
  - nvm list
  - nvm install
  - nvm use
- npx cowsay helloworld
- a word about nodemon

01 | Introduction to Node.js

Run node

```
➜  ~ node
Welcome to Node.js v18.2.0.
Type ".help" for more information.
>
```

01 | Introduction to Node.js

Run a node program

- **node {a .js file name}**

01 | Introduction to Node.js

Debug a node program

- default debugger
  - node inspect {a .js file name}
    - c
    - n
    - s
    - o
- custom debug client
  - node –inspect {a .js file name}
  - normally your IDE will run this for you

01 | Introduction to Node.js

Hello World Exercise(s)

1. **Hello World**
2. **Hello {name} from params**
3. **Hello {name} from env**
4. **Templates Strings/String Interpolation - `Hello ${name}`**

```
// hints
console.log('message');
process.argv
process.env.NAME
```

02 | Asynchronous Node.js

02 | Asynchronous Node.js

- Three were given to the Elves

02 | Asynchronous Node.js

- Remember the event loop?
- Remember the browser callback example?

```
<html>
<head>
  <script type="text/javascript">
    function loadUserListFromServer() {
      serverAPI.getUserList(function(err,users){
        if(err){
          alert(err)
        }else{
          document.getElementById('users').innerHTML = users;
        }
      })
    }
  </script>
</head>
<body>
  <p id="parahgraph">Hello World</p>
  <div id="users"></div>
  <button onclick="document.getElementById('parahgraph').style.color='red';">
    Change Text Color
  </button>
  <br/><br/>
  <button onclick="loadUserListFromServer();">
    Load users list from server
  </button>
</body>
```

02 | Asynchronous Node.js

1st gen node code: Callback hell (real world example)

```
// get all the original jpegs, and create the edited jpegs.
function reserveWork()
{
  beanstalkd.reserve(function(err, jobid, payload){ reportError(err);
    beanstalkd.bury(jobid, 1024, function(err){ reportError(err);
      var spin = JSON.parse(payload.toString());
      console.dir(spin);
      spin.shortid = spin.short_id;
      var s3key = spin.shortid+"/spin.zip";

      console.log(("["+spin.shortid+"] STARTED (beanjob #"+jobid+")"));


      s3.getObject({Bucket: s3bucket, Key: s3key}, function(err, data) { if(err) console.error("Could not get "+s3key); reportError(err);
        fs.mkdirs(path.dirname(s3key),function(err){ reportError(err);
          fs.writeFile(s3key,data.Body,function(err){ reportError(err);
            // spin.zip is on the file system now
            var cmd = "unzip -o "+s3key+" -d "+path.dirname(s3key);
            exec(cmd,function(){
              console.log("Stuff is unzipped!");

              fs.mkdirs(path.dirname(s3key)+"/orig",function(){
                var vfs = ["null"];
                var rots = [null, "transpose=2", "transpose=2,transpose=2", "transpose=2,transpose=2,transpose=2"];
                var rotidx = parseInt(spin.rotation_angle,10)/90;
                if(rotidx) vfs.push(rots[rotidx]);
                var vf = "-vf "+vfs.join(",");
                var ffmpeg_cmd = "ffmpeg -i "+path.dirname(s3key)+"/cap.mp4 -q:v 1 "+vf+" -pix_fmt yuv420p "+path.dirname(s3key)+"/orig/%03d.jpg";
                exec(ffmpeg_cmd,function(){
                  console.log("Done with ffmpeg");
                  // Upload everything to S3
                  Step( function(){
                    for (var i=1; i<=spin.frame_count; i++)
                    {
                      var s3key = spin.shortid + "/orig/" + ("00"+i).substr(-3) + ".jpg";
                      uploadOrig(s3key, this.parallel());
                    }
                  },
                  function(){
                    fs.readFile(spin.shortid+"/labels.txt",function(err,data){
                      if(err || !data)
                        data = new Buffer("{}");
                      s3.putObject({Bucket: s3bucket, Key: spin.shortid+"/labels.json", ACL: "public-read", ContentType: "text/plain", Body: data}, function(err, data){ reportError(err);
                        console.log("All files are uploaded");
                        beanstalkd.use("editor",function(err,tube){ reportError(err,jobid);
                          beanstalkd.put(1024,0,300,JSON.stringify(spin), function(err,new_jobid){ reportError(err,jobid);
                            console.log("Added new job to beanstalkd.");
                            beanstalkd.destroy(jobid, function(){
                              console.log(("["+spin.shortid+"] FINISHED (beanjob #"+jobid+")"));
                              reserveWork();
                            });
                          });
                        });
                      });
                    });
                  });
                });
              });
            });
          });
        });
      });

    });
  }
```

02 | Asynchronous Node.js

1st gen node code: Callback hell (real world example)

```
60  var projects = db.get('projects');
61  var editors = db.get('editors');
62
63  var s3params;
64  var mediaRet;
65  var metadataRet;
66
67  projects.findOne({
68          _id: payload.project_id
69  },function(err,project){
70          if(err){
71                  console.log('error fteching one project: ' + util.inspect(err));
72          }else{
73
74                  editors.findOne({_id: project.editor_id},function(err,editor){
75                          if(err){
76                                  console.log('error fteching one editor: ' + util.inspect(err));
77                          }else{
78                                  var form = {
79                                          access_token: editor.dropbox.access_token
80                                  };
81
82                                  request('https://api.dropbox.com/1/metadata/auto/' + 'editor/' + project.title + '-' + project._id + '/' + proje
83                                          if(error){
84                                                  console.log('error in dropbox metadata api: ' + util.inspect(error));
85                                          }else{
86                                                  if(response.statusCode != 200){
87                                                          console.log('error in dropbox metadata api: ' + util.inspect(body));
88                                                  }else{
89                                                          metadataRet = JSON.parse(body);
90
91                                                          request.post('https://api.dropbox.com/1/media/auto/' + 'editor/' + project.title + '-' +
92                                                                  if(error){
93                                                                          console.log('error in dropbox media api: ' + util.inspect(error));
94                                                                  }else{
95                                                                          if(response.statusCode != 200){
96                                                                                  console.log('error in dropbox media api: ' + util.inspect(body))
97                                                                          }else{
98
99                                                                                  mediaRet = JSON.parse(body);
100                                                                                 console.log('media url is: ' + mediaRet.url);
101                                                                                 https.get(mediaRet.url, function(httpRes){
102                                                                                         s3params = {
103                                                                                                 Bucket: AWS_BUCKET,
104                                                                                                 Key: project._id + '/' + project.final_cut.file_
105                                                                                                 Body: httpRes.
```

02 | Asynchronous Node.js

Same code implemented with Promises

```
68      projects.findOne({_id: payload.project_id})
69      .then(project => {
70          return editors.findOne({_id: project.editor_id})
71      })
72      .then(editor => {
73          return request('https://api.dropbox.com/1/metadata/auto/' + 'editor/' + project.title + '-' + project._id + '/' + project.final_cut.file_name + '?access_
74      })
75      .then(dropboxMetadataResponse => {
76          return request.post('https://api.dropbox.com/1/media/auto/' + 'editor/' + project.title + '-' + project._id + '/' + project.final_cut.file_name,{form: fo
77      })
78      .then(dropboxMediaResponse => {
79          return pipeToS3(dropboxMediaResponse)
80      })
81      .catch(err => {
82          console.log(err)
83      })
84      .finally(() => {
85          consloe.log('finsihed!')
86      })
```

02 | Asynchronous Node.js

What is a Promise?

The Promise object represents the **eventual completion (or failure)** of an asynchronous operation and its resulting value.

02 | Asynchronous Node.js

Basic Promise example

```
18
19    const data = getDataFromServer();
20    console.log(data);
21
```

02 | Asynchronous Node.js

Basic Promise example

```
18
19    const data = getDataFromServer();
20    console.log(data);
21
```

```
➜  Desktop node pr.js
Promise { <pending> }
➜  Desktop
```

02 | Asynchronous Node.js

Promise statuses

```
19  const data = getDataFromServer();
20  console.log(data);
```

```
→  Desktop node pr.js
Promise { <pending> }
→  Desktop
```

```
→  Desktop node pr.js
Promise { 'Hello World!\n' }
→  Desktop
```

```
→  Desktop node pr.js
Promise {
  <rejected> [Error: ENOENT: no such file or directory, open '/Uses/Shaharsol/helloworld.txt'] {
    errno: -2,
    code: 'ENOENT',
    syscall: 'open',
    path: '/Uses/Shaharsol/helloworld.txt'
  }
}
```

02 | Asynchronous Node.js

Basic Promise example

```
11  const data = getDataFromServer();
12  data.then((results) => {
13      console.log(results);
14  })
15
16  console.log(data)
```

02 | Asynchronous Node.js

Basic Promise example

```
11  const data = getDataFromServer();
12  data.then((results) => {
13      console.log(results);
14  })
15
16  console.log(data)
```

```
→  Desktop node pr.js
Promise { <pending> }
Hello World!
```

02 | Asynchronous Node.js

How did it work?

```
5  const fs = require('fs');
6
7  const getDataFromServer = () => {
8      return new Promise((resolve, reject) => {
9          fs.readFile('/Users/shaharsol/helloworld.txt',(err, data) => {
10             if (err) {
11                 reject(err);
12             } else {
13                 resolve(data.toString());
14             }
15         })
16     })
17 }
18
19 const data = getDataFromServer()
20 data.then(fileContents => {
21     console.log(fileContents);
22 })
23
24 console.log(data);
```

02 | Asynchronous Node.js

How did it work?

```
5  const fs = require('fs');
6
7  const getDataFromServer = () => {
8      return new Promise((resolve, reject) => {
9          fs.readFile('/Users/shaharsol/helloworld.txt',(err, data) => {
10             if (err) {
11                 reject(err);
12             } else {
13                 resolve(data.toString());
14             }
15         })
16     })
17 }
18
19 const data = getDataFromServer()
20 data.then(fileContents => {
21     console.log(fileContents);
22 })
23
24 console.log(data);
```

02 | Asynchronous Node.js

Promise rejection

```
7  const getDataFromServer = () => {
8      return new Promise((resolve, reject) => {
9          fs.readFile('/Some/Non/Existing/File.txt',(err, data) => {
10             if (err) {
11                 reject(err);
12             } else {
13                 resolve(data.toString());
14             }
15         })
16     })
17 }
18
19 const data = getDataFromServer()
20 data.then(fileContents => {
21     console.log(fileContents);
22 })
23 .catch(err => {
24     console.log(err);
25 })
26
27 console.log(data);
```

02 | Asynchronous Node.js

Promise rejection

```
7  const getDataFromServer = () => {
8      return new Promise((resolve, reject) => {
9          fs.readFile('/Some/Non/Existing/File.txt',(err, data) => {
10             if (err) {
11                 reject(err);
12             } else {
13                 resolve(data.
14             }
15         })
16     })
17 }
18
19 const data = getDataFromServe
20 data.then(fileContents => {
21     console.log(fileContents)
22 })
23 .catch(err => {
24     console.log(err);
25 })
26
27 console.log(data);
```

```
→  Desktop node pr.js
Promise { <pending> }
[Error: ENOENT: no such file or directory, open '/Some/Non/Existing/File.txt'] {
  errno: -2,
  code: 'ENOENT',
  syscall: 'open',
  path: '/Some/Non/Existing/File.txt'
}
→  Desktop
```

02 | Asynchronous Node.js

Promise chaining

```
68      projects.findOne({_id: payload.project_id})
69      .then(project => {
70          return editors.findOne({_id: project.editor_id})
71      })
72      .then(editor => {
73          return request('https://api.dropbox.com/1/metadata/auto/' + 'editor/' + project.title + '-' + project._id + '/' + project.final_cut.file_name + '?access_
74      })
75      .then(dropboxMetadataResponse => {
76          return request.post('https://api.dropbox.com/1/media/auto/' + 'editor/' + project.title + '-' + project._id + '/' + project.final_cut.file_name,{form: fo
77      })
78      .then(dropboxMediaResponse => {
79          return pipeToS3(dropboxMediaResponse)
80      })
81      .catch(err => {
82          console.log(err)
83      })
84      .finally(() => {
85          consloe.log('finsihed!')
86      })
```

02 | Asynchronous Node.js

Promise concurrency

```
37  p1 = getUsers();
38  p2 = getOrders();
39  p3 = getInventory();
40
41  Promise.all([p1, p2, p3])
42  .then(results=>{
43      console.log(results);
44  })
45  .catch(err => {
46      console.log(err);
47  })
```

02 | Asynchronous Node.js

Promise concurrency

```
19  const getUsers = () => {
20      return new Promise((resolve, reject) => {
21          resolve(['Maynard James Keenan', 'Adam Jones', 'Danney Care
22      })
23  }
24
25  const getOrders = () => {
26      return new Promise((resolve, reject) => {
27          resolve(['Order1', 'Order2', 'Order3'])
28      })
29  }
30
31  const getInventory = () => {
32      return new Promise((resolve, reject) => {
33          resolve(['Item1', 'Item2', 'Item3', 'Item4'])
34      })
35  }
36
37  p1 = getUsers();
38  p2 = getOrders();
39  p3 = getInventory();
40
41  Promise.all([p1, p2, p3])
42  .then(results=>{
43      console.log(results);
44  })
45  .catch(err => {
46      console.log(err);
47  })
```

```
→  Desktop node pr.js
[
  [
    'Maynard James Keenan',
    'Adam Jones',
    'Danney Carey',
    'Justin Cancellor',
    "Paul D'Amour"
  ],
  [ 'Order1', 'Order2', 'Order3' ],
  [ 'Item1', 'Item2', 'Item3', 'Item4' ]
]
→  Desktop
```

02 | Asynchronous Node.js

Promise concurrency

```
19  const getUsers = () => {
20      return new Promise((resolve, reject) => {
21          resolve(['Maynard James Keenan', 'Adam Jones', 'Danney Carey', 'Justin Cancellor', 'Paul D\'Amour'])
22      })
23  }
24
25  const getOrders = () => {
26      return new Promise((resolve, reject) => {
27          resolve(['Order1', 'Order2', 'Order3'])
28      })
29  }
30
31  const getInventory = () => {
32      return new Promise((resolve, reject) => {
33          reject('Empty Inventory!!!')
34      })
35  }
36
37  p1 = getUsers();
38  p2 = getOrders();
39  p3 = getInventory();
40
41  Promise.all([p1, p2, p3])
42  .then(results=>{
43      console.log(results);
44  })
45  .catch(err => {
46      console.log(err);
47  })
```

```
➜  Desktop node pr.js
Empty Inventory!!!
➜  Desktop
```

02 | Asynchronous Node.js

- Promise.any() - fulfills when any of the input's promises fulfills, with this first fulfillment value. It rejects when all of the input's promises reject
- Promise.allSettled() - fulfills when all of the input's promises settle (including when an empty iterable is passed), with an array of objects that describe the outcome of each promise.
- Promise.race() - This returned promise settles with the eventual state of the first promise that settles.

02 | Asynchronous Node.js

ES2017: This is still not human readable

```
68      projects.findOne({_id: payload.project_id})
69      .then(project => {
70          return editors.findOne({_id: project.editor_id})
71      })
72      .then(editor => {
73          return request('https://api.dropbox.com/1/metadata/auto/' + 'editor/' + project.title + '-' + project._id + '/' + project.final_cut.file_name + '?access_
74      })
75      .then(dropboxMetadataResponse => {
76          return request.post('https://api.dropbox.com/1/media/auto/' + 'editor/' + project.title + '-' + project._id + '/' + project.final_cut.file_name,{form: fo
77      })
78      .then(dropboxMediaResponse => {
79          return pipeToS3(dropboxMediaResponse)
80      })
81      .catch(err => {
82          console.log(err)
83      })
84      .finally(() => {
85          consloe.log('finsihed!')
86      })
```

02 | Asynchronous Node.js

Async/await

```
68      const workflow = async () => {
69          try {
70              const project = await projects.findOne({_id: payload.project_id});
71              const editor = await editors.findOne({_id: project.editor_id});
72              const dropboxMetadataResponse = await request('https://api.dropbox.com/1/metadata/auto/' + 'editor/' + project.title + '-' + project._id + '/' + proj
73              const dropboxMediaResponse = await request.post('https://api.dropbox.com/1/media/auto/' + 'editor/' + project.title + '-' + project._id + '/' + proje
74              await pipeToS3(dropboxMediaResponse)
75          } catch (err) {
76              console.log(err);
77          }
78      }
```

02 | Asynchronous Node.js

Async/await

```
68      const workflow = async () => {
69          try {
70              const project = await projects.findOne({_id: payload.project_id});
71              const editor = await editors.findOne({_id: project.editor_id});
72              const dropboxMetadataResponse = await request('https://api.dropbox.com/1/metadata/auto/' + 'editor/' + project.title + '-' + project._id + '/' + proj
73              const dropboxMediaResponse = await request.post('https://api.dropbox.com/1/media/auto/' + 'editor/' + project.title + '-' + project._id + '/' + proje
74              await pipeToS3(dropboxMediaResponse)
75          } catch (err) {
76              console.log(err);
77          }
78      }
```

02 | Asynchronous Node.js

async/await example

```
5   const getDataFromServer = () => {
6       return new Promise((resolve, reject) => {
7           resolve('data from server')
8       })
9   }
10
11  const workflow = async () => {
12      try {
13          const data = getDataFromServer()
14          console.log(data)
15      } catch (err) {
16          console.log(err)
17      }
18  }
19
20  workflow();
```

02 | Asynchronous Node.js

async/await example

```
5  const getDataFromServer = () => {
6      return new Promise((resolve, reject) => {
7          resolve('data from server')
8      })
9  }
10
11 const workflow = async () => {
12     try {
13         const data = getDataFromServer()
14         console.log(data)
15     } catch (err) {
16         console.log(err)
17     }
18 }
19
20 workflow();
```

```
➜  Desktop node pr.js
Promise { 'data from server' }
➜  Desktop
```

02 | Asynchronous Node.js

async/await example

```
5   const getDataFromServer = () => {
6       return new Promise((resolve, reject) => {
7           resolve('data from server')
8       })
9   }
10
11  const workflow = async () => {
12      try {
13          const data = await getDataFromServer()
14          console.log(data)
15      } catch (err) {
16          console.log(err)
17      }
18  }
19
20  workflow();
```

```
→  Desktop node pr.js
Promise { 'data from server' }
→  Desktop
```

Don’t forget to await…

02 | Asynchronous Node.js

Async function return value

```
82  const someAsyncFunction = async () => {
83      return 1;
84  }
85
86  const result = someAsyncFunction();
87  console.log(result);
```

02 | Asynchronous Node.js

Async function return value

```
82  const someAsyncFunction = async () => {
83      return 1;
84  }
85
86  const result = someAsyncFunction();
87  console.log(result);
```

```
→  Desktop node pr.js
Promise { 1 }
→  Desktop
```

02 | Asynchronous Node.js

Async function return value is always a promise

```
82  const someAsyncFunction = async () => {
83      return 1;
84  }
85
86  (async () => {
87      const result = await someAsyncFunction();
88      console.log(result);
89  })();
90
```

```
→  Desktop node pr.js
1
→  Desktop
```

02 | Asynchronous Node.js

Async exercise - use this server-mock-function to calc how many candles are lit in Hanukkah, in 3 different ways…

```
const howManyCandlesCallback = (dayNumber, callback) => {
    if ( dayNumber < 1 ) {
        return callback ('day cannot be smaller than 1');
    }

    if ( dayNumber > 8 ) {
        return callback ('No Isro Chag for Hannukah!');
    }

    return callback ( null, dayNumber + 1 );
}
```

- **using callbacks**
- **using then**
- **using async/await**

02 | Asynchronous Node.js

Async exercise - now use this more realistic server-mock-function and see if you were correct…

```
const howManyCandlesCallback = (dayNumber, callback) => {
    return setTimeout(() => {
        if ( dayNumber < 1 ) {
            return callback ('day cannot be smaller than 1');
        }

        if ( dayNumber > 8 ) {
            return callback ('No Isro Chag for Hannukah!');
        }

        return callback ( null, dayNumber + 1 );
    }, (Math.random() + 1 ) * 1000);

};
```

03 | File System with Node.js

03 | File System with Node.js

Now that we know how to work with async, let’s do some IO!

03 | File System with Node.js

Remember this? Node adds some essential built-in modules on top of native JavaScript

```
const fs = require('fs');
```

03 | File System with Node.js

Some other important node modules

```
const http = require('http');    // handle http, server, request, response, mime types etc
const fs = require('fs')         // enables interacting with the file system
const os = require('os')         // operating system-related utility methods and properties
const path = require('path')     // utilities for working with file and directory paths
const url = require('url')       // utilities for URL resolution and parsing
const util = require('util')     // various utilities
```

03 | File System with Node.js

We'll use the **fs** node module

```
const fs = require('fs');
// or
import * as fs from 'node:fs';
// which to use?
```

03 | File System with Node.js

We'll use the **fs** node module

REQUIRE	ES6 IMPORT AND EXPORT
Require is Non-lexical, it stays where they have put the file.	Import is lexical, it gets sorted to the top of the file.
It can be called at any time and place in the program.	It can't be called conditionally, it always run in the beginning of the file.
You can directly run the code with require statement.	To run a program containing import statement you have to use experimental module feature flag.
If you want to use require module then you have to save file with '.js' extension.	If you want to use import module then you have to save file with '.mjs' extension.

03 | File System with Node.js

For simplicity, we'll stick with **require** for this course

03 | File System with Node.js

Let’s read a local file contents

1. using readFile (callback)
2. using promises (with either then or async/await)
3. using readFileSync

```
// hints
readFile('file name', 'encoding', callback)
const fs = require('fs/promises')
readFileSync('file name', 'encoding')
```

03 | File System with Node.js

readFileSync is blocking!!!

all fs.xxxSync methods are blocking!!!

use only in CLI!!!

03 | File System with Node.js

use **path** module to be OS agnostic

```
const fs = require('fs/promises');
const path = require('path');

(async () => {
    const data = await fs.readFile(path.join('inner-directory','content.txt'), 'utf8')
    console.log(data);
})();
```

03 | File System with Node.js

create a directory

```
const fs = require('fs/promises');

(async () => {
    await fs.mkdir('new-dir')
})();
```

03 | File System with Node.js

Some more fs functions

- chown
- rmdir
- unlink
- etc… https://nodejs.org/api/fs.html

04 | Basic Web Server with Node.js

04 | Basic Web Server with Node.js

Now that we know how to work with async, and files… let's build a web server!

But first, let's have Postman installed.

- postman = curl with GUI
- easy manipulation of
  - verb
  - input
  - content type
- presentation of
  - response
  - status

04 | Basic Web Server with Node.js

A minimal Node.js **native** web server

```
require('http').createServer((req, res) => {
    res.writeHead(200).end('My first minimal node web server!');
}).listen('8080', 'localhost');
```

04 | Basic Web Server with Node.js

A minimal Node.js **native** web server

```
const http = require('http');

const HOST = 'localhost';
const PORT = '8080';

const requestListener = function (req, res) {
    res.writeHead(200);
    res.end("My first node web server!");
};

const server = http.createServer(requestListener);

server.listen(PORT, HOST, () => {
    console.log(`Server is running on http://${HOST}:${PORT}`);
});
```

04 | Basic Web Server with Node.js

We can require only what we need (deconstruction)…

```
const { createServer } = require('http');

const HOST = 'localhost';
const PORT = '8080';

const requestListener = function (req, res) {
    res.writeHead(200);
    res.end("My first node web server!");
};

const server = createServer(requestListener);

server.listen(PORT, HOST, () => {
    console.log(`Server is running on http://${HOST}:${PORT}`);
});
```

04 | Basic Web Server with Node.js

Which style do you prefer?

```
const http = require('http');

const HOST = 'localhost';
const PORT = '8080';

const requestListener = function (req, res) {
    res.writeHead(200);
    res.end("My first node web server!");
};

const server = http.createServer(requestListener);

server.listen(PORT, HOST, () => {
    console.log(`Server is running on http://${HOST}:${PORT}`);
});
```

```
const { createServer } = require('http');

const HOST = 'localhost';
const PORT = '8080';

const requestListener = function (req, res) {
    res.writeHead(200);
    res.end("My first node web server!");
};

const server = createServer(requestListener);

server.listen(PORT, HOST, () => {
    console.log(`Server is running on http://${HOST}:${PORT}`);
});
```

04 | Basic Web Server with Node.js

Let’s return a json object

```
{ schoolName: 'John Bryce' }

// hint: use res.setHeader(name, value);
// 2nd hint: fail and find out.
```

04 | Basic Web Server with Node.js

Now let's return CSV output

A	B	C
id	name	age
123456789	Israel Israeli	50

04 | Basic Web Server with Node.js

Now let's return CSV file

```
// hint
res.setHeader("Content-Disposition", "attachment;filename=johnbryce.csv");
```

04 | Basic Web Server with Node.js

Multi endpoints server

- ***/age*** should return “age endpoint”
- ***/name*** should return “name endpoint”
- **any other request** should return 404

```
// hint
console.log(req);
```

04 | Basic Web Server with Node.js

Multi endpoints/methods server

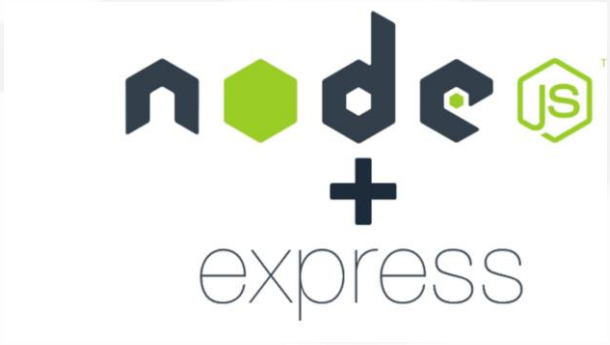
- **GET /age** should return “age endpoint”
- **POST /name** should return “name endpoint”
- **any other request** should return 404

```
// hint
console.log(req);
```

04 | Basic Web Server with Node.js

At this point, you should ask yourself

- **So our entire server is actually an requestListener huge function?**
- **Don’t we have some framework to standardize and make sever development more streamlined?**



05 | NPM

05 | NPM

External modules.

05 | NPM

Write a script that generates a 5 letters code for a bit.ly like system.

example: aSdTy

```
// hint
Math.random()
Math.round()
```

05 | NPM

The same with **chance.js** npm module:

```
const Chance = require('chance');
const chance = new Chance();

console.log(chance.string({
    length: 5,
    pool: 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
}));
```

05 | NPM

Native http GET:

```
const https = require('https');

let request = https.get('https://jsonplaceholder.typicode.com/users?_limit=2', (res) =>
  let data = '';

  res.on('data', (chunk) => {
    data += chunk;
  });

  res.on('close', () => {
    console.log('Retrieved all data');
    console.log(JSON.parse(data));
  });
});
```

05 | NPM

Axios npm module http GET:

```
const axios = require('axios');

(async () => {
  const response = await axios('https://jsonplaceholder.typicode.com/users?_limit=2');
  console.log(response.data);
})()
```

05 | NPM

Conclusion:

- npm modules come in very handy
- 1000s of modules for any niche requirement (explore https://npmjs.com)
- use at own risk… (e.g. https://therecord.media/malware-found-in-npm-package-with-millions-of-weekly-downloads)

05 | NPM

Installing npm modules:

- globally: npm install -g [module name]
- locally: cd into a directory and…
  - npm install [module name]
  - npm install –save [module name]
  - npm install –save-dev [module name]
  - look at package.json

05 | NPM

Some [of my] favorite modules

- **axios / node-fetch** - http operations
- **config** / **dotenv** - configuration
- **winston** - logging (you don’t really console.log in prod…)
- **moment** - date/time manipulation (deprecated, can use **dayjs** instead)
- **joi** - input validation
- **aws-sdk** - self explanatory…
- **express** - the most popular nodejs server framework
- **nestjs** - express successor for TypeScript (depends on **express**)

05 | NPM

Real world example

```
"dependencies": {
  "axios": "^1.3.2",
  "cls-hooked": "^4.2.2",
  "config": "^3.3.9",
  "ejs": "^3.1.8",
  "express": "^4.18.2",
  "http-errors": "^2.0.0",
  "joi": "^17.8.3",
  "moment": "^2.29.4",
  "morgan": "^1.10.0",
  "pg": "^8.9.0",
  "sequelize": "^6.28.0",
  "sequelize-cli": "^6.6.0",
  "stripe": "^11.12.0",
  "uuid": "^9.0.0"
},
"devDependencies": {
  "eslint": "^8.34.0",
  "eslint-config-airbnb-base": "^15.0.0",
  "eslint-plugin-import": "^2.27.5"
}
```

05 | NPM

NPM versioning

- `version` Must match `version` exactly
- `>version` Must be greater than `version`
- `>=version` etc
- `<version`
- `<=version`
- `~version` "Approximately equivalent to version" See semver
- `^version` "Compatible with version" See semver
- `1.2.x` 1.2.0, 1.2.1, etc., but not 1.3.0
- `http://...` See 'URLs as Dependencies' below
- `*` Matches any version
- `""` (just an empty string) Same as `*`
- `version1 - version2` Same as `>=version1 <=version2`.
- `range1 || range2` Passes if either range1 or range2 are satisfied.
- `git...` See 'Git URLs as Dependencies' below
- `user/repo` See 'GitHub URLs' below
- `tag` A specific version tagged and published as `tag` See `npm dist-tag`
- `path/path/path` See Local Paths below

06 | express Framework

06 | express Framework

And now… express

06 | express Framework

express hello world

```
const express = require('express')
const app = express()
const port = 3000
const host = 'localhost';

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

app.listen(port, host, () => {
  console.log(`Example app listening on port ${port}`)
})
```

06 | express Framework

http verbs. Implement:

1. POST /user
2. DELETE /ticket
3. PATCH /employee
4. PUT /organization

```
// hint
app.{method}('/path', () => {})
```

06 | express Framework

Path params Implement:

1. POST /user/{id} (meaningless in reality…)
2. DELETE /ticket/{id}
3. PATCH /employee/{id}
4. PUT /organization/{id}

```
// hint
app.{method}('/path/:param', () => {})
console.log(req)
```

06 | express Framework

Query params. Implement:

1. GET /?id={whatever}

print {whatever}

```
// hint
console.log(req.query)
```

06 | express Framework

post body. Implement:

1. POST / data ”id=whatever”

print {whatever}

```
// hint
console.log(req)
```

06 | express Framework

Couldn’t find it, did ya?

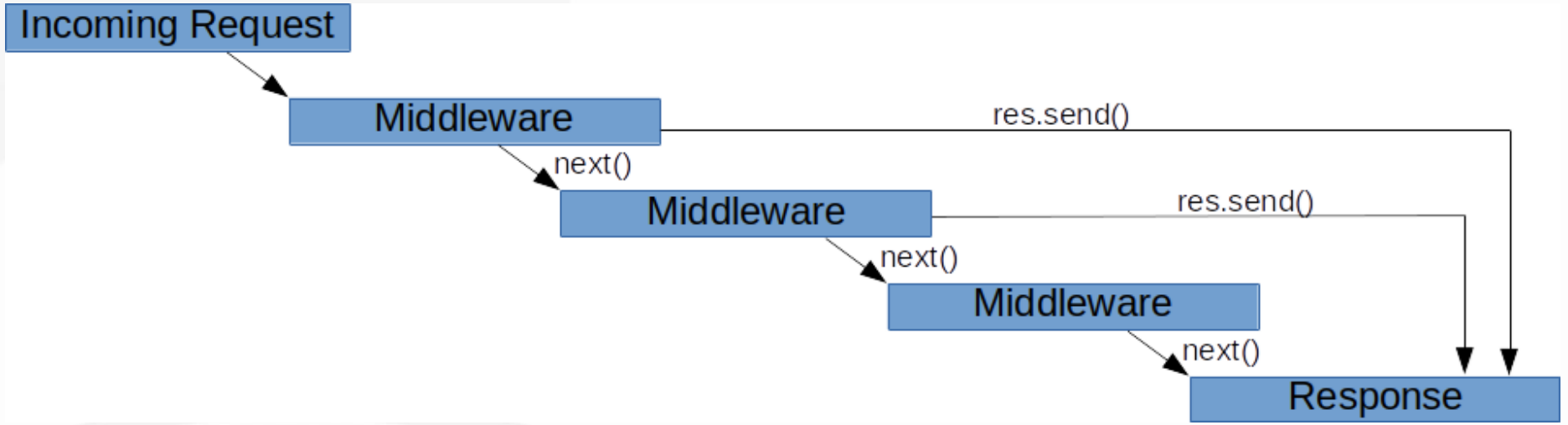
It’s hidden somewhere in the stream…

urlencoded middleware to the rescue…

```
app.use(express.urlencoded({extended: false}));
```

06 | express Framework

Middleware??? what is a middleware?



06 | express Framework

- An express app is a collection of chained middlewares
- middle - between request and response
- cycle starts with incoming request
- ends with response.send
- anything in between is middleware
- middleware has access to request, response, and next middleware
- middleware order is important and sequential (determines next)
- special middleware: error middleware, put those last…
- can filter middlewares by path
- better with Routers

06 | express Framework

- some provided middlewares (by expressjs):
  - express.json (accept application/json)
  - express-session
  - cookie-parser
  - morgan (logging)
  - serve-static (serve static files - better use CDN though…)
  - more…

06 | express Framework

We can create our own middlewares

```
const connectDB = (req, res, next) => {
    // connect to DB
    if (err) {
        return next(err)
    }

    req.db = db // now will be available in other middlewares
    return next()
}

module.exports = connectDB;
```

06 | express Framework

Exercise: create an auth middleware

- request must have Authorization header
- value structure: Bearer {token}
- allow where token === ‘123’
- respond with 401 otherwise
- where is the right place for this middleware in the order?

```
// hints
console.log(req.headers)
res.status(401)
res.end()
res.send()
```

06 | express Framework

Real world example: auth middleware for bearer token (oauth standard)

```
const db = require('../models');

const auth = async (req, res, next) => {
  const unauthorized = () => {
    next({
      code: 401,
      message: 'unauthorized',
    });
  };

  const header = req.headers.authorization; // struct is Bearer token

  if (!header) {
    unauthorized();
  }
  const parts = header.split(' ');
  const apiKey = parts[1];

  if (!apiKey) {
    unauthorized();
  }

  const shop = await db.Shop.findOne({ where: { apiKey } });
  if (shop) {
    req.shop = shop;
    next();
  } else {
    unauthorized();
  }
};

module.exports = auth;
```

06 | express Framework

Exercise: create a middleware to log all incoming POST requests statuses.
Where should it be placed in the order? think carefully ;-)

```
// hints
req.method
res.on('finish', () => {})
```

06 | express Framework

Error middleware

1. gets 4 params (err, req, res, next)
2. invoked only by next(err) from another middleware
3. don’t forget to next(err) from your catch blocks!!!

```
const errorHandler = (error, req, res, next) => {
    res.status(error.status || 400).send(error);
};

module.exports = errorHandler;
```

06 | express Framework

Error middleware exercise:

1. enforce naive input validation
2. fail with 400 on error

06 | express Framework

Think: How to implement a 404 error handler?

06 | express Framework

Summary exercise:
create the following app:

1. auth with bearer 123
2. read users data from https://jsonplaceholder.typicode.com/users
3. filter out according to pagination params (e.g. 2,2)
4. return either json or XML depending on query string tag
5. allow only GET ‘/users’ endpoint
6. return 404 otherwise
7. example request: /?format=xml&offset=2&limit=2

0x | From here on, we're building an app

0x | A crypto dashboard app

Enables users to monitor online rates of their favorite crypto coins.

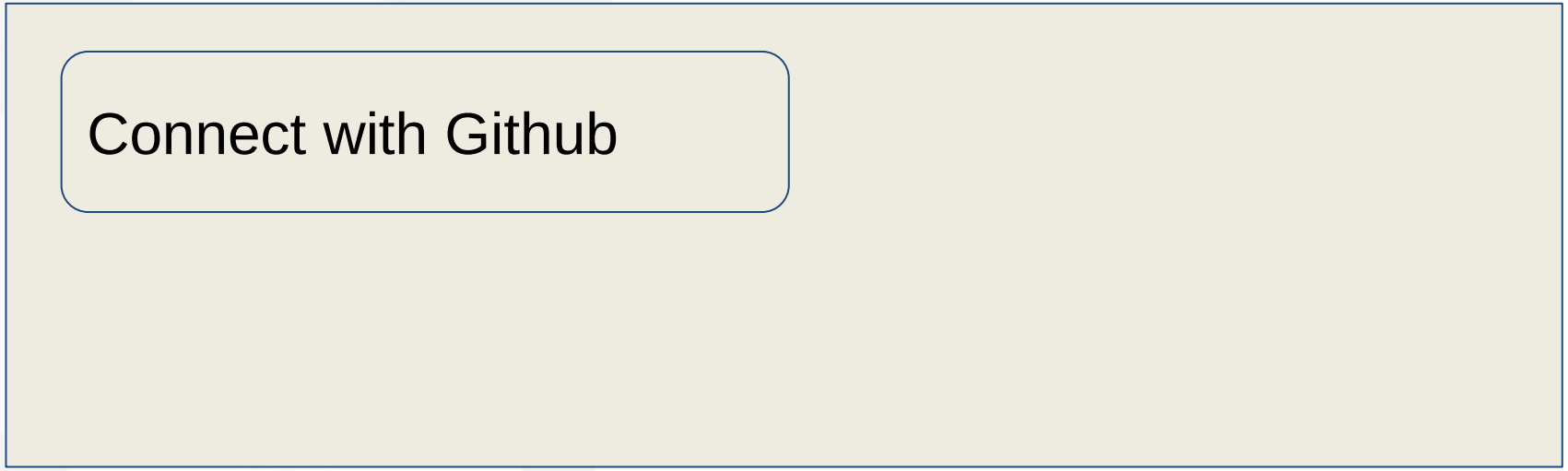
usecases:

- signup (using Github)
- add symbol
- monitor symbols

0x | A crypto dashboard app

views:

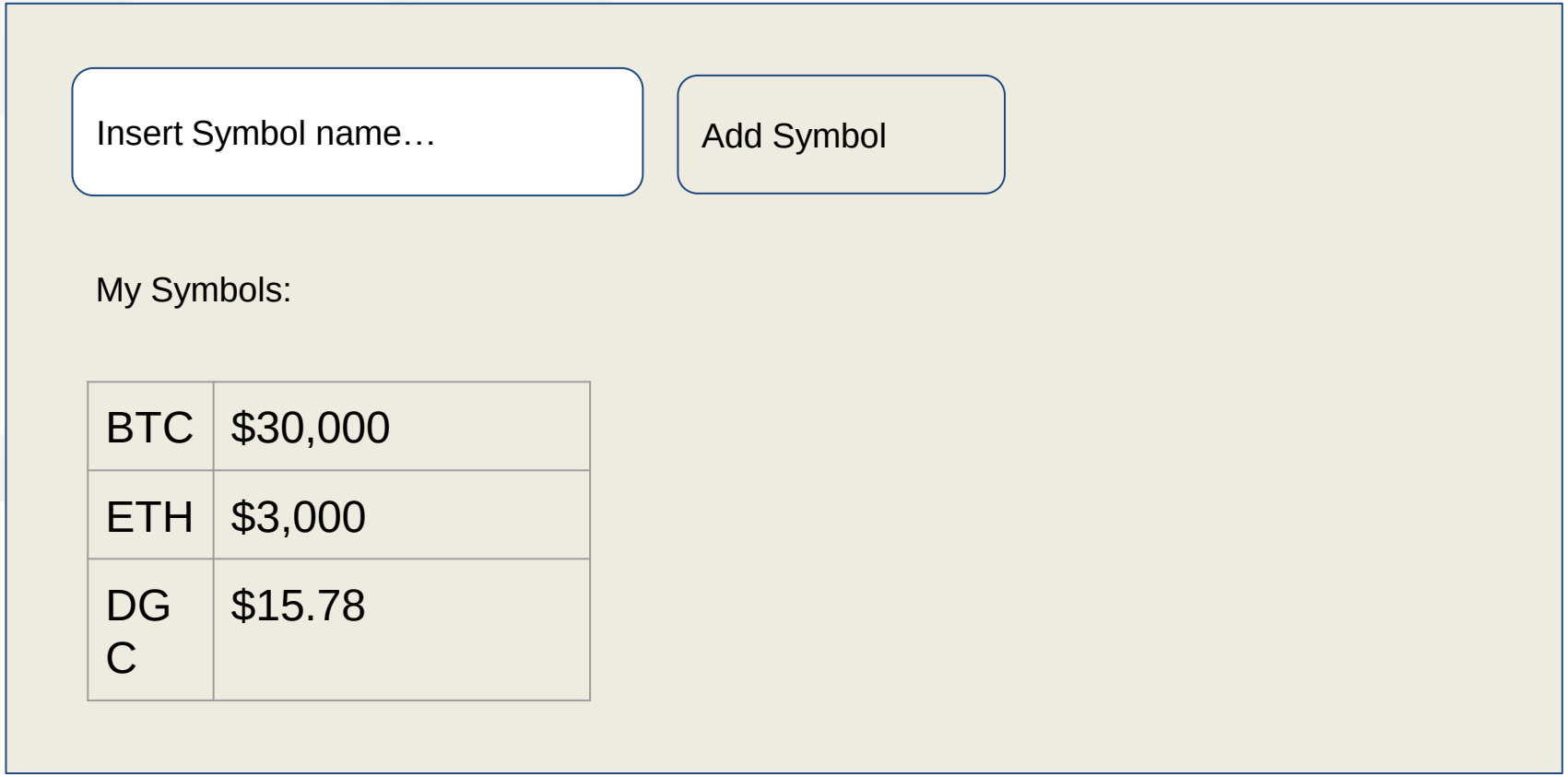
welcome:



0x | A crypto dashboard app

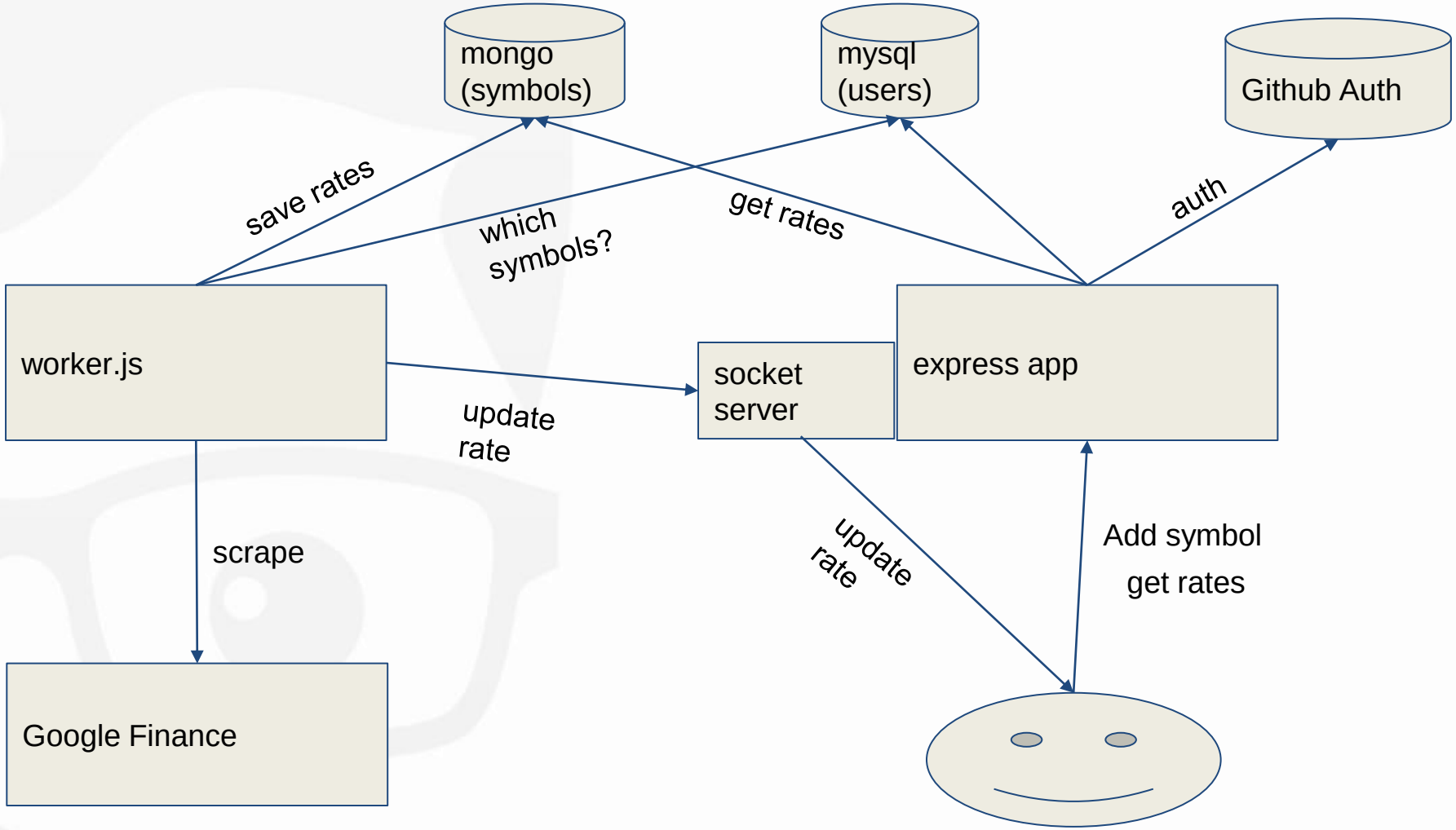
views:

dashboard:



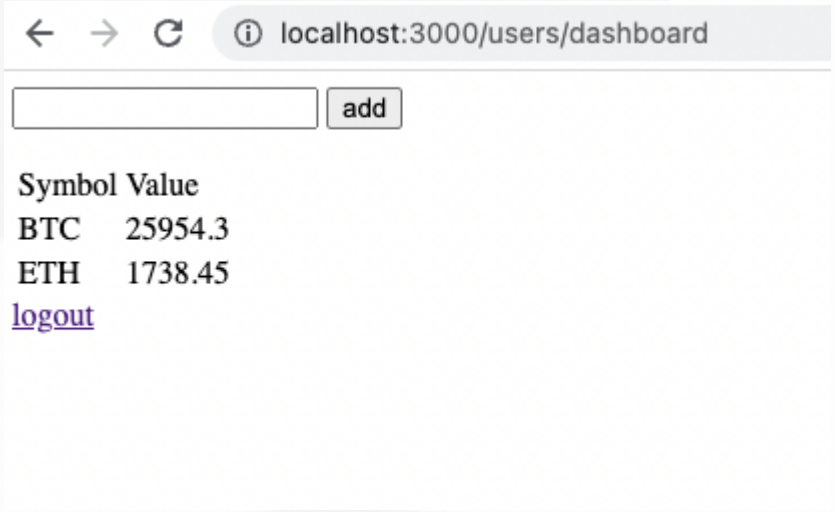
0x | A crypto dashboard app

Architecture:



0x | A crypto dashboard app

Pardon my UI skills:

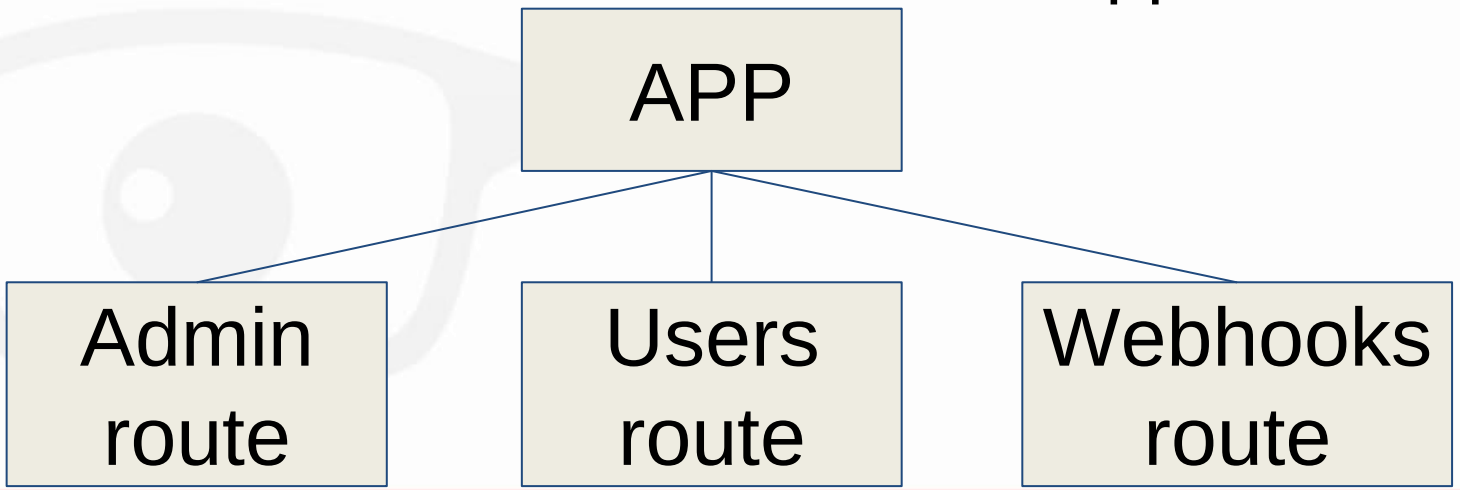


07 | express Routing

07 | express Routing

Routing:

- Route is a “mini app”
- Enables granularity
- More control
- Code readability
- Note: each route can also be an app…



07 | express Routing

Real world example:

```
const morgan = require('morgan');
const cartsRouter = require('./routes/carts');
const shopsRouter = require('./routes/shops');
const clicksRouter = require('./routes/clicks');
const db = require('./models');
const sequelizeTransaction = require('./middlewares/seque
const errorHandler = require('./middlewares/error-handler
const notFoundHandler = require('./middlewares/404');

const app = express();

// config
const port = config.get('app.port') || 3000;
const host = config.get('app.host') || 'localhost';

app.use(sequelizeTransaction);

app.use(morgan('combined'));

// input setup
app.use(express.json());
app.use(express.urlencoded({ extended: false }));

// route setup
app.use('/carts', cartsRouter);
app.use('/shops', shopsRouter);
app.use('/clicks', clicksRouter);
```

07 | express Routing

Basic router

```
const express = require('express');
const router = express.Router();

const healthcheck = async (req, res) => {
  res.send('I\'m healthy');
};

router.get('/healthcheck', healthcheck);

module.exports = router;
```

07 | express Routing

Routers and middlewares

```
const express = require('express');
const router = express.Router();
const someMiddleware = require('./middlewares/some-middleware');

const healthcheck = async (req, res) => {
  res.send('I\'m healthy');
};

router.use(someMiddleware) // will apply to all routes from here
router.get('/healthcheck', healthcheck);

module.exports = router;
```

07 | express Routing

Routers and middlewares

```
const express = require('express');
const router = express.Router();
const someMiddleware = require('./middlewares/some-middleware');

const healthcheck = async (req, res) => {
  res.send('I\'m healthy');
};

// will apply only to this GET routing
router.get('/healthcheck', someMiddleware, healthcheck);

module.exports = router;
```

07 | express Routing

Routers and middlewares

```
const express = require('express');
const router = express.Router();
const someMiddleware = require('./middlewares/some-middleware');
const someMiddleware2 = require('./middlewares/some-middleware2');
const someMiddleware3 = require('./middlewares/some-middleware3');

const healthcheck = async (req, res) => {
  res.send('I\'m healthy');
};

// will apply only to this GET routing
router.get('/healthcheck', someMiddleware, someMiddleware2, someMiddleware3, healthcheck)

module.exports = router;
```

07 | express Routing

Routing [also] means routing the request along the middlewares chain…

```
const express = require('express');
const router = express.Router();
const someMiddleware = require('./middlewares/some-middleware');
const someMiddleware2 = require('./middlewares/some-middleware2');
const someMiddleware3 = require('./middlewares/some-middleware3');

const healthcheck = async (req, res) => {
  res.send('I\'m healthy');
};

// will apply only to this GET routing
router.get('/healthcheck', someMiddleware, someMiddleware2, someMiddleware3, healthcheck)

module.exports = router;
```

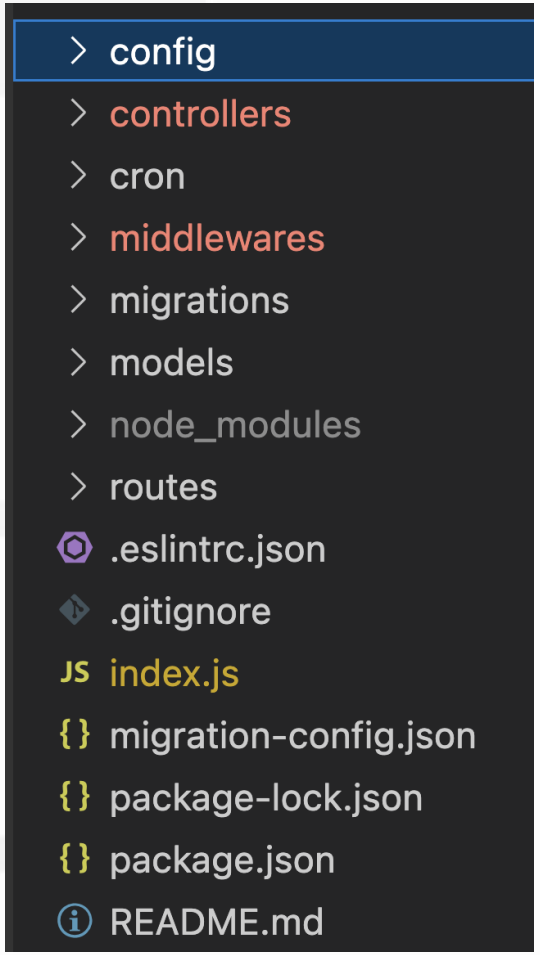
07 | express Routing

[My] recommended app structure:

|- config
|- controllers (contain the backend logic)
|- middlewares
|- routes
|- views (we'll learn that soon enough ;-), useless for API servers)
|- models
app.js
.gitignore (ignore node_modules/)
package.json (describe app)
README.md (detailed instructions for colleagues how to start the app)

07 | express Routing

[My] recommended app structure real world example:



07 | express Routing

Routing exercise:

1. create an app
2. create 2 routers
    a. admin
        i. /dashboard
        ii. must be authenticated
        iii. log each request to admin.log
    b. guests
        i. /welcome
        ii. /dashboard
        iii. use session
        iv. log each request to guests.log

07 | express Routing

Routing exercise:

1. create an app
2. create 2 routers
    a. users
        i. GET /welcome (homepage for guests)
        ii. GET /dashboard (homepage for users)
        iii. GET /logout (ends with a redirect to /welcome)
        iv. POST /symbol (adds a symbol user wants to follow)
    b. github
        i. / (will authenticate with Github)
        ii. /callback (github will use to pass auth data)

08 | express Input Validation

08 | express Input Validation

We'll use JOI.

- npm install joi

08 | express Input Validation

we have this controller function:

```
const removeFromCart = async (req, res, next) => {
```

08 | express Input Validation

we create this validator:

```
const removeFromCartValidator = Joi.object({
  shopProductId: Joi.string().required(),
  shopCartId: Joi.string().required(),
});
```

08 | express Input Validation

Joi middleware

```
module.exports = (validator) => async (req, res, next) => {
  try {
    const validated = await validator.validateAsync(req.body);
    req.body = validated;
    return next();
  } catch (err) {
    //* Pass err to next
    //! If validation error occurs call next with HTTP 422. Otherwise HTTP 500
    if (err.isJoi) { return next(createHttpError(422, { message: err.message })); }
    return next(createHttpError(500));
  }
};
```

08 | express Input Validation

Putting it all together

```
router.delete('/:cart_id', auth, joi(removeFromCartValidator), removeFromCart);
```

08 | express Input Validation

Exercise

- create a POST /user endpoint
- validate
  - name required
  - email required and valid
  - birthdate required and valid date
- use https://joi.dev/api/?v=17.8.3 for reference

08 | express Input Validation

Exercise

- create a POST /symbol endpoint
- validate
  - symbol
    - string
    - required
    - exactly 3 characters long
    - alphanumeric
    - upper case
- use https://joi.dev/api/?v=17.8.3 for reference

09 | express Views

09 | express Views

Views.

If you’re developing an API, you don’t need views.

You simply:

```
res.json(data);

// this equals

res.setHeader('Content-type', 'application/json');
res.send(data);
```

09 | express Views

Otherwise, if you insist on SSR:

```
// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');
```

09 | express Views

Prepare your templates

```
<table class="table">
    <tr>
        <th>id</th>
        <th>status</th>
        <th>description</th>
    </tr>
    <% tasks.forEach((task) => { %>
        <tr>
            <td><%=task.id%></td>
            <td><%=task.status%></td>
            <td><%=task.description%></td>
        </tr>
    <% }) %>
</table>
```

09 | express Views

And render them

```
const tasks = await getTasks();
res.render('tasks/list', {
  tasks
});
```

09 | express Views

Views exercise:

- create two endpoints:
  - /welcome
    - contains a “connect with Github” button/link
  - /dashboard
    - contains input to add new symbol
    - contains userSymbols table

10 | express Config

10 | express Config

- Two popular solutions
  - config
  - dotenv
- we'll use config
- npm install config

10 | express Config

- Config demo

10 | express Config

- Config exercise
  - github
    - clientId
    - secret
  - mysql
    - host
    - user
    - password
    - database
    - port
  - mongo
    - host
    - port
    - db

11 | MySQL with Node.js

11 | MySQL with Node.js

Let’s start a local MySQL server

```yaml
version: '3.8'

volumes:
  mysql_data:
    driver: local

services:
  mysql:
    image: mysql:latest
    volumes:
      - mysql_data:/var/lib/mysql
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: mydb
      MYSQL_USER: username
      MYSQL_PASSWORD: password
      MYSQL_TCP_PORT: 3306
    ports:
    - 3306:3306
    expose:
      - 3306
```

11 | MySQL with Node.js

Install MySQL npm module

> npm install mysql2

(mysql2 is simpler than mysql…)

11 | MySQL with Node.js

Let’s connect to MySQL from node

```
// hints
mysql.createConnection({
    host,
    user,
    password,
});

connection.connect(callbackFunction)
```

11 | MySQL with Node.js

Callbacks are so 2009…

Let’s promisify them.

```
// hints
const util = require('util');
someCallbackFunction = util.promisify(someCallbackFunction);
await someCallbackFunction();
```

11 | MySQL with Node.js

or… we can use… ;-) but we won't for this course.

```
const mysql = require('mysql2/promise');
```

11 | MySQL with Node.js

Create a users table

```
CREATE TABLE users (
      id int auto_increment,
      username varchar(255) not null,
      primary key (id)
    )

CREATE TABLE users_symbols (
      id int auto_increment,
          user_id int not null,
     symbol varchar(3) not null
      primary key (id)
    )
```

```
// hint

connection.query(`
  some SQL query
`);
```

11 | MySQL with Node.js

insert a user

11 | MySQL with Node.js

insert two users in a row.

11 | MySQL with Node.js

What were we missing?

Transaction

11 | MySQL with Node.js

Insert two users in a row within a transaction.

demonstrate:

- a rollback
- a commit

11 | MySQL with Node.js

Should we use connection pooling in express apps?

Think carefully.

11 | MySQL with Node.js

Connection pooling

```
const mysql = require('mysql2');
const util = require('util');

const pool = mysql.createPool({
  host: "localhost",
  user: "username",
  password: "password",
  database: 'mydb',
  waitForConnections: true,
  connectionLimit: 10,
  maxIdle: 10,
  idleTimeout: 60000,
  queueLimit: 0,
});

pool.query = util.promisify(pool.query);

(async () => {
  try {
    const result = await pool.query(`
      select * from users;
    `)
    console.log(result);
  } catch (e) {
    console.log(e);
  }
})();
```

11 | MySQL with Node.js

Build an express middleware that

- connects to MySQL
- starts a transaction
- commit on success
- rollback on failure

12 | MongoDB with Node.js

12 | MongoDB Node.js

- MongoDB is a document nosql database
- documents are stored in collections
- collections are schemaless
- yet fully indexable
- Working with MongoDB, node server and javascript client was the original “FullStack” stack. JS from db to server to client.

12 | MongoDB Node.js

- Start a local mongo server:

```
docker run --name mongodb -e
MONGO_INITDB_DATABASE=mymongo -d -p
27017:27017 mongo:latest
```

12 | MongoDB Node.js

- We’ll work with mongoose.
- npm install mongoose
- let’s connect

```
// hint
mongoose.connect('mongodb://127.0.0.1:27017/mymongo')
```

12 | MongoDB Node.js

Let’s define a user schema:

```
{
    name: {
        first: String,
        last: String,
    },
    email: String,
    birthday: Date,
}
```

```
// hint
new mongoose.Schema(SchemaObject)
```

12 | MongoDB Node.js

Let’s define a model derived from this schema:

```
// hint
mongoose.model('Model Name', schema);
```

12 | MongoDB Node.js

Now let's CRUD!

- create a user and save it

```
// hint
const instance = new SomeModel(modelData);
await instance.save();
```

12 | MongoDB Node.js

Now let’s CRUD!

- Let’s read the user we just created

```
// hint
const instance = await SomeModel.findOne(filterObject)
```

12 | MongoDB Node.js

Now let’s CRUD!

- Let’s read all users

```
// hint
const instance = await SomeModel.find(filterObject)
```

12 | MongoDB Node.js

Now let's CRUD!

- Let's update a user

```
// hint
const instance = await SomeModel.findOne(filterObject)
instance.some?.inner?.property = 'whatever';
await instance.save();
```

12 | MongoDB Node.js

Now let's CRUD!

- Let's remove the user

```
// hint
await Model.deleteOne(filterObj)
// or
await Model.deleteMany(filterObj)
// note
await instance.remove() // is deprecated
```

13 | User authentication with Node.js

13 | User Authentication with Node.js

Industry standard: passportjs.

13 | User Authentication with Node.js

Auth middleware

```
const passport = require('passport');
const LocalStrategy = require('passport-local').Strategy;

passport.use(new LocalStrategy({
        usernameField: 'email',
        passwordField: 'password',
    },
    async (email, password, done) => {
        try {
            // in reality we will fetch from db according to email
            const user = {
                email: 'shahar@johnbryce.co.il',
                password: '12345678',
            }
            if (!user) {
                return done(null, false, { message: 'Incorrect username/password.' });
            }

            // in reality we'll compare hashed+salted password to saved hashed+salted password
            if (user.password !== password) {
                return done(null, false, { message: 'Incorrect username/password.' });
            }
            return done(null, user);
        } catch (err) {
            return done(err);
        }
    }
));

passport.serializeUser((user, done) => {
    done(null, user);
});

passport.deserializeUser((user, done) => {
    done(null, user);
});

module.exports = passport;
```

13 | User Authentication with Node.js

App using our auth middleware

```
const path = require('path');
const auth = require('./auth');
const app = express()
const port = 3000
const host = 'localhost';

app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.use(express.json());
app.use(express.urlencoded({ extended: false }));
app.use(cookieParser());
app.use(session({
    secret: 'secret',
    resave: false,
    saveUninitialized: false,
    cookie: {
      maxAge: 1000 * 60 * 60 * 24 * 365 * 5,
    },
  }));

app.use(auth.initialize());
app.use(auth.session());

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

app.get('/dashboard', (req, res) => {
res.send('Dashboard')
})

app.get('/login', (req, res) => {
    res.render('index');
})

app.post('/login', auth.authenticate('local', {
    successRedirect: '/dashboard',
    failureRedirect: '/login',
}));

app.listen(port, host, () => {
```

13 | User Authentication with Node.js

Passport exercise:

- implement /signup and /login using either mysql or mongodb
- when the user is logged in, show a welcome {email} message on the /dashboard page

```
// hint
req.user // <- this is where passport saves the logged in user record...
```

13 | User Authentication with Node.js

Passport exercise:

- implement /signup and /login using mysql and github as auth provider (https://www.passportjs.org/packages/passport-github2/)
- when the user is logged in, show a welcome {email} message on the /dashboard page

```
// hint
req.user // <- this is where passport saves the logged in user record...
```

14 | Real Time with Node.js

14 | Real Time with Node.js

The WebSocket API (WebSockets)

The **WebSocket API** is an advanced technology that makes it possible to open a two-way interactive communication session between the user's browser and a server. With this API, you can send messages to a server and receive event-driven responses without having to poll the server for a reply.

14 | Real Time with Node.js

We'll use socket.io

- More than just websockets framework
- enables fallback to other methods (polling)
- cluster enabled

14 | Real Time with Node.js

socket.io server

- npm install socket.io

14 | Real Time with Node.js

basic server

```
const express = require('express');
const app = express();
const http = require('http');
const server = http.createServer(app);
const { Server } = require("socket.io");
const io = new Server(server);
const path = require('path');

// view engine setup
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', 'ejs');

app.get('/', (req, res) => {
  res.render('index');
});

io.on('connection', (socket) => {
  console.log('a user connected');
});

server.listen(3000, () => {
  console.log('listening on *:3000');
});
```

14 | Real Time with Node.js

basic client

```
<html>
    <head>
        <script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
        <script>
            var socket = io();
        </script>
    </head>
    <body>
        <p id="main">
            hello socket.io
        </p>
    </body>
</html>
```

14 | Real Time with Node.js

server disconnect aware

```
io.on('connection', (socket) => {
  console.log('a user connected');
  socket.on('disconnect', () => {
    console.log('user disconnected');
  });
});
```

14 | Real Time with Node.js

server emit events to clients

```
io.on('connection', (socket) => {
  console.log('a user connected');
  io.emit('my custom event',{name: 'admin'});
  socket.on('disconnect', () => {
    console.log('user disconnected');
  });
});
```

14 | Real Time with Node.js

client receives emitted events from server

```
<html>
  <head>
    <script src="https://cdn.socket.io/4.5.4/socket.io.min.js"></script>
    <script>
      var socket = io();
      socket.on('my custom event', function(msg) {
        document.getElementById('main').innerHTML = msg.name;
      });
    </script>
  </head>
  <body>
      <p id="main">
          hello socket.io
      </p>
  </body>
</html>
```

14 | Real Time with Node.js

Exercise:

- an express app with a single GET /
- returns a socket aware page with a placeholder <p> tag
- scrape https://www.google.com/search?q=bitcoin+to+usd regularly (each 10 secs) to get the current BTC rate

```
const cheerio = require('cheerio');
const $ = cheerio.load(anHTMLString); // get this with axios
// now you can use $ jQuery style
const btcRate = $(selector).text();
```

15 | Testing

15 | Testing

```
{
  "name": "bookstore",
  "version": "1.0.0",
  "description": "A bookstore API",
  "main": "server.js",
  "author": "Sam",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.15.1",
    "config": "^1.20.1",
    "express": "^4.13.4",
    "mongoose": "^4.4.15",
    "morgan": "^1.7.0"
  },
  "devDependencies": {
    "chai": "^3.5.0",
    "chai-http": "^2.0.1",
    "mocha": "^2.4.5"
  },
  Debug
  "scripts": {
    "start": "SET NODE_ENV=dev && node server.js",
    "test": "mocha --timeout 10000"
  }
}
```

15 | Testing

```
-- controllers
---- models
------ book.js
---- routes
------ book.js
-- config
---- default.json
---- dev.json
---- test.json
-- test
---- book.js
package.json
server.json
```

15 | Testing

```
//During the test the env variable is set to test
process.env.NODE_ENV = 'test';

let mongoose = require("mongoose");
let Book = require('../app/models/book');

//Require the dev-dependencies
let chai = require('chai');
let chaiHttp = require('chai-http');
let server = require('../server');
let should = chai.should();


chai.use(chaiHttp);
//Our parent block
describe('Books', () => {
    beforeEach((done) => { //Before each test we empty the database
        Book.remove({}, (err) => {
           done();
        });
    });
/*
  * Test the /GET route
  */
  describe('/GET book', () => {
      it('it should GET all the books', (done) => {
        chai.request(server)
            .get('/book')
            .end((err, res) => {
                  res.should.have.status(200);
                  res.body.should.be.a('array');
                  res.body.length.should.be.eql(0);
              done();
            });
      });
  });

});
```

15 | Testing

```
/*
* Test the /POST route
*/
describe('/POST book', () => {
    it('it should not POST a book without pages field', (done) => {
        let book = {
            title: "The Lord of the Rings",
            author: "J.R.R. Tolkien",
            year: 1954
        }
      chai.request(server)
          .post('/book')
          .send(book)
          .end((err, res) => {
                res.should.have.status(422);
            done();
          });
    });
    it('it should POST a book ', (done) => {
        let book = {
            title: "The Lord of the Rings",
            author: "J.R.R. Tolkien",
            year: 1954,
            pages: 1170
        }
      chai.request(server)
          .post('/book')
          .send(book)
          .end((err, res) => {
                res.should.have.status(200);
                res.body.should.be.a('object');
                res.body.should.have.property('message').eql('Book successfully added!');
                res.body.book.should.have.property('title');
                res.body.book.should.have.property('author');
                res.body.book.should.have.property('pages');
                res.body.book.should.have.property('year');
            done();
          });
    });
  });
});
```

15 | Testing

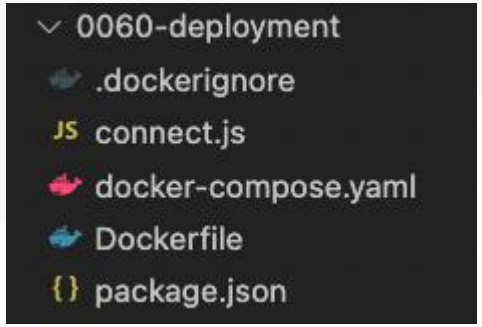
Exercise:

- test the passport app
  - signup
    - missing params
    - successful signup
  - login
    - wrong username/password
    - successful login
  - dashboard
    - verify the “welcome {email}” is rendered

16 | Deployment

16 | Deployment

Let’s deploy this demo app:



16 | Deployment

The js file:

0060-deployment > JS connect.js > ...

```
1   const mysql = require('mysql2');
2
3   const connection = mysql.createConnection({
4     host: process.env.DB_HOST,
5     user: process.env.DB_USER,
6     password: process.env.DB_PASSWORD,
7     database: process.env.DB_DATABASE,
8     port: process.env.DB_PORT,
9   });
10
11  connection.connect(function(err) {
12    if (err) throw err;
13    console.log("Connected!");
14  });
15
```

16 | Deployment

The package.json file:

0060-deployment > {} package.json > ...

```
1  {
2    "dependencies": {
3      "mysql2": "^3.2.0"
4    }
5  }
6
```

16 | Deployment

The Dockerfile:

0060-deployment > Dockerfile

```
1   FROM node:14
2
3   WORKDIR ./
4   COPY package.json .
5   RUN npm install
6   COPY . .
7   CMD node connect.js
```

16 | Deployment

The .dockerignore file:

0060-deployment > .dockerignore

```
1    node_modules
```

16 | Deployment

The docker-compose file:

0060-deployment > docker-compose.yaml

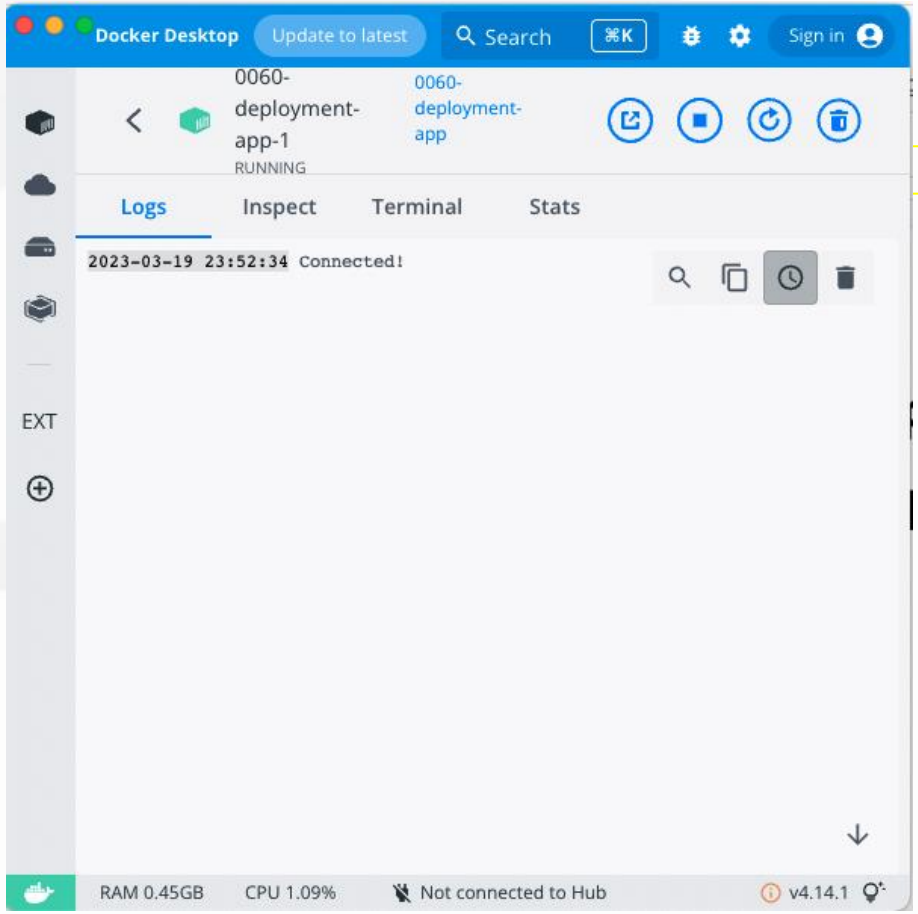
```yaml

services:
  mysql:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: password
      MYSQL_DATABASE: mydb
      MYSQL_USER: username
      MYSQL_PASSWORD: password
      MYSQL_TCP_PORT: 3306
    ports:
    - 3306:3306
    expose:
      - 3306
    restart: unless-stopped
    volumes:
      - db:/var/lib/mysql
    healthcheck:
      test: ["CMD", "mysqladmin" ,"ping", "-h", "localhost"]
      timeout: 20s
      retries: 10
  app:
    depends_on:
      mysql:
        condition: service_healthy
    build: ./
    restart: unless-stopped
    environment:
      - DB_HOST=mysql
      - DB_USER=username
      - DB_PASSWORD=password
      - DB_NAME=mydb
      - DB_PORT=3306
    stdin_open: true
    tty: true

volumes:
  db:
```

16 | Deployment

The result:



16 | Deployment

Exercise:

- Deploy an express app that connects to mysql (the passport one is a good candidate…)
- show it works by manipulating data online

17 | Microservices

17 | Microservices

What’s wrong with Monolithic apps

17 | Microservices

What's good about Monolithic apps

- Easy deployment.
- Easy e2e testing.
- Collateral functionalities like logging, caching, security etc. are implemented once.
- Performance advantage as parts of the system interact directly with each other, with no network latency.

17 | Microservices

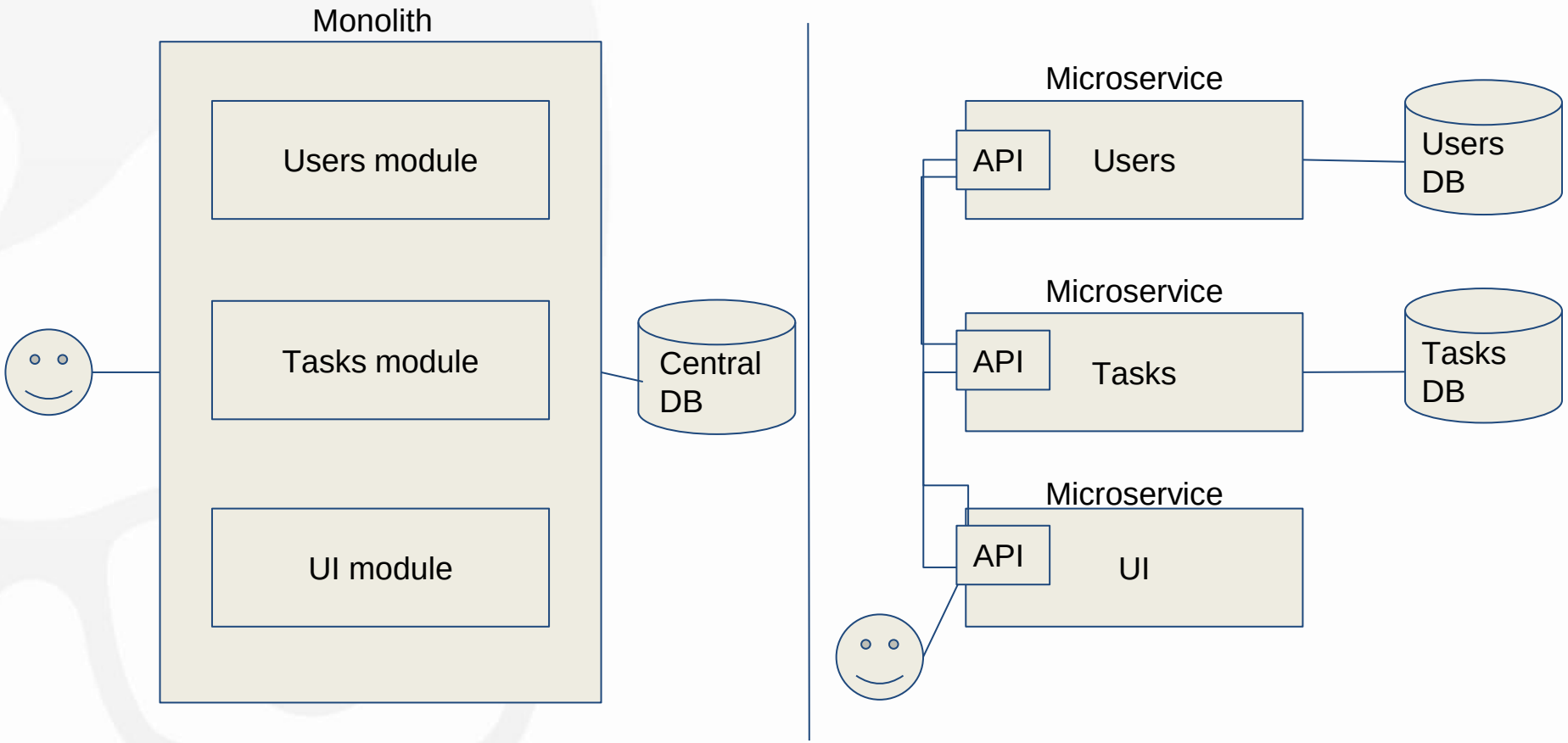
What's wrong with Monolithic apps

- Development and maintenance become ever more complicated as code base grows.
- CI/CD (mostly CD) is hard to achieve.
- Difficult to scale.
- One bug can crash the entire system (tight coupling)
- Adoption of new tech is hard.

17 | Microservices

Welcome Microservices

17 | Microservices



17 | Microservices

Microservices

Architectural style that structures an application as a collection of services that are

- Highly maintainable and testable
- Loosely coupled
- Independently deployable
- Organized around business capabilities
- Owned by a small team

17 | Microservices

Microservices Benefits

- Faster to develop.
- Easier to understand and maintain.
- Teams are more focused.
- Developers experience many types of envs and languages.
- Each service release is independent of the others.
- Scale is granular.

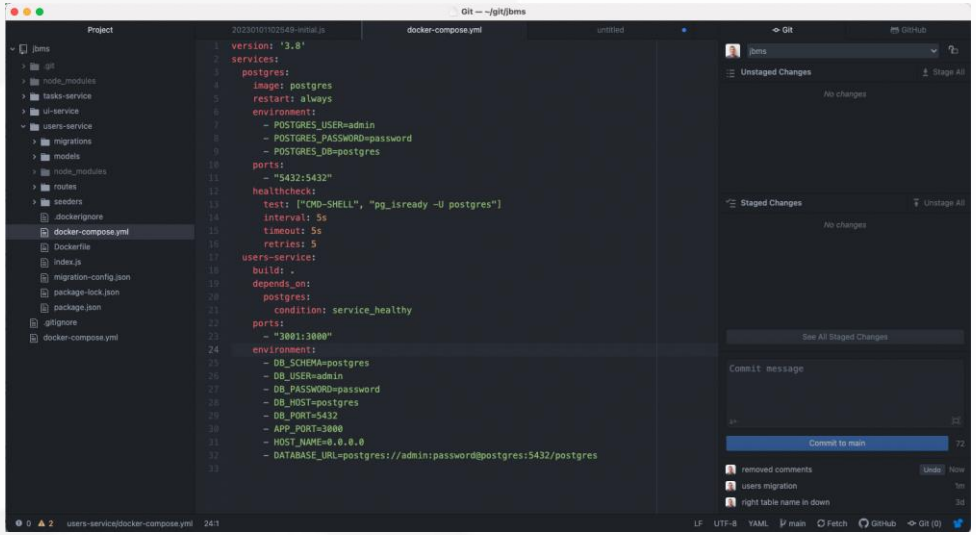
17 | Microservices
Microservices Challenges

- Handle partial failures and unavailability of upstream services.
- Distributed transactions
- Harder e2e testing
- Deployment is complex
- Service Discovery
- Ops/DevOps overhead
- Inter service communication results in network latency

Demo

17 | Microservices

git clone git@github.com:shaharsol/jbms.git



17 | Microservices

Exercise:

Break the passport app into a microservices app:

- users service
- ui service

18 | TypeScript

18 | Typescript

Folder Structure

18 | Typescript

Package.json

```
{
  "devDependencies": {
    "@tsconfig/node16": "^1.0.3",
    "@types/node": "^18.15.5",
    "@typescript-eslint/eslint-plugin": "^5.56.0",
    "@typescript-eslint/parser": "^5.56.0",
    "eslint": "^8.36.0",
    "typescript": "^5.0.2"
  }
}
```

18 | Typescript

tsconfig.json

```
{
  "$schema": "https://json.schemastore.org/tsconfig",
  "display": "Node 16",
  "extends": "@tsconfig/node16/tsconfig.json",
  "compilerOptions": {
    "lib": ["es2021"],
    "module": "commonjs",
    "target": "es2021",
    "outDir": ".dist/",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true,
    "moduleResolution": "node"
  },
  "include": ["src/**/*"],
  "exclude": ["node_modules", "**/*.spec.ts"]
}
```

18 | Typescript

cli.ts

```ts
const name = 'Shahar';
const aNumberAge = 24;
const aStringAge = '24';

type Person = {
    name: string,
    age: number,
};

const person: Person = {
    name,
    age: aNumberAge
};

console.log(person);
```

18 | Typescript

cli.ts

```
const name = 'Shahar';
const aNumberAge = 24;
const aStringAge = '24';

type
};

cons
    age: aStringAge
};

console.log(person);
```

```
(property) age: number

Type 'string' is not assignable to type 'number'. ts(2322)

cli.ts(7, 5): The expected type comes from property 'age' which is
declared here on type 'Person'

View Problem (⌥F8)    No quick fixes available
```

18 | Typescript

Exercise:

- You guessed it right. Turn the passport app into a TypeScript app.
- hints:
  - You'll need to add many @types/xxx to your dev dependencies
  - You'll need to type each constant, variable and parameter

19 | Miscs

19 | Miscs

- publishing NPM packages
- debugging

01 | Introduction to Node.js

TypeScript

npm install –save-dev typescript

or globally: npm install -g typescript

matrix

JOHN BRYCE
תלמדו הייטק. זה עובד!
a matrix company